The design and implementation of an interface box supporting SIP 1.0 commands

Citation for published version (APA):
Kouwenberg, R. F. W. M. (1991). *The design and implementation of an interface box supporting SIP 1.0 commands*. (IPO rapport; Vol. 835). Instituut voor Perceptie Onderzoek (IPO).

Document status and date:
Published: 11/12/1991

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Institute for Perception Research
P.O. Box 513 - 5600 MB Eindhoven

11.12.1991

Rapport no. 835

The design and implementation of an interface box supporting SIP 1.0 commands

R.F.W.M. Kouwenberg

summary

A new software protocol has been developed by the Institute for Perception Reserach (IPO), which can be used to communicate between a host computer and an input-output device by means of an rs232 channel: the Serial Interface Protocol. This protocol [Eggen1991] describes a number of command tokens with which it is possible to send and receive input-output control commands.

The inputs and outputs are controlled by a so called 'interface box', which executes commands received from the central host computer, controls the outputs and monitors the status of input devices. The input and output devices to be attached can range from simple switches and leds to complex joggles.

This report describes a hardware and software test implementation of rudimentary Sip 1.0 commands in an interface box to monitor and control up to 64 inputs and outputs. To fully make use of the full sip 1.0 command protocol a software update is necessary.

1 Introduction

The research of the 'Instituut voor Perceptie Onderzoek' (IPO), is concerned with sensory and cognitive information processing and communication by humans interacting with flexible information systems.

A part of the research programme is occupied with the development of new consumer electronics. As such it's goal is to improve the human tending of consumer electronics.

Many prototypes of possible future products have to be tested for their suitability for the consumers. An important saving of time can be achieved by splitting the device in two parts. One part is the front panel which contains input devices (e.g. switches) and output devices (e.g. leds). The other part is the hardware machinery (e.g. the machinery of a DAT recorder). The functions of the hardware machinery can be simulated on a computer (e.g. a SUN). If one wants to test a new device, then only the front panel will have to be built for each different layout.

The front panel in the test facility will be controlled by a so called "interface box". This interface box will control the output devices and monitor the operations of a test person on the input devices of the front panel. The interface box will communicate with a host computer which will simulate the commands programmed by the test person. Further will the host computer send commands to the interface box to set the outputs of the interface box, placed on the front panel, to the appropriate display function.

With this set-up it is possible to gather information of the use of the device and whether e.g. the placement of an input at a different location on the front panel might improve the ergonomics of the device, while only having to rebuild the front panel for this test.

The main purpose is to develop a new interface for the testing of ergonomics of consumer electronics.

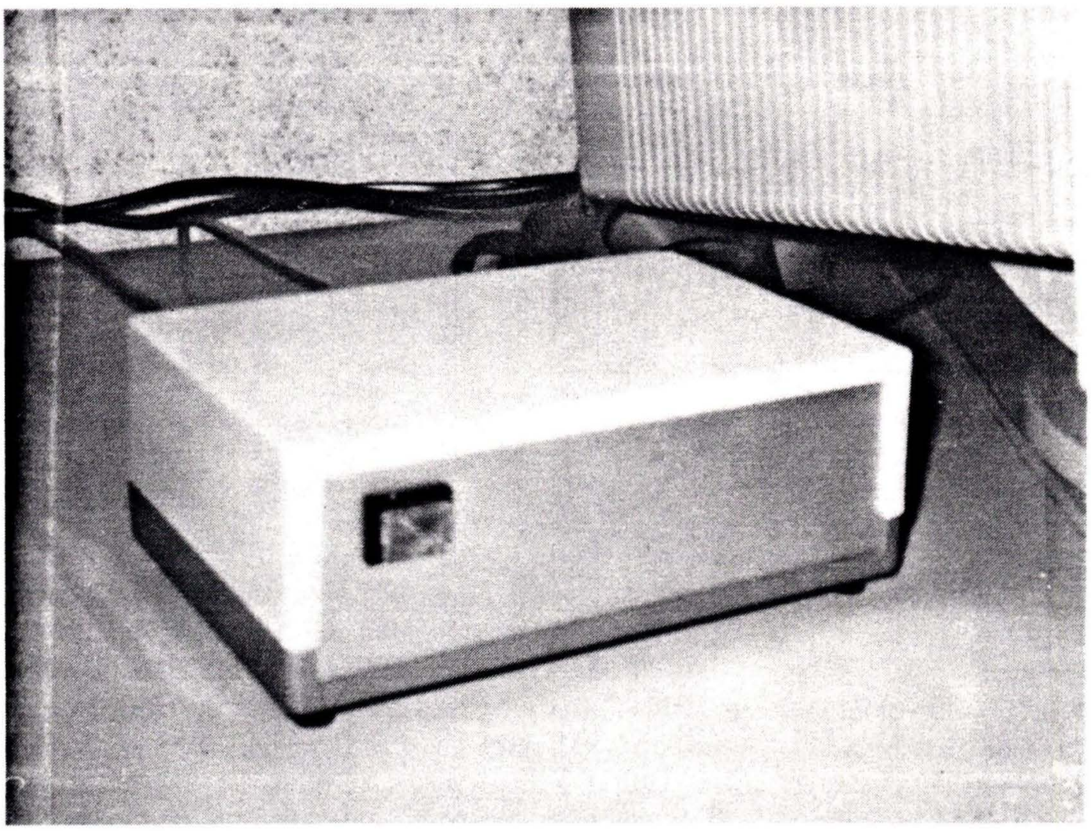
The interface box is to control a number of input and output devices. Possible input devices can range from simple switches to complex joggles. Output devices can range from leds to digital-analog converters.

The following chapters will describe the design and the layout of the software and hardware of the interface box.

2 Design of the interface box

Preceding the design of the interface box, a software command protocol had been developed [Eggen 1991], to be used to communicate between the interface box and the host computer. The specification of the serial communication link had also been written [Eggen 1991].

The serial interface protocol (SIP 1.0) describes a set of commands to be supported. The main goal of the research was to develop a software expandable implementation in hardware of SIP 1.0. The commands to be supported are : System_Reset, Output_Set, Input_On / Off, Input_Activate. In a future version further commands can be implemented. At least 32 inputs and outputs should be supported, where the specific 'kind' of input and output is to be easily interfaced with a minimum of extra hardware.
The interface box will be connected to a Sun computer; a specific serial connection is to be provided. As the interface box will be used for test simulations, situations of a specific action of the test person or a command sequence sent by the host computer, which would cause programm crashes are to be evaded. The interface box itself should also be solid, portable and (sofware) expandable.



3 Sip 1.0 commands supported

3.1 Introduction

The following paragraphs will describe the sip 1.0 commands that are supported by the interface box. To fully understand 'what's happening inside', one should read the next chapters.
Sip 1.0 commands consist of a command byte and a maximum of 2 databytes (the label commands can be larger, but are not implemented in this version of the interface box software).

A special provision is made in the software of the interface box to decrease the amount of information which has to cross the serial interface link: the running status. Whenever a valid command has been executed, the running status is set to the last executed command. Subsequent databytes can be sent by the host computer without having to repeat the same command byte. As such it is e.g. possible to set all 64 outputs to a specific value with 1 command byte and 128 databytes instead of 192 (64 times 1 command and 2 databytes) bytes: a saving of 63 bytes.

3.2 System_Reset

The system_reset command is invoked by sending #0FFH (255D) from the host computer to the box. After receiving this commands, all software parameters are set up as they were after a hardware (power on) reset. The software parameters are:

- all outputs off,
- all input transitions signalled,
- blinking time 640 ms,
- receive buffer empty.

3.3 Label_Exclusive

The interface box can't execute any label commands (= command bytes higher than #0F0H); these commands cause 'command not implemented' errors.

A complete label command consists of a label command (the 'begin'), an unknown number of bytes, not containing #0F0H (240D), followed by the end of label command byte: #0F0H. The problem of this command is (besides that it is not imlemented), that within the label command, all kinds of bytes might appear; the only restriction of the label message is that is does not contain the ELE byte. It would be possible that the interface box would interpret the bytes of the label command, resulting in unpredictable errors. To overcome this problem the interface box just ignores everything coming after a label command, untill it receives the ELE command byte, after which it sends a "Command_Not_Implemented" error to the host computer.

3.4 Output_Set

The command byte for output_set is #85H (133D), followed by 2 data bytes: first the output number, then the desired function. The output number has to be in the range of #0..3FH (0..63D), exceeding this range will result in a data byte error. The function byte following the sip 1.0 protocol can be one of four bytes:

- 00000000B output off
- 00000001B output on blinking
- 00000010B output on inverse blinking
- 01111111B output on continuous

No error checking is done on the function databyte; only the two righmost bits (LSB and LSB-1) are used.
The blinking time is set after a reset to 640 ms, however it can be set under software control to any 10ms multiple.
The interface box is not able to determine which of the outputs are in fact connected. It is however possible to adapt the software to accept only a smaller range of output numbers.

3.5 Output_All_Set

To set all the outputs at once one can use the Output_all_set command. The command byte is #0C7H (199D) followed by a function byte. The same remarks as above apply, except that this command sets all output to the desired function.

3.6 Input_Activate

After a hardware or software startup all the input transitions will be signalled. To change this the host computer can issue the Input_Activate command (#0CAH, 202D), followed by 2 databytes: inputnumber and function. The range for the inputnumber is #0..3FH (0..63D); exceeding this range will result in a data byte error. The function byte following the sip 1.0 protocol can be one of four bytes:

- 00000000B input on/off transition not signalled,
- 00000001B only input on transition signalled,
- 00000010B only input off transition signalled,
- 01111111B both input on and off signalled.

No error checking is done on the function databyte; only the two righmost bits (LSB and LSB-1) are used.

This command is especially useful when the host computer is not interested in the status of specific switches. To decrease the communication rate on the serial channel the signalling of trivial switches can be turned off or on.

3.7 Set_Blinking_Time

After a soft or hardware reset the blinking time is set to #40H times 10ms. With the Set_blinking_Time command (#0C1H / 193D), this time can be set by the host computer to a 10ms multiple in the range 0 - 2550ms.

Two databytes accompany the command:

- 0XXXXXXXB low blink time
- 0YYYYYYXB MSB of blink time

3.8 Input_On

The interface box can be used to monitor the status of up to 64 inputs. When an input switch is pressed the interface box signals the transition off to on, using the command byte #89H (137D, Input_on), together with 1 databyte: the inputnumber 0..3FH (0..63D) to the host computer. The way in which the inputs are multiplexed is drawn in appendix 3.
The host computer can set the status of the connected switches on or off so that not all transitions are signalled (see input_Activate).

3.9 Input_Off

When a switch is released the interface box generates the input off command (#88H/136D Input_off+1 byte input 0..3FH/63D), similar remarks can be made as the Input_On command.

3.10 Data_Byte_Error

The interface box is only able to process commands in a given range. The output commands can only be executed on a number of 0..63D. Exceeding this range will generate a Data_Byte_Error (#0C1H/193D).

3.11 Command_Not_Implemented

At this moment only the given above commands are implemented in the interface box. Further commands listed in the SIP 1.0 protocol may be implemented in the future. Commands not listed and label commands will generate a Command_Not_Implemented message (#0CFH/197D).
A command_Not_Implemented error will reset the running status of the interface box.

4 Hardware overview

4.1 Introduction

Within the Institute for Perception Research (IPO) many designs have been made with the 8031μp [Intel 1985]. Numerous designs incorporated this μp, an input matrix and an output matrix [e.g. Waterham 1989, Bierens 1989, Zelissen 1990], therefore a choice was made to use this processor. A circuitry based on this microprocessor can be devided in a processor part and an I/O circuitry part.

4.2 Processor circuitry

The processor circuitry is based on the 8031 μp which has the following specifications:

- 8-bit CPU
- On chip oscillator and clock circuitry
- 32 I/O lines
- 64 Kbyte address space for external data memory
- 64 Kbyte address space for external program memory
- two 16 bit timers/counters
- five-source interrupt structure with two priority levels
- full duplex serial port
- boolean processor

The circuitry is built up with a low address latch, memory (ROM and RAM), serial and I/O circuitry.

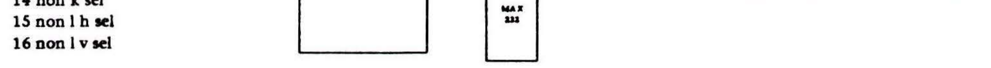


figure 4.2a processor print layout

The processor print (fig 4.2a) contains the microcontroller (Intel 8031), an address latch buffer (Philips 74HCT573), a program memory (Signetics 27C256), a data memory (Hitachi 6264) and a serial line buffer integrated circuit (Maxim MAX 232).

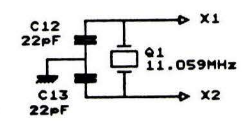


figure 4.2b clock circuitry

The microcontroller has an on chip clock oscillator. All that is needed externally are two capacitors and a crystal (fig 4.2b). A lot of different clock frequencies can be chosen, however the μP genarates it's serial data by dividing the clock frequency with a constant. For an accurate serial data rate of 19200 baud a crystal with a frequency of 11.059 MHz is to be chosen [Intel 1985]. The reset pulse circuit is simple (fig. 4.2c).

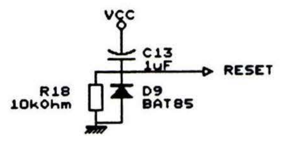


figure 4.2c reset circuitry

This combination of a diode, capacitor and resistor causes a hardware reset on the 8031 every time the power is restored after a power on, or a power failure. It is adapted from the microcontroller data book [Intel 1985], with a diode added to discharge the capacitor every time the power is removed, so that no negative source can damage the μP internals (the BAT85 is chosen for it's low voltage drop of 0.2V).

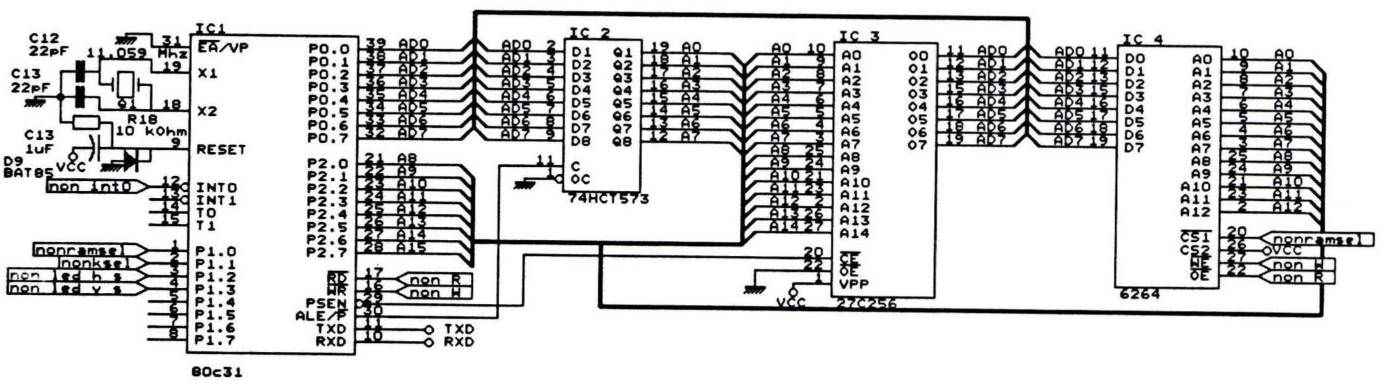


figure 4.2d microprocessor print circuitry

The total circuitry of the microprocessor print is drawn in figure 4.2d. The lower address lines (A0..7) are multiplexed with the data lines (D0..7) as AD0..7. An 8 bit latch (ic 2) buffers the address lines. The microprocessor pulls its ALE/nonP line high to make the latch read a new low address combination. In a next clock cycle the high address line and the databus appear correct on the ports. A program memory read is accompanied by a low PSEN. A data memory selection is accompanied by a low nonR or a low nonW signal. The actual selection of the RAM ic has to be done by making the 'nonramsel' line low. A selection of one of the 3 I/O latches or the I/O buffer can be made by selecting the appropriate control lines.

The program is placed in 32kbyte EPROM memory (ic 3), the data memory has an 8kbyte capacity (ic 4). The choice of a large memory space in proportion to the actual program space needed has been made is partly on cost aspect ratio (price per kilobyte), partly on future memory demands.

A complete microprocessor machine cycle consists of 12 clock cycles. This causes the time for the execution of every line of program code (a processor cycle) to take about 1 μS. Some operations can be done in 1 cycle others take more machine cycles.

The 8031 μp generates it's serial signal on TTL level (+5 and 0 volts). In the definition of RS-232-C[1], a logic '1' is represented by a voltage from +5 to +15 volt; a logic '0' by a voltage from -15 to -5 Volt. A special line driver, ic 9 MAX 232, is used to convert the +5 volts level to +10volts and 0 volts to -10 volts so that the serial output signal corresponds with the RS-232-C [McNamara 1977] definition. This IC is chosen because of it's capability of generating the positive and negative levels from a single 5 volts power supply. Of this ic only one of the two possible in/out combinations is used (fig. 4.2e).

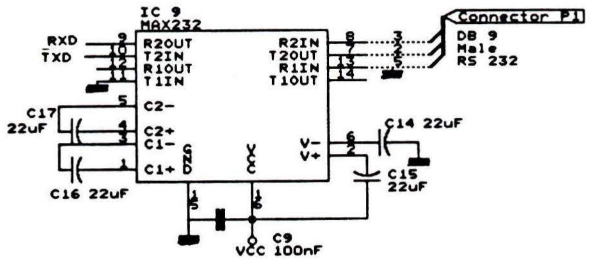


figure 4.2e RS-232-C interface

The databus, the 5 Volts and ground power supply, and 6 control lines are directed to theI/O print. The pin connections are listed in the appendices (App. 3).

[1] [Mc Namara 1977]

4.3 I/O circuitry

A quad exclusive OR ic, 74HCT02 (ic11), and a hex inverter, 74HCT04 (ic10) see to the correct level signals of select lines of the 4 I/O integrated circuits. The logic is written in table 4.3.

table 4.2 select lines arguments

ic nr.	argument
ic5	nonW XOR non led hor sel
ic6	nonW XOR non led ver sel
ic7	nonW XOR nonksel
ic8	NOT (nonksel XOR nonR)
nonint0	NOT (sw.row1 OR sw.row2 OR ... OR sw.row 8)

The interface box can control up to 64 outputs. The outputs are multiplexed in a 8*8 matrix, that is one row (driven by ic 6) at a time is pulled low, after which 8 lines are used to set the 8 selected outputs to the desired 'data' ('ON' equals a high level). The program is written to switch each vertical line, containing 8 horizontal outputs, 'ON' for approximately 1.25ms every 10ms, resulting in a 100Hz 'flicker' for the display. At this frequency it is hardly noticeable that the outputs are not on continuously. A resistor is placed internally to limit the current.

After a hard or a software reset, all outputs are off, that is the rows are still pulled low one at a time, but the lines are all low ('data' = 0).

The input matrix consists of a latch (74hct573) and a buffer (74hct244). Information is read by setting a latch row high and reading the information achieved by the buffer. A provision is made for future expansion to read the input matrix by means of a interrupt routine.

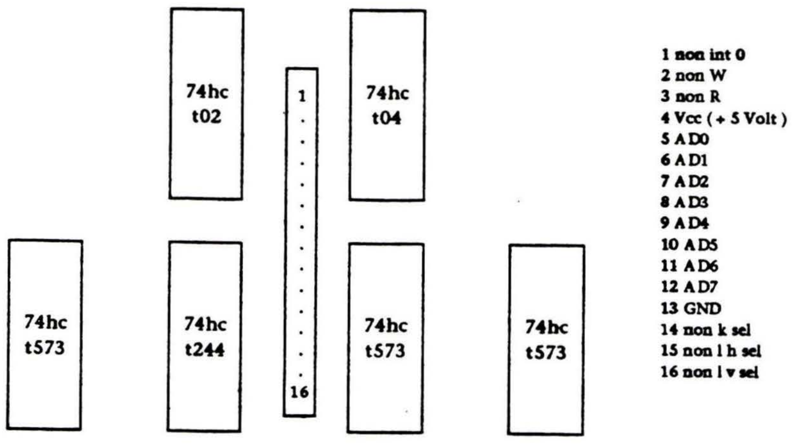


figure 4.3a I/O print circuitry layout

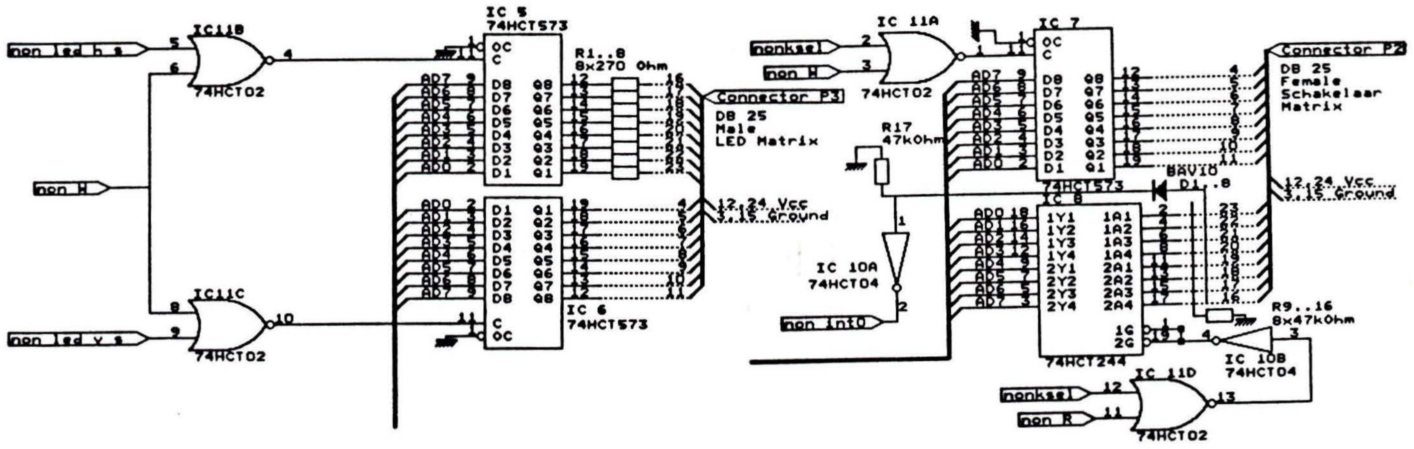


figure 4.3b I/O circuitry

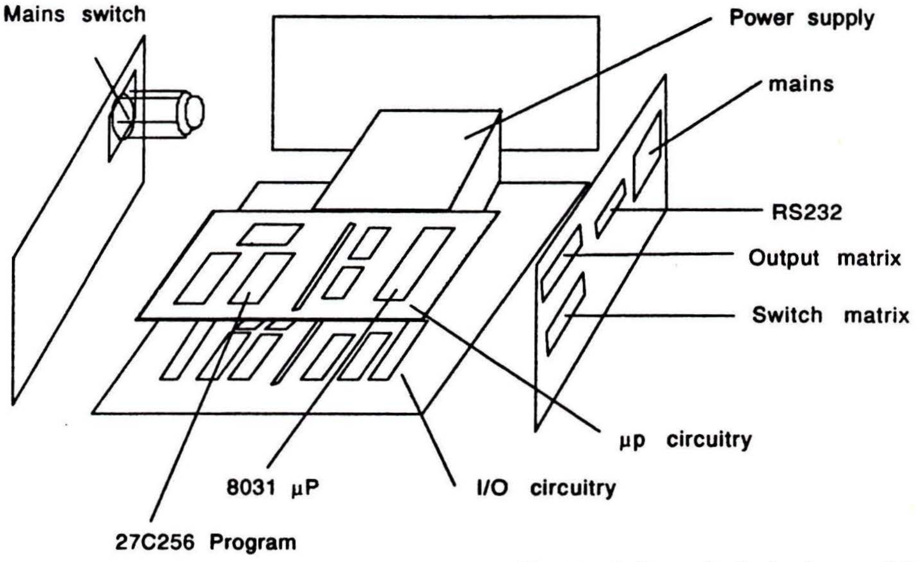


figure 4.3 exploded view of interface box

The case (fig. 4.3, dimensions 14.0*20.5*7.5cm) houses besides the processor and I/O circuitry a single 5 Volts/1 Ampère short-circuit protected power supply. The external connectors and circuitry are wired with flat cable. A connector is placed between the I/O and processor circuitry. This design is chosen to make the interface box more compact. The program can be updated easily by exchanging the eprom on the top print. A parts list and a circuit layout is provided in the appendices (App. 1,2).

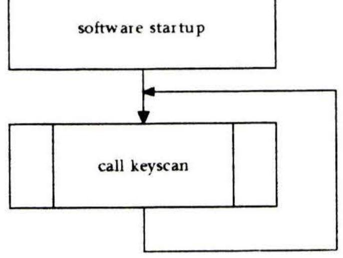
5 Software overview

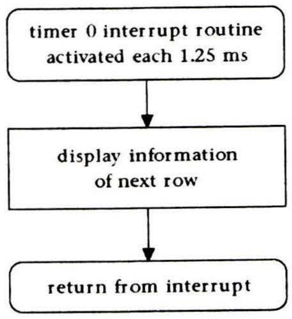
5.1 Introduction

After having built the hardware circuitry, the program software had to be written. At first the serial connection was tested by echoing bytes sent by a host computer back to the host computer and placing these received bytes in a buffer. By use of a 8031 in-circuit debugger the bytes in the buffer could be compared with the sent bytes. The serial routine worked.
A next subroutine was written to display two groups (blinking and inverse blinking) of 64 outputs after one another. The outputs are multiplexed in an 8 by 8 matrix, which means that only one line of 8 outputs can be displayed at a time. If all the outputs are displayed consecutively with a high frequency, an observant will not notice the difference between e.g. a led blinking with a frequency of 75Hz and a led that is on continuously. For timing reasons a display frequency of 100Hz is chosen[2], this results in an approximately 1.25 ms 'ON' and a 8.75 ms 'OFF' period of each output (12.5% dutycycle).

5.2 Program layout

Having written and tested the serial input and output routine and the display routine, the program was written. The program is built up with 3 routines (fig 5.2 a..c).





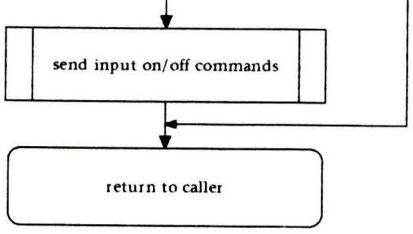


Figure 5.2a main programm Figure 5.2b Timer 0 routine Figure 5.2c Keyscan

[2] The SIP 1.0 defines time intervals in multiples of 10ms. If one chooses a multiplex rate with a display 'flicker' of 100Hz then each output can be displayed regardless how small the time interval.

The current program is built up with two interrupt routines: a serial in/out and a clock interrupt. The clock interrupt is set to be activated each 1.25ms, the serial interrupt will be activated whenever a byte is received or a byte has been completely sent. A hardware provision is made to make the reading of the input matrix on interrupt basis possible, however it is not implemented in this program. The input matrix is read in a subroutine which is called repeatedly from the main program.

table 5.2.1 interrupt vector declaration

name	8031 address vector	routine called
reset	0H	init
timr0int	0BH	do125ms
serint	23H	serint

Memory layout

The 8031 μp can address 128 bytes of internal memory. Certain locations of this address space is allocated to a specific part of the program. The remaining memory is used for the stack (table 5.2.2).

table 5.2.2 memory layout from 0H to 7FH

name of variable	address space	description
register 0..7	8H	internal 8031 register 0..7
blink	1H	blinking frequency in 10 ms low byte
blink2	1H	" " " " " high "
com	1H	last running status command received
complus1	1H	byte +1 of command
complus2	1H	byte +2 of command
countblink	1H	amount of scans done low byte
countblink2	1H	" " " " high "
ledgr	1H	blinkgroup current on matrix
t0ibb	1H	timer 0 int row backup
t0ibib	1H	timer 0 int line info pointer backup
ledgr1	8H	line infos of blink group
ledgr2	8H	" " " inverse blink group
input	18H	input table ; 00..07 newscan ; 08..0F oldscan / just pressed ; 10..17 just released
rbuffer	10H	receive buffer
andmatrix	10H	used for input_activate ; 00..07 input_on status ; 08..0F input_off status
begstack	17H	Stack from 69H to 7FH

5.3 Hardware startup routine 'init'

A power on, or a restart after a power failure causes the execution of the program code which is pointed to by the reset vector (table 5.2.1). The reset vector points to the hardware startup routine (*init*). This routine executes a number of commands:

- set the mode of timers 0 and 1,
- set the baud rate using timer 1,
- set the mode of the serial port,
- start timers 0 and 1.

The hardware startup continues with the execution of the commands of the software startup (*init2*):

- set interrupt flags of timer 0,1,serial port ON,
- set begin of stack,
- clear 8031 internal RAM,
- set table input on/off to all input transitions signalled,
- set receive buffer pointer to no bytes buffered,
- set the blink rate,
- set the first line to be displayed,
- set normal blinking group as the group that is first displayed.

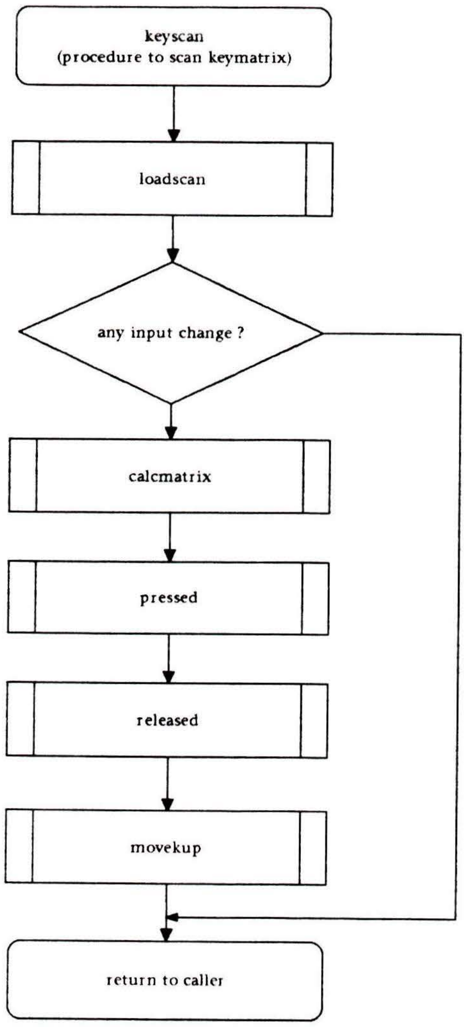
5.4 Main program

The main program consists of an endless loop in which the input scan routine (*keyscan*) is called.

The bytes sent to the interface box are received through interrupt. A transmit ready interrupt will free the serial channel so that a new byte can be sent to the host computer. Commands parsed to the interface box are interpreted and executed within the interrupt routine. With a baud rate of 19200 bps, approximately 570μs is available before the next byte will have been arrived completely. In this time interval all implemented commands can be interpreted and executed.

5.5 Subroutine keyscan

The subroutine *keyscan* uses 5 subroutines (*loadscan*, *calcmatrix*, *pressed*, *released*, *movekup*). *Loadscan* reads the input matrix. If no changes are detected then *keyscan* will return to caller (figure 5.5).



Calcmatrix is used to calculate the released and pressed switches. The subroutines *pressed* and *released* are used to signal changes in the status of the inputs to the host computer. A move routine which replaces the old with the new data completes the subroutine.

Subroutine loadscan

This subroutine reads the present status of the input matrix. If the same data appears as the last time the subroutine was called, then the subroutine exits with a signal that no change has taken place.
If a change has been detected then the routine tries to read n times the same result to avoid noise caused by rumbling. If it fails in this action, it tries to read the new scan n times, etc. If all's well then the routine returns to the caller.

Subroutine calcmatrix

This subroutine requires a new and an old scan of the input matrix.
Using two formulae (table 5.5) it calculates changes in the status of the inputs (released: change from 'ON' to 'OFF' and pressed vice versa).

Figure 5.5 Keyscan subroutine

table 5.5 expressions to calculate input changes

```
Pressed  = (old AND new) XRL new AND input_on  table
Released = (old XRL new) AND old AND input_off table
```

The routines returns to caller with the pressed and released input table. The new scan table has not been changed.

Subroutine pressed

Pressed walks through the pressed table, calculates the associated input number of set bits, and signals (using *sendbyte*) these changes from 'OFF' to 'ON' of every input to the host computer by using the command byte Input_On, followed by the input number. No changes are made to the input tables.

Subroutine released

The same remarks as above app4ly, except that the released table and the command 'Input_Off' are used.

Subroutine movekup

This subroutine moves the newscan table to the oldscan location.

Subroutine sendbyte

This subroutine sends a byte via the rs232 link to the host computer. The odd parity of the byte to be sent (contained in the Accumulator) is placed in the 9th bit register of the serial port (TB8). An internal loop is executed as long as a bit (f0) is set. This bit is set by software to indicate that a byte is being sent. F0 is reset after a serial send ready interrupt. The serial buffer of the μp (SBUF) is filled with the contents of the Acc and f0 is set. A return to caller completes the subroutine.

5.6 Subroutine interrupt timer 1

The timer 0 interrupt routine is restarted every 1.25 ms. It places the information of 8 outputs (1 each row) on one line. The routine checks if 8 lines have been displayed. If this is true, then it increases a counter. The counter is compared with the blink frequency. If they match then the pointer to the current blink group is inversed (from blinking to inverse blinking and vice versa). An output that is continuously on will be 'ON' in the blinking and the inverse blinking group (figure 5.6).

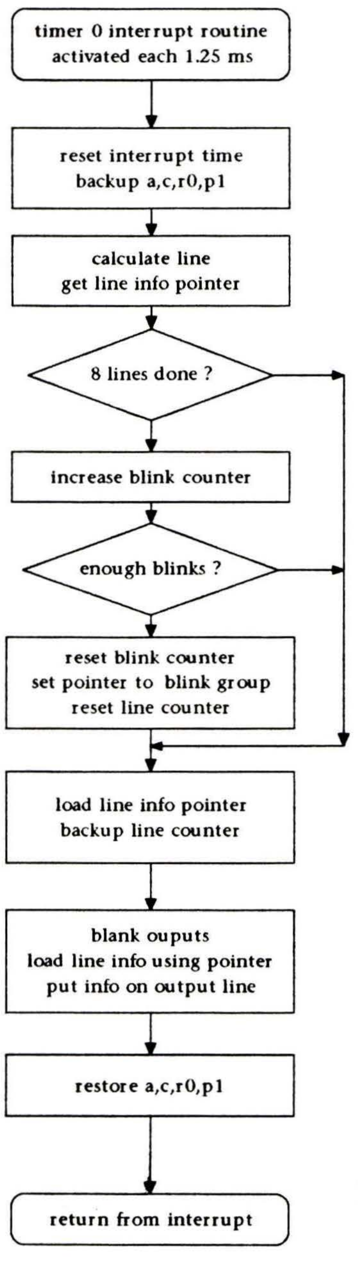

timer 0 interrupt routine activated each 1.25 ms
reset interrupt time backup a,c,r0,p1
calculate line get line info pointer
8 lines done ?
increase blink counter
enough blinks ?
reset blink counter set pointer to blink group reset line counter
load line info pointer backup line counter
blank ouputs load line info using pointer put info on output line
restore a,c,r0,p1
return from interrupt

Figure 5.6 Timer 1 interrupt

5.7 Interrupt serial port

The routine that is called after an interrupt (caused by the serial port), begins at the vector address of 8031 serial interrupt. If the send ready flag has been set by hardware, it resets the f0 bit to signal that the transmit channel is ready to send the next output byte.

The buffer of the serial input is set up as 16 bytes, with the first byte a pointer to the last received byte. A value of the first byte equal to the address of the first byte indicates that no bytes are buffered; a value equal to the start addres plus 16 indicates that the buffer is full.

If the receive flag has been set by hardware then the subroutine moves the received byte from sbuf to a buffer ('rbuffer'), it increases the pointer of the number of bytes buffered and calls the command handler to try to execute a command. After completing possible commands it returns to the program in which the interrupt occured.

Subroutine command handler

The command handler is called from the serial interrupt routine every time a byte has been received through the serial link. First the routine tries to find out if the first buffered byte is a command byte ($\geq$ 80H), if so, it loads the command byte from the buffer, else it tries to load the running status command.
A nonzero byte will indicate a command byte. Only 5 commands are supported: System_Reset, Output_Set, Output_All_Set, Input_Activate, Set_Blinking_Time. The label commands ([Eggen,1991]) are not implemented but could cause progam errors (see § 5.7.3). The command handler will compare the command byte with each of these five commands; if they match then the appropriate command is executed.

System_Reset command

The System_Reset command makes the interface box execute a software startup. The software startup (*init2*) follows the hardware startup; the software startup restarts the program (see paragraph 5.3).

Label_Exclusive command

The Label_Exclusive command is <u>NOT</u> implemented in the current program for the interface box. However, to make the interface box foolproof, provisions had to be made to make sure that the program couldn't execute any enclosed data of the label command. The routine deletes all data coming after the label exclusive command, until it finds an ELE (#0F0H / 240 D). After having found an ELE, it deletes the ELE and the Label_Exclusive command. It ends with sending a Command_Not_Implemented by calling *sendbyte*.

Output_Set command

With the Output_Set command an output can be set to a function. 4 Different functions are defined: off, blinking, inverse blinking and on continuously.
The routine sets a pointer to the blinking and inverse blinking output table, the running status to the Output_Set command and calls *bitop*, a part of the Input_Activate routine.

Output_All_Set command

The Output_All_Set command sets the output tables to the wanted bit function. The {LSB} of the databyte is placed in all the bits of the normal blinking group, the {LSB-1} of the databyte is placed in all the bits of the inverse blinking group. The running status is set to the Output_All_Set command.

Input_Activate command

With the Input_Activate command an input status can be set to a function. 4 Different functions are defined: input on/off not signalled, only input on, only input off, and both signalled. The routine sets the entry of an input in the input on/off table. The running status is set to the Input_Activate command.
The entry is number is contained in the first databyte. A pointer is set to the address of the associated entry in the input table (*bitop*). A possible range error is signalled to the host computer by sending a Data_Byte_Error command. The routine sets the entry in the input table to the function, contained in the second databyte, wanted.

Set_Blinking_Time command

The Set_Blinking_Time command is accompanied by two databytes. The first databyte contains the lower part of the blinking time, the second databyte contains the {MSB} of the blinking time. The routine places the {LSB} of the second databyte in the {MSB} of the first databyte. The result is placed in the memory location 'blink' to be used by the interrupt timer 1 routine. The counters are reset.

Movedown routine

The commands called in the command handler read the received bytes on at a time. They are deleted when a complete command or an errors occurs. The deleting of the received bytes is done in the *movedown* routine. It erases the oldest byte received (FIFO), moves down received bytes, if any, and sets the pointer to the address of the last received byte.

Label_Exclusive command

The Label_Exclusive command is NOT implemented in the current program for the interface box. However, to make the interface box foolproof, provisions had to be made to make sure that the program couldn't execute any enclosed data of the label command. The routine deletes all data coming after the label exclusive command, until it finds an ELE (#0F0H / 240 D). After having found an ELE, it deletes the ELE and the Label_Exclusive command. It ends with sending a Command_Not_Implemented by calling *sendbyte.*

Output_Set command

With the Output_Set command an output can be set to a function. 4 Different functions are defined: off, blinking, inverse blinking and on continuously.
The routine sets a pointer to the blinking and inverse blinking output table, the running status to the Output_Set command and calls *bitop*, a part of the Input_Activate routine.

Output_All_Set command

The Output_All_Set command sets the output tables to the wanted bit function. The {LSB} of the databyte is placed in all the bits of the normal blinking group, the {LSB-1} of the databyte is placed in all the bits of the inverse blinking group. The running status is set to the Output_All_Set command.

Input_Activate command

With the Input_Activate command an input status can be set to a function. 4 Different functions are defined: input on/off not signalled, only input on, only input off, and both signalled. The routine sets the entry of an input in the input on/off table. The running status is set to the Input_Activate command.
The entry is number is contained in the first databyte. A pointer is set to the address of the associated entry in the input table (*bitop*). A possible range error is signalled to the host computer by sending a Data_Byte_Error command. The routine sets the entry in the input table to the function, contained in the second databyte, wanted.

Set_Blinking_Time command

The Set_Blinking_Time command is accompanied by two databytes. The first databyte contains the lower part of the blinking time, the second databyte contains the {MSB} of the blinking time. The routine places the {LSB} of the second databyte in the {MSB} of the first databyte. The result is placed in the memory location 'blink' to be used by the interrupt timer 1 routine. The counters are reset.

Movedown routine

The commands called in the command handler read the received bytes on at a time. They are deleted when a complete command or an errors occurs. The deleting of the received bytes is done in the *movedown* routine. It erases the oldest byte received (FIFO), moves down received bytes, if any, and sets the pointer to the address of the last received byte.

6 Conclusions

During the research which was conducted from mid april till june 1991 a number of problems occurred, each with a specific remedy: the timing of the serial port of the 8031 wasn't working properly (caused by a wrong crystal setting of the in-circuit debugger), the Sun serial port setting was malfunctioning (System_Reset commands were continuously sent to the interface box !), the reset circuit wasn't working properly (high impedance capacitor used), output lines were duplicated (wrong dip switch setting of the debugger), registers were overwritten (wrong use of stack variables), etc., etc. To get to know a microprocessor also takes a lot of time, many difficult (?) questions were answered by two specialists.

However, the final product with rudimentary versions of sip 1.0 commands is in working order. As the first goal was to implement less than the realized commands, simple program expansion made it possible that a few extra commands were implemented. A total of 64 input and 64 outputs can be monitored and controlled by the interface box.

The interface box hardware can be updated easily, a μP circuitry upgrade can be accomplished by exchanging the upper print, an I/O update can be done by exchanging the lower print. Hardware connections are described in the appendix (App. 1,3). The data memory (8 Kb RAM) is reserved for future use.

Future versions of implementation of the sip protocol can be programmed and can replace the current program (of the 32Kb ROM a large part is still unused).

The serial signal definition of SIP 1.0 (19200 baud, 8 data, 1 start, odd parity, 1 stop bits), is a bit awkward; as no error protocol is defined (in case of a parity error), it also can not be implemented. I would advise to choose a different protocol : 19200 baud, 8 data, 1 start, no parity, no stop bit. In this way the interface box can also be connected to PC-like computers.

The interface in it's present can also be used for many things besides it's intentional goal: burglar alarms (remote controlled), proces control (measuring of data), etc. It's universal serial bus can be completed with a telephone modem to connect it to a phone line. In this way the interface box can be placed far away from the host computer.

Literature list

Mc Namara, J.E. (1977)
Technical aspects of data communication
Bedford, U.S.A.: Digital Press

Intel. (1985)
Microcontroller handbook
data book

Philips. (1985)
High speed CMOS IC06N
data book

Bierens, E.J.J (1989)
Eén-toets bediening voor de Pocketstem.
IPO Report no. 709.

Maxim. (1989)
Integrated circuits
data book

Zelissen, M.L.M. (1990)
De gemoderniseerde Pocketstem.
IPO Report no. 767.

Waterham, R.P. (1989)
The "Pocketstem": an easy-to-use speech communication aid for the vocally handicapped.
Thesis, Technical University Eindhoven.

Deliege, R.J.H. (1989)
The "Tiepstem": an experimental Dutch keyboard-to-speech system for the speech impaired.
Thesis, Technical University Eindhoven.

Eggen, J.H. (1991)
The serial interface protocol (SIP)
Manual IPO 109

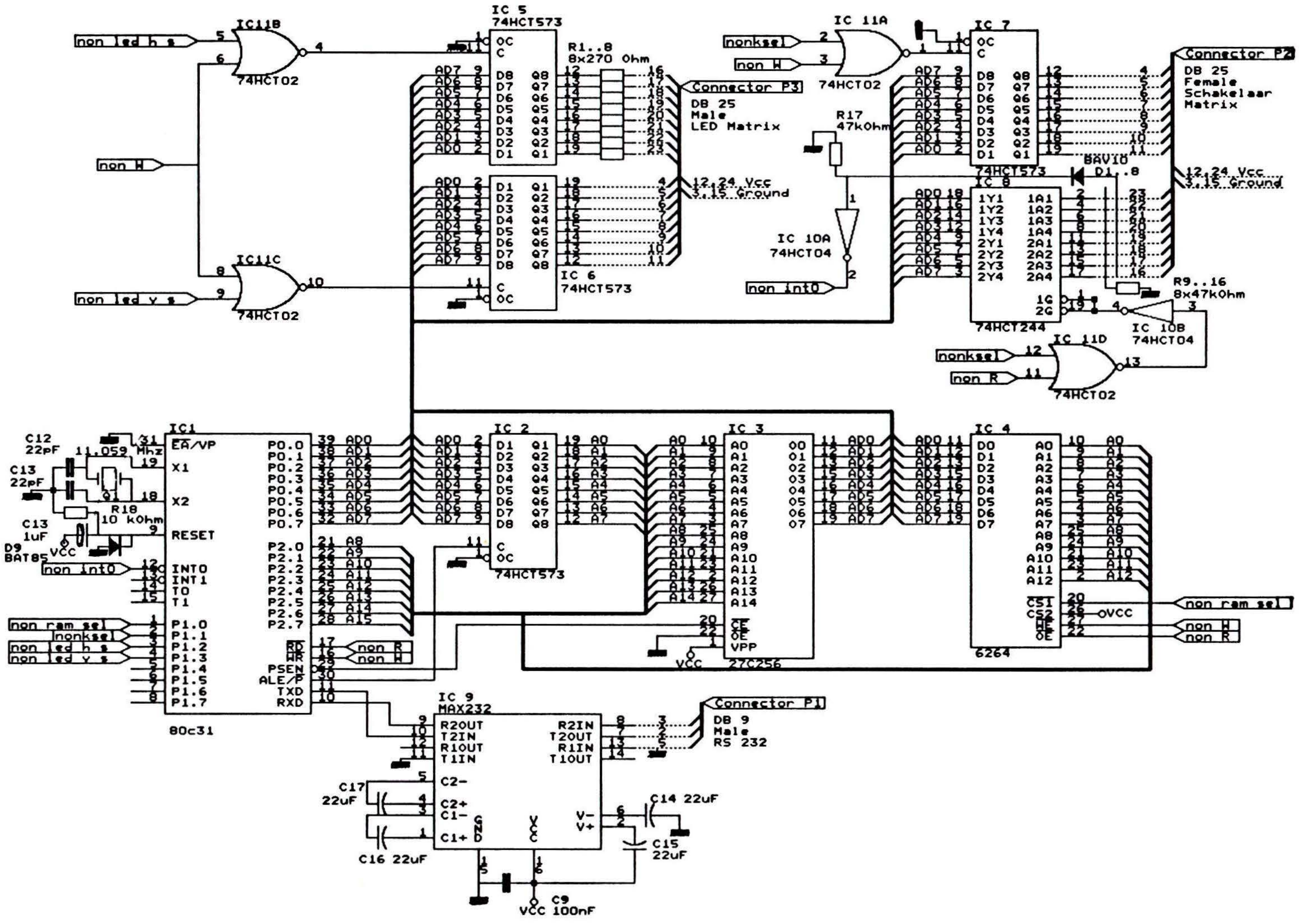
Kouwenberg, R.F.W.M. (1991)
The Sip Interface Box (SIB)
Manual IPO 112

Acknowledgements

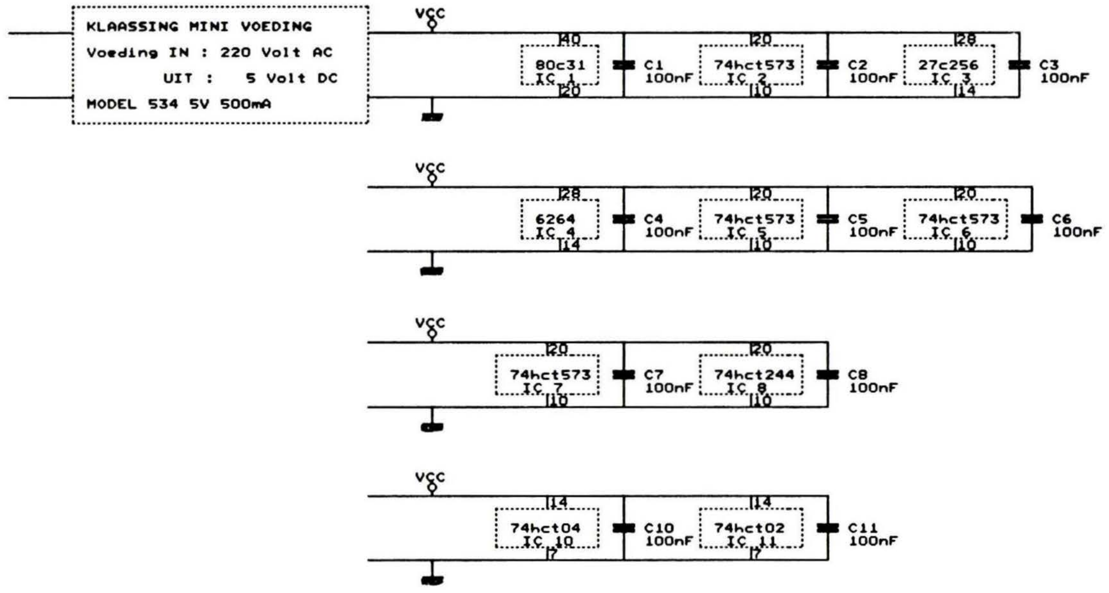
I hereby wish to speak my gratitude to the people who helped me during the research, i.c. in person Dr. Ir. R.P. Waterham, Dr. Ir. R.J.H. Deliege, Ing. J.G. Jonker, Ir. J.H. Eggen, the secretary, the people at the workshop and many others.

October 1991, Rob F.W.M. Kouwenberg

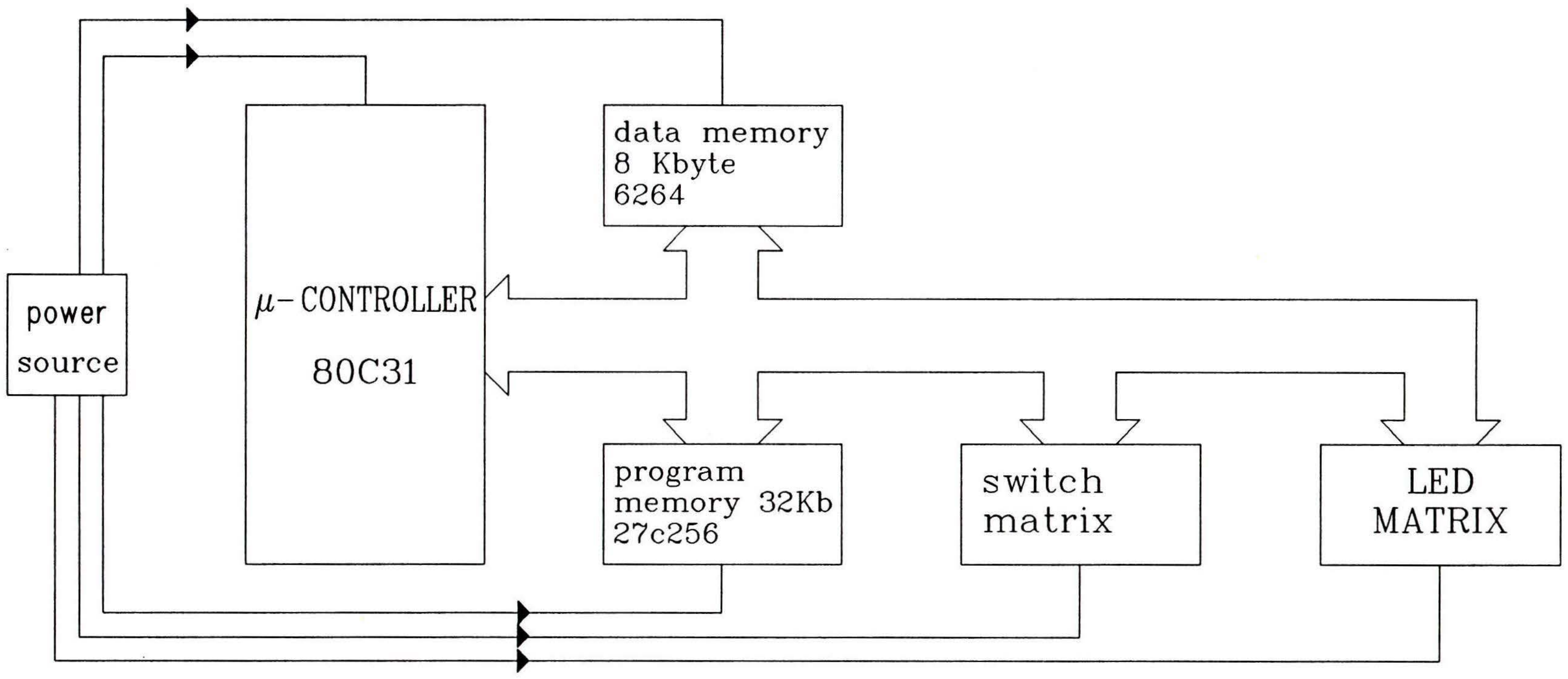
Appendix 1a Orcad circuitry drawing



Appendix 1b Power lines



Appendix 1c Schematic diagram



Appendix 2: Parts list

I.C's :	
IC 1	80C31
IC 2	74HCT573
IC 3	27C256
IC 4	6264
IC 5	74HCT573
IC 6	74HCT573
IC 7	74HCT573
IC 8	74HCT244
IC 9	MAX232
IC 10	74HCT04
IC 11	74HCT02

Weerstanden :	
R 1..R 8	270 Ohm
R 9..R16	47 kOhm
R17	10 kOhm

Condensatoren :	
C 1..C11	100 nF
C12,C13	22 pF
C14..C17	22 uF

Diodes :	
D1..D8	BAV10
D9	BAT85

Voeding :
KRP model 442

Ic voetjes:	Connectoren:	Diversen:
1 x 40 pens	2 x 20 pens dil	Schakelaar 220 volt
2 x 28 pens	1 x 10 pens dil	Kastje
5 x 20 pens	2 x 25 pens d-sub	Experimenteerprint
1 x 16 pens	1 x 9 pens d-sub	9 + 25 polig lint
2 x 14 pens	220 volt connector	draad