Giant Meter wave Radio Telescope
National Center for Radio Astrophysics
Tata Institute of Fundamental Research

NCRA • TIFR

[redacted]

Interface between a PC parallel port and GPIB devices

Author : L. Pommier
Project director : T. L. Venkatasubramani
Date : 06/02/04

Table of Contents

Illustration Index

Index of Tables

Index

Label	*Full Name*
ECP	Extended Capability Port
EPP	Enhanced Parallel Port
DAQ	Data Acquisition
DMA	Direct Memory Access
GMRT	Giant Meterwave Radio Telescope
GPIB	General Purpose Interface Bus
lpr	Local printer port
OS	Operating System
SPP	Standard Parallel Port

1.Introduction

The Giant Meter wave Radio Telescope (GMRT) is a radio telescope located in Khodad, at 80 km from Pune. It is composed of an array of 30 antennas spread over distances up to 25 km. Each antenna is 45 m in diameter, and has been designed to operate at a range of frequencies from 50 to 1500 MHz.

For all its experiments and internal divisions, the GMRT requires different data acquisition systems. This document deals with one of these systems: the parallel port-to-GPIB driver.

The General Purpose Interface Bus (GPIB) is an interface system through which interconnected electronic devices communicate. It is a digital, 8-bit parallel communication interface which usually allows a PC to be connected with programmable instruments, to control them and to acquire their data. Because of its robustness and its versatility to many devices, the GPIB was accepted as the industry standard IEEE 488.

The parallel port-to-GPIB driver connects a GPIB instrument to the PC parallel port. It has been developed by S. Joardar in GMRT to make the parallel port reproduce the GPIB protocol and some of its most important functions.

Since the 2 interfaces have different electrical specifications and since it can be useful to control many GPIB devices with this driver, the question of electrical compatibility between the 2 sides was raised.

This document first gives a overview of the GPIB and the Parallel Port interfaces. Then a second part presents the software and hardware features of the existing parallel port-to-GPIB driver, and analyzes the electrical compatibility between the 2 interfaces. A last part finally designs an adapter box that can be included just after the PC to improve the electrical transmission.

2.The GPIB interface

2.1.GPIB messages

The GPIB carries device-dependant messages and interface messages.

- Device-dependant messages, often called *data* or data messages, contain device-specific information such as programming instructions, measurements results, machine status, and data files.
- Interface messages manage the bus itself. They are usually called *commands* or commands messages. Interface messages perform such tasks as initializing the bus, addressing and unaddressing devices, and setting device modes for remote or local programming.

The term command as used here shouldn't be confused with some device instructions, which can also be called commands. Such device-specific instructions are actually data messages.

2.2.GPIB devices

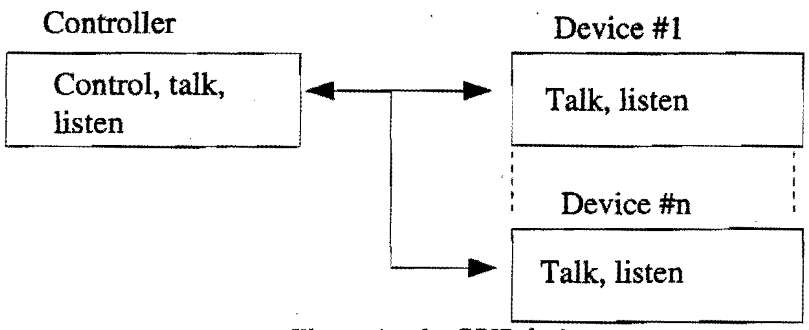


Illustration 1 : GPIB devices

GPIB devices can be talkers, listeners or controllers.

A talker sends *data* messages to one or more listeners. The controller manages the flow of information on the GPIB by sending *commands* to all devices (like a switching center of a city telephone system). A controller is necessary when the active or addressed talker or listener must be changed. The controller function is usually handled by a computer.

The bus supports one system controller, usually a computer (GPIB board), and up to 14 additional instruments. Devices are connected with a cable assembly consisting of a shielded 24-conductor cable with both a plug and receptacle connector at each end. With this design, you can link devices in a linear configuration, a star configuration, or a combination of the two. All devices and boards connected to the GPIB are assigned a unique GPIB address.

2.3.GPIB Signals

The interface system consists of 16 signal lines and 8 ground return or shield drain lines. The 16 signal lines are divided into the following three groups.

2.3.1.Data lines

The lines DIO1 through DIO8 are used to transfer addresses, control information and data. The formats for addresses and control bytes are defined by the IEEE 488 standard. Data formats are undefined and may be ASCII (with or without parity) or binary. DIO1 is the Least Significant Bit (note that this will correspond to bit 0 on most computers).

2.3.2.Management lines

The 5 interface management lines (ATN, EOI, IFC, REN, SRQ) manage the flow of control and data bytes across the interface.

- The ATN (Attention) signal is asserted by the Controller to indicate that it is placing an address or control byte on the data bus. ATN is released to allow the assigned Talker to place status or data on the data bus. The Controller regains control by reasserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.
- The EOI (End or Identify) signal has two uses. A Talker may assert EOI simultaneously with the last byte of data to indicate end-of-data. The Controller may assert EOI along with ATN to initiate a parallel poll. Although many devices do not use parallel poll, all devices should use EOI to end transfers (many currently available ones do not).
- The IFC (Interface Clear) signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After releasing IFC, the System Controller is the Active Controller.
- The REN (Remote Enable) signal is asserted only by the System Controller. Its assertion does not place devices into remote control mode; REN only enables a device to go into remote mode when addressed to listen. When in remote mode, a device should ignore its local front panel controls.
- The SRQ (Service Request) line is like an interrupt: it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a serial poll. The requesting device releases SRQ when it is polled.

2.3.3.Handshake lines

The 3 handshake lines (NRFD, NDAC, DAV) control the transfer of message bytes among the devices and form the method for acknowledging the transfer of data. This handshaking process guarantees that the bytes on the data lines are sent and received without any transmission errors and is one of the unique features of the IEEE-488 bus.

- The NRFD (Not Ready for Data) handshake line is asserted by a Listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD released (i.e. ready for data) until all devices have released it.
- The NDAC (Not Data Accepted) handshake line is asserted by a Listener to indicate it has not yet accepted the data or control byte on the data lines. Note that the Controller will not see NDAC released (i.e. data accepted) until all devices have released it.
- The DAV (Data Valid) handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the data lines and has had the minimum specified stabilizing time. The byte can now be safely accepted by the devices.

2.4.The handshaking protocol

The handshaking process is outlined as follows. When the Controller or a Talker wishes to transmit data on the bus, it sets the DAV line high (data not valid), and checks to see that the NRFD and NDAC lines are both low, and then it puts the data on the data lines.

When all the devices that can receive the data are ready, each releases its NRFD (not ready for data) line. When the last receiver releases NRFD, and it goes high, the Controller or Talker takes DAV low indicating that valid data is now on the bus.

In response each receiver takes NRFD low again to indicate it is busy and releases NDAC (not data accepted) when it has received the data. When the last receiver has accepted the data, NDAC will go high and the Controller or Talker can set DAV high again to transmit the next byte of data.

Note that if after setting the DAV line high, the Controller or Talker senses that both NRFD and NDAC are high, an error will occur. Also if any device fails to perform its part of the handshake and releases either NDAC or NRFD, data cannot be transmitted over the bus. Eventually a time-out error will be generated.

The speed of the data transfer is controlled by the response of the slowest device on the bus, for this reason it is difficult to estimate data transfer rates on the IEEE-488 bus as they are always device dependent.

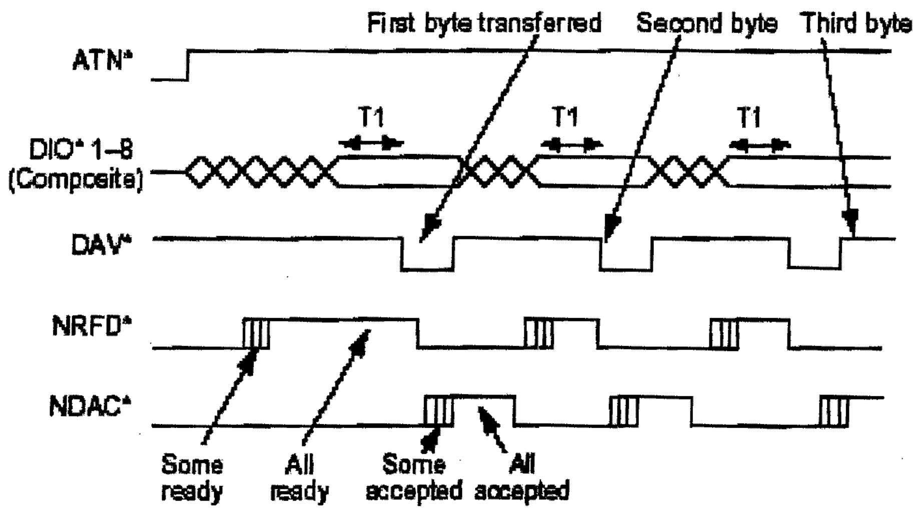


Illustration 2 : GPIB handshaking

3.The parallel port interface

3.1.The interface modes

As the design of the PC evolved, several manufacturers introduced improved versions of the parallel port. The new port types are compatible with the original design, but add new abilities, mainly for increased speed.

- Standard Parallel Port (SPP):

The SPP mode is the original IBM PC parallel port. It was based on an existing Centronics printer interface. SPPs can transfer 8 bits at once to a peripheral, using a protocol similar to that used by the original Centronics interface (Compatibility mode). The SPP doesn't have a byte-wide input port, but for peripheral-to-PC transfers, SPPs can use a Nibble mode that transfers 4 bits at a time using the status inputs.

- PS/2-type:

An early improvement to the parallel port was the bidirectional data port introduced on IBM's model PS/2. The bidirectional port enables a peripheral to transfer 8 bits at once to a PC. Byte mode is a 8-bit data transfer protocol that PS/2-type ports can use to transfer data from peripheral to PC.

- Enhanced Parallel Port (EPP):

An EPP can read or write a byte on the data port in one cycle of the IAS expansion bus (about 1 microsecond), compared to 4 cycles for previous types. It can also switch directions quickly. So the EPP mode allows high-speed transfer of bytes in either direction. Handshaking signals of this mode can distinguish between data and address transfer.

- Extended Capabilities Port (ECP):

ECPs propose the same performances than EPPs. ECPs also have support for DMA (Direct Memory Access) transfers. A control byte can contain an address or a data-compressed information. A FIFO buffer stores bytes to send and bytes received.

3.2.The parallel port signals

The parallel port signals are arranged into 3 registers of 8 bits. Those registers are in a special memory area dedicated to accessing input and output (I/O) devices.

The D-sub usual connector has 25 contacts, including 8 grounds and 17 transmitted signals. The next parts detail the registers bits, including the ones that don't appear at the connector.

3.2.1.The Data Register

The data register is located at the parallel port base address.

Bit	*Pin (D-sub connector)*	*Signal*	*Inverted at connector?*
0	2	Data bit 0	no
1	3	Data bit 1	no
2	4	Data bit 2	no
3	5	Data bit 3	no
4	6	Data bit 4	no
5	7	Data bit 5	no
6	8	Data bit 6	no
7	9	Data bit 7	no

Table 1 : Parallel port bits of the Data Register

The data port, or data register, holds the byte written to the data output. In bidirectional data ports, when the port is configured as input, the data register holds the byte read at the connector's data pins.

3.2.2.The Status Register

The status register is located at the parallel port base address + 1.

Bit	*Pin (D-sub connector)*	*Signal*	*Inverted at connector?*
0	-	May be timeout	-
1	-	unused	-
2	-	unused	-
3	15	nError	no
4	13	Select	no
5	12	PaperEnd	no
6	10	nAck	no
7	11	Busy	yes

Table 2 : Parallel port bits of the Status Register

The status port, or status register, holds the logic states of five inputs, S3 through S7. Bits S0-S2 don't appear at the connector. The status register is **read only**, except for S0, which is a timeout flag on ports that support EPP transfers, and can be cleared by software. On many ports, the status bits have pull-up resistors.

3.2.3.The Control Register

The data Register is located at the parallel port base address + 2.

Bit	Pin (D-sub connector)	Signal	Inverted at connector?
0	1	nStrobe	yes
1	14	nAutoLF	yes
2	16	nInit	no
3	17	nSelectIn	yes
4	-	Interrupt enable	-
5	-	Direction control for bidirectional data ports	-
6	-	unused	-
7	-	unused	-

Table 3 : Parallel port bits of the Control Register

The control port, or control register, holds the states of 4 bits, C0 to C3.

Conventionally, these bits are used as outputs. On most SPPs, however, the control bits are open-collector or open-drain type, that means they can also function as inputs (cf. Paragraph 4.4.1). However, on most ports that support EPP and ECP modes, to improve switching speed, the control output are push-pull type and can't be used as inputs. On some multi-mode ports, the control bits have push-pull outputs in the advanced modes, and for compatibility they switch to open-collector/open-drain outputs when emulating a SPP.

Bits C4 to C7 don't appear at the connector. In bidirectional ports, the C5 sets the direction of the data port. It is set to 0 for output (data outputs enabled), 1 for input (data outputs disabled). Usually you must first configure the port for bidirectional use (system's CMOS setup screen, accessed on boot-up), in order for this bit to have an effect. It is unused in SPPs.

3.3.Accessing the parallel port

A port on a system motherboard has configuration options (base address and mode) in the system setup screen (the CMOS setup) accessible on boot-up. Usually the parallel port base address is located at 3BCh, 378h, or 278h. Now different options allow to access the parallel port.

3.3.1.Programming ways

The usual way to access the parallel port is to use the DEBUG program under a DOS prompt:

- D 40:00 displays the BIOS data area containing the addresses of the serial and parallel ports
- A 100 allows to enter instructions in the code segment at offset 100H, then write:

To write 0x00 to the port:

- mov dx, 378H ; *enters the port address*
- mov al, 00H ; *enters the value to write*
- out al, dx ; *write the value at the port adress*
- jmp 100 ; *resets Instruction Pointer to 100H*

To read the port:

- mov dx, 378H ; *enters the port address*
- in al, dx ; *read the value at the port adress and stores it in al register*
- jmp 100 ; *resets Instruction Pointer to 100H*

- R command displays the registers and flags contents
- T executes instructions one by one
- U 100,107 displays the written instructions

This DEBUG program works under Windows OS up to Win95. For higher versions, it gives incoherent results.

Another way is to write a program in C, ASM, Visual Basics... with specific functions talking to PC ports, and to run this program under any OS.

3.3.2.Linux direct access and results

Finally with a Linux OS, one can directly access I/O ports under the administrator account using the *inb <port>* and *outb <port> <value>* functions in a terminal (*ioport*). Examples:

- inb 0x37a reads byte at the port addressed at 37aH
- outb 0x37a 0x00 writes 00H byte to the port addressed at 0x37a

On the WincInt4 PC of the GMRT, the parallel port base address is 378H (CMOS setup or *cat /proc/ioport*). Using the last method, experiments with a multimeter and a +5V DC source give the following results:

	Data port pins (0x378)	Status port pins (0x379)	Control port pins (0x37a)
ECP or EPP mode	Write only with C5=0 Read only with C5=1	Read only	Write only
SPP mode	Write only with C5=0 Read only with C5=1	Read only	Write Read

Table 4 : Read / Write possibilities from PC to its parallel port

Writing C5=1 automatically puts 0xFF on the data pins. This condition is required when the parallel port open-collector outputs are not switched off for bi-directional transmission (cf. Paragraph 5.2.1), and then this task has not to be done by any software command.

When the control pins are used as inputs, they come back to their last output value when the incoming signal ends. High and Low TTL values can be read whatever the last output was.

4.The existing parallel port-to-GPIB interface

This chapter first introduces the software part of the parallel port-to-GPIB interface and details the hybrid cable connections between the PC and the devices. These two points have been developed by Shubendu Joardar. The lowest level is then to analyze the electrical compatibility between the parallel port and the GPIB port.

4.1.The parallel port-to-GPIB driver

The parallel port driver allows to control one device thanks to an Assembly language code (gpib.asm) running under Linux, Windows 16 or Windows 32. It reproduces the GPIB protocol and thus transmit and receive data with any GPIB device. The code resumes the following basic functions :

- ✔ *initialize,*
- ✔ *GPIB_send,*
- ✔ *iowrite,*
- ✔ *GPIB_stat,*
- ✔ *GPIB_Get,*
- ✔ *ioread,*
- ✔ *GPIB_SerPoll,*
- ✔ *GPIB_PutAdd,*
- ✔ *GPIB_GetData.*

Those functions are further used in an application code (testprogram.c) in C language. The following lines explain how to execute the 2 programs:

- *nasm -f elf -o gpib.o gpib.asm* : compile assembly driver program
- *gcc -Wall -c -o testprogram.o testprogram.c* : compile C application program
- *gcc gpib.o testprogram.o* : link the two object files
- *./a.out* : under an account which has necessary permission to access any PC port under Linux, for the final execution.

4.2.Cable connection between the two interfaces

The DB-25 parallel port connector and the GPIB connector are different, so an hybrid cable had been made to interface both sides.

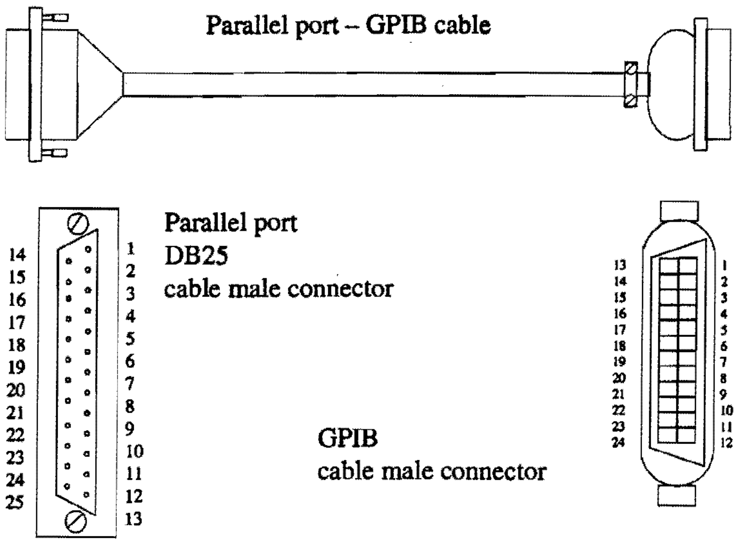


Illustration 3 : Parallel port – GPIB hybrid cable and connectors

Connection on Parallel port		Direction for GPIB functions	Connection on GPIB port	
Pin Name	Pin Nb		Pin Nb	Pin Name
Data strobe	1	◀—▶	6	DAta Valid
Data bit 1 to 4	2 to 5	◀—▶	1 to 4	Data I/O 1 to 4
Data bit 5 to 8	6 to 9	◀—▶	13 to 16	Data I/O 5 to 8
Acknowledge	10	——▶	9	InterFace Clear
Busy	11	Not connected		
Paper end	12	——▶	17	Remote ENable
Select	13	◀—▶	5	End Or Identify
Auto Line Feed	14	——▶	11	ATtentioN
Error	15	◀——	10	Service ReQuest
Initialize printer	16	◀—▶	7	Not Ready For Data
Select input	17	◀—▶	8	No Data ACcepted
Ground	18	——	12	Shield
Ground	19 to 25	——	18 to 24	Ground

Table 5 : Connections of the hybrid cable

☑ *Notes :*

➢ The parallel port has 25 pins, and the GPIB connector has 24 pins, so the parallel port pin number 11 is not connected in the hybrid cable.

➢ Ground lines are not twisted with signal lines like in GPIB cable. If some interferences between signals appear, the length of the cable must be reduced..

➢ The parallel port connector can have a maximum of 12 outputs (8 on data port and 4 on control port in SPP mode), so some line directions of the GPIB interface cannot be provided with this hybrid system:

- Bidirectional transmission is necessary for data (data port) and handshaking lines (control port).
- EOI, IFC and REN are not allowed in the PC-to-peripheral direction. So the IFC and REN functions are not provided by this driver, and the EOI is used only when reading from the peripheral device.

➢ REN, SRQ and IFC are unidirectional lines.

4.3.Electrical specifications

On parallel ports and GPIB controllers, drivers and receivers are used to make the electrical transition between the terminal card (low current for ICs) and the bus (high current for line transmission).

☑ Vocabulary note :

- *a driver is a component which can increase electrical outputs features (not to be confused with the software that implements functions of a port).*
- *a buffer is a component that will adapt its outputs to the following circuit.*
- *a receiver is a buffer for a port input (not to be confused with talker / receiver of the GPIB protocol).*
- *a transmitter is a driver for a port outputs.*
- *a transceiver is both a receiver and a transmitter combined.*

4.3.1.Parallel port electrical specifications

The IEEE 1284 standard specifies electrical characteristics for parallel-port drivers and receivers. It describes 2 types of devices: Level 1 devices are similar to the design of the original port, while Level 2 devices give better performances while remaining compatible with the original interface. Many newer port controllers meet the Level 2 specifications.

➤ The voltage specifications are the same for the 2 IEEE1284 levels:

	TTL High level	*TTL Low level*
transmitters	$V_{O\,min} = 2.4V$	$V_{O\,max} = 0.4V$
receivers	$V_{I\,min} = 2.0V$	$V_{I\,max} = 0.8V$

Table 6 : IEEE 1284 voltage specifications

➤ Current specifications:

☑ Note : *In usual data sheet specifications, a negative current indicates it goes out of the device pin. For a TTL high level, the current goes from the transmitter to the receiver (the transmitter sources current), and it is the inverse for a TTL low level (the transmitter sinks current). Maximum or minimum then only refer to absolute values.*

In IEEE1284 level 1 standard, transmitters outputs must be able to source a maximum of -0.32 mA, and to sink a maximum of 12 mA.

In IEEE1284 level 2 standard, transmitters outputs must be able to source / sink a maximum of -/+ 12 mA.

Receivers inputs can support currents up to +/-20 mA (input diode). In TTL high level, they sink a maximum of a few µA (usually +20 µA). In TTL low level, they sink a maximum of a few uA for data lines (usually -20 µA), and of -3 to -5 mA for control and status lines.

➤ Transmitters outputs types:

- ✔ Data port connections are 8 bidirectional pins, with outputs originally totem-pole type (74LS374).
- ✔ Status port pins are inputs only (LS TTL logic gates).
- ✔ Control port connections are 4 bidirectional pins, with outputs originally open-collector type (7404 open-collector inverters with 1.4 kΩ pull-up resistances).

Now components for parallel port of new PCs can vary according to the controller used. Parallel port controllers are nowadays integrated on PC motherboards and they are compliant to the level 2 of IEEE 1284 standard.

The IEEE1284 transceivers specifically used for parallel port intėrfacing are the 74ACT1284, 74VHC161284, 74LVX161284, and 74LVXZ161284 bidirectional buffers.

4.3.2.GPIB electrical specifications

The IEEE 488.1 standard specifies electrical characteristics for GPIB drivers and receivers.

➢ The voltage specifications are the same for the 2 IEEE1284 levels:

	TTL High level	*TTL Low level*
transmitters	$V_{O\,min} = 2.5V$	$V_{O\,max} = 0.5V$
receivers	$V_{I\,min} = 2.0V$	$V_{I\,max} = 0.8V$

Table 7 : IEEE 488 voltage specifications

➢ Current specifications:

GPIB transmitters are able to source a maximum of -5.2 mA (TTL high level), and to source a maximum of 48 mA (TTL low level).

GPIB receivers source a maximum of -1.6 mA, and sink a maximum of 50 µA.

The IEEE 488 load current for the GPIB bus at the receivers inputs is given by the following graphic:

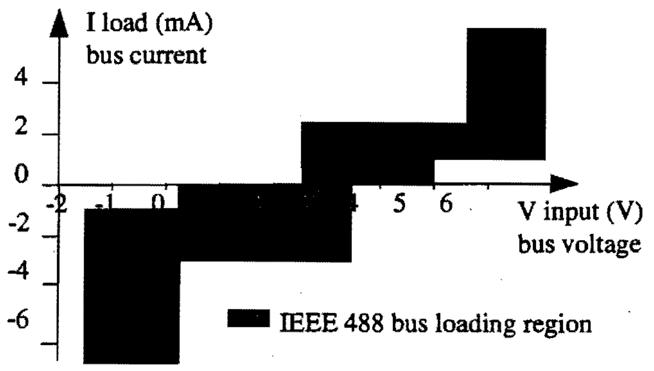


Illustration 4 : IEEE 488 bus current according to bus voltage

➢ Transmitters outputs types:

- ✔ Data port connections are 8 bidirectional pins, with 3-state outputs. The transceiver (SN74160) switches these outputs to open-collector ones for the parallel polling command.
- ✔ SRQ, NDAC and NRFD are bidirectional pins with open-collector outputs (SN74162).
- ✔ ATN, DAV, EOI, IFC and REN are bidirectional pins with 3-state outputs (SN74162).

4.4.Electrical compatibility between the two interfaces

To analyze interfaces compatibility, it is first useful to understand the functional of each output type.

4.4.1.Output types

4.4.1.1.Open-collector and open-drain

- The pull-up resistor is external (typically 4.7 kΩ with Vcc=+5V)
- The switching speed is slow (resistor to charge)
- The open-collector is a LS TTL technology. In CMOS components, the equivalent is open-drain output.
- The output can be connected to other open-collector outputs for bidirectional lines. With the following configuration, one must first write 1 to the output before reading the input.

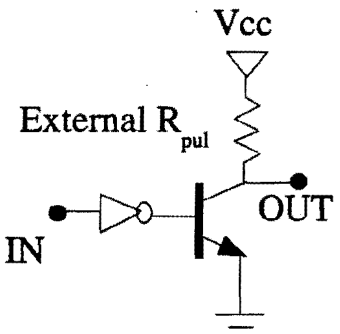


Fig. 5 : open-collector output

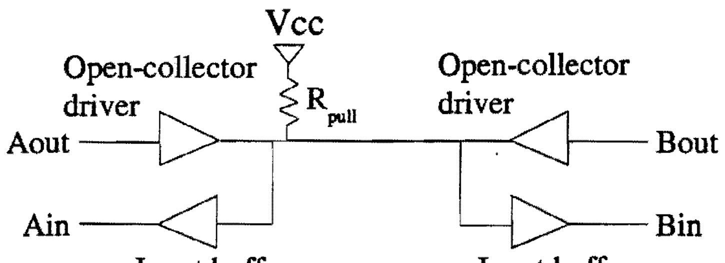


Fig. 6 : bidirectional line with open-collector outputs

Aout	Bout	Ain,Bin
0	0	0
0	1	0
1	0	0
1	1	1

When 1 is written to Aout, Ain follows Bout.

4.4.1.2.Totem-pole and push-pull

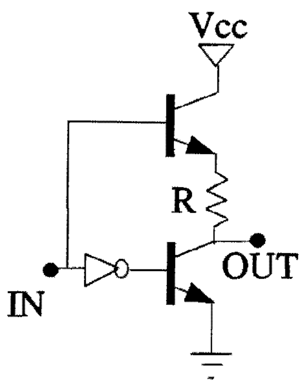


Fig. 7 : Totem-pole output

- The resistor is typically 60Ω with Vcc=+5V
- The switching speed is high
- This output cannot be directly connected to other outputs. If IN is High and some Low signal arrive on OUT, Vcc is connected on the other side to the Ground with just a low resistor between them. Then the result is unpredictable and large current may destroy involved components. This problem is solved with a serial resistor on the line (typically 330Ω), but the switching speed will become low.
- It is a LS TTL technology. In CMOS components, a similar output is achieved, but with equal current-sinking and current-sourcing abilities (push-pull type).

4.4.1.3. Tri-state

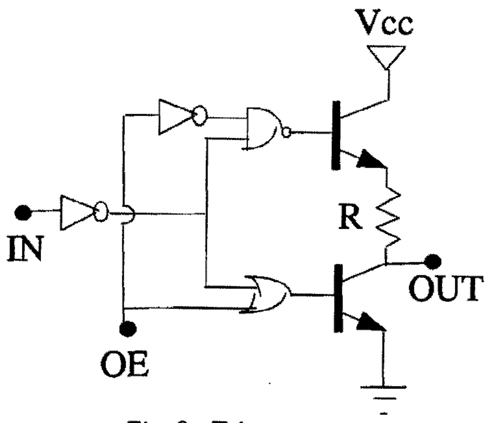


Fig. 8 : Tri-state output

- The resistor is typically 60Ω with Vcc=+5V.
- The switching speed is high.
- It requires an extra input (OE) to control the output enabling / disabling. When OE is High, both transistors are OFF and the output has no effect on external circuit (High Impedance state).
- Such control on output allows bidirectional lines (this driver must be disabled when the termination is used as an input), with fast transfer rates.
- It is the usual configuration on computer buses access. Every line termination is directed so that only one output is enable at a time.

4.4.1.4. Conclusion

Characteristics of outputs are very different according to their type.

Those figures are a simplification of transmitters circuits actually used. With a more complicated circuit, open-collector transmitters can also be enabled or disabled by a 3-state control.

The output impedance defines the switching speed. As an output switches, the voltage must charge or discharge through the cable's capacitance, and the lower the output impedance is, the faster the voltage can change.

A low output impedance prevent it from being directly connected to other outputs for a bidirectional configuration (cf. Totem-pull description). A serial impedance or a 3-state control must then be considered.

4.4.2.Conclusion on electrical compatibility between the two interfaces

Both interface sides have transceivers that protect internal circuits from the bus high currents. Still two specific points can become problematic for interfacing directly with the hybrid cable the 2 interfaces:

- the parallel port may not provide enough current to communicate with many GPIB devices. GPIB transmitters specifications allow currents up to -5.2 mA and +48 mA whereas parallel port transmitters allow currents up to +/-12 mA in level 2 standard, or -0.32 mA and 12 mA in level 1 standard. GPIB currents on the bus are showed in Illustration 4.

So level 1 parallel port drivers are not stronger enough to communicate with GPIB devices.

Level 2 parallel port drivers are less strong than GPIB ones, but GPIB systems are designed to support up to 16 devices. So we can conclude that actual parallel ports can communicate with ~4 devices (12mA/48mA *16 devices).

- When two outputs are directly connected and when they are not both open-collector type, a high current can happen (cf. Paragraph 4.3.3.1).

This can be the case for DAV and ATN lines, but on both sides, port controllers have one bit to control the direction of their bidirectional pins. The GPIB protocol does not make the controller and the device talk at the same time, so outputs on the same line are not enabled at the same time and this problem does not happen here.

5.Further interfacing solution: the adapter box

5.1.Introduction

As concluded in the latest part, actual parallel ports can communicate with many GPIB devices thanks to the assembly driver. Now if some user wants to use this driver with more devices, or with an IEEE1284 Level 1 parallel port controller, it is necessary to add new transmitters after the parallel port to make it output more current.

So some hardware must be added between the two sides. This will be an adapter box localed between the PC parallel port and the first device along the hybrid cable. On the PC side, the adapter box will transmit according to the parallel port specifications. On the devices side, the adapter box will transmit according to the GPIB specifications.

Only one device must be connected to the PC, and from this device whether a linear-chained or a star-chained configuration can be formed thanks to GPIB cables which connectors are both plug and receptacle:

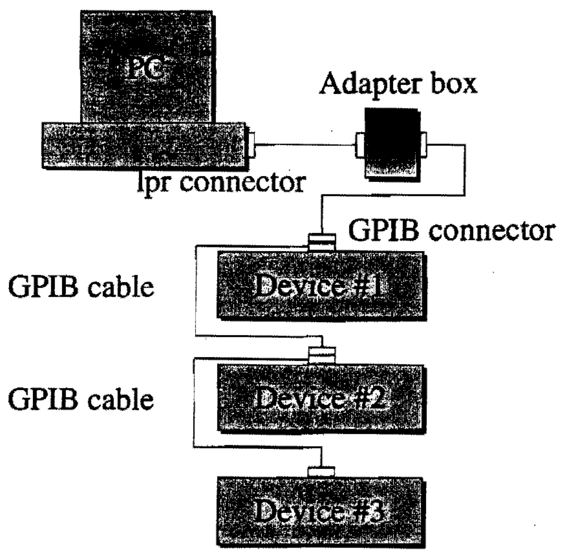


Illustration 9 : Linear-chained configuration

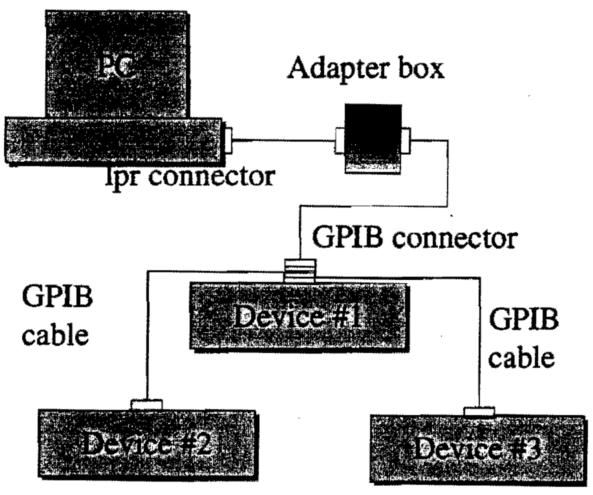


Illustration 10 : Star-chained configuration

5.2.Adapter box Design

5.2.1.Adapter box without direction control

5.2.1.1.Presentation

The first idea is to use transmitters without direction control, so that outputs are always enabled. On both PC and devices side, some totem-pole outputs on bidirectional lines will then face the adapter box outputs, so some serial resistance is necessary to protect the components against high current.

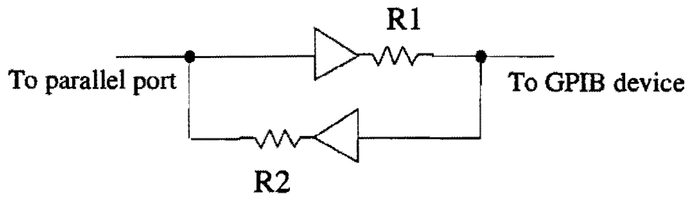


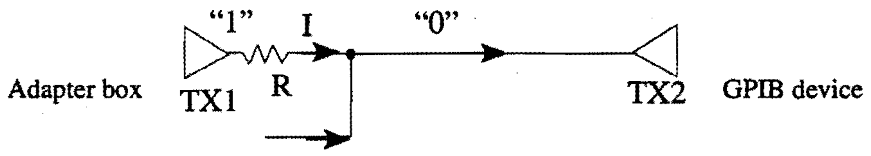
Illustration. 11 : Adapter box transceiver without direction control for bidirectional lines

5.2.1.2.Resistances not compatible with GPIB currents

The resistances are necessary to protect outputs connected on the line against high current. At the same time, resistances must not change the value transmitted by the output. The following equations resumes those conditions on the GPIB device side ("1" is TTL High level, and "0" is TTL low level).

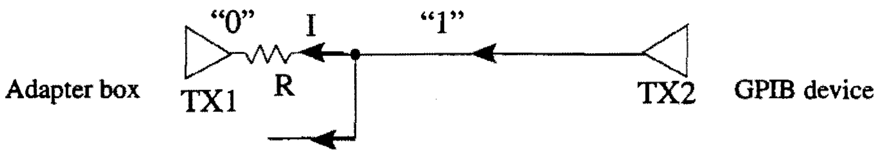
When the device talks, it outputs one value that will face the adapter box buffer which output value can be different for a short while, before the loop changes it.

- TX1 had set "1", TX2 sets "0"



$$V_{OH} - R.I = 0 \; with \; I < I_{OH.max} \; supported \; by TX1 \qquad (1) \quad R > \frac{V_{OH}}{I_{OH.max}}$$

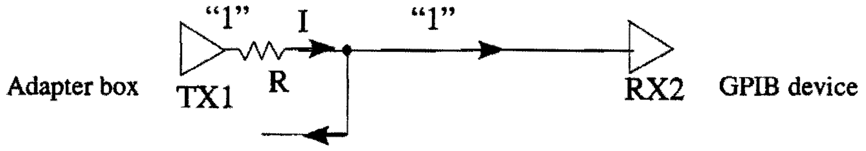
- TX1 had set "0", TX2 sets "1"



$$V_{OH} - R.I = 0 \; with \; I < I_{OL.max} \; supported \; by TX1 \quad (2) \quad R > \frac{V_{OH}}{I_{OL.max}}$$

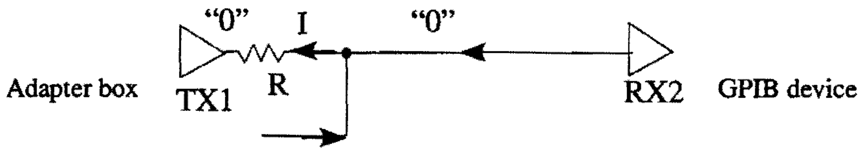
When the PC talks, the device transmitters are disabled by a 3-state control and receivers are enabled.

- TX1 sets "1", RX2 must receive "1"



$V_{OH.min} - R.I > V_{IH.min}$ (3) $R < \frac{(V_{OH.min} - V_{IH.min})}{I_H}$

- TX1 sets "0", RX2 must receive "0"



$V_{OL.max} + R.I_L < V_{IL.max}$ (4) $R < \frac{(V_{OL.max} - V_{IL.max})}{I_L}$

With numerical values of GPIB buffers:

Typical values:	V_{OH} min = 2.5 V	I_{OH} max = 5.2 mA
V_{OH} = 3.4 V	V_{IH} min = 2.0 V]	I_{OL} max = 48 mA
I_H = 2 mA	V_{OL} max = 0.5 V	
I_L = 2 mA	V_{IL} max = 0.8 V	

Results:

(1) R > 654 Ω	(2) R > 71 Ω	(3) R < 250 Ω	(4) R < 150 Ω

Conclusion:

With such values, resistance requirements are not compatible. It is possible to slightly vary the resulting values by assuming that typical values will be different, but finally the goal is to achieve higher currents to communicate with many GPIB devices, and then a serial resistance protection will change transmitted TTL values.

5.2.2.Adapter box with a 3-state control

5.2.2.1.Presentation

A good solution to design the adapter box is to control the direction of.each line inside the box. The next analysis shows that only one bit is required, and this bit can be controlled by adding a few lines inside the assembly driver.

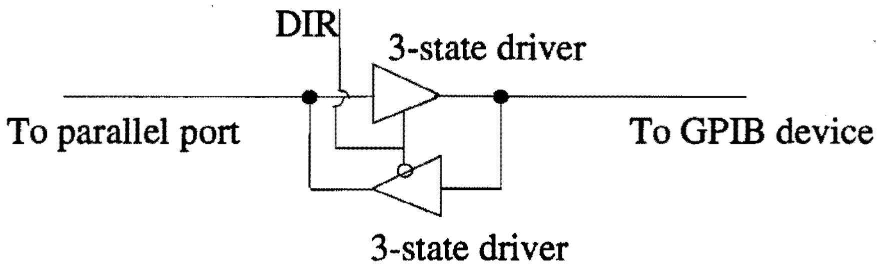


Illustration. 12 : Adapter box transceiver with a 3-state control for bidirectional lines

Signal	Direction: PC ◀ ▶ GPIB device	
	Talker = PC	Talker = a device
DAV	⟶	⟵
Data I/O (8)	⟶	⟵
IFC	⟶	⟶
REN	⟶	⟶
EOI	⟶	⟵
ATN	⟶	⟶
SRQ	⟵	⟵
NRFD	⟵	⟶
NDAC	⟵	⟶

Table 8 : GPIB lines direction

According to the GPIB protocol, it is possible to know every line direction when a function is called:

1. A command message can happen if the bus needs new managing tasks (new talker for next data message...). For this command, the talker is always the controller (PC).
2. A data message is then transfered with the last talker defined (PC or a device).

So according to who the talker is, DIO (8), EOI, and DAV are in one direction, and NRFD and NDAC are in the other direction.

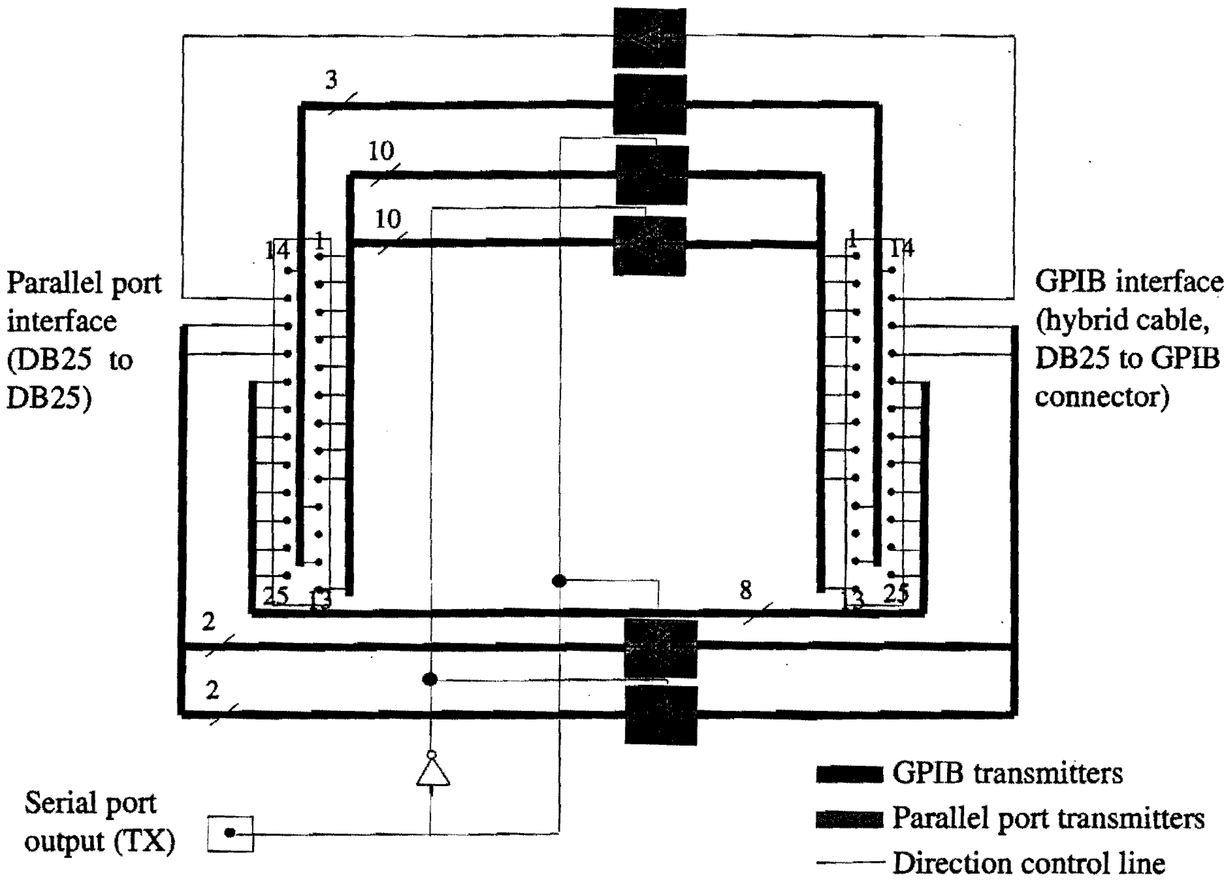
ATN, IFC, REN, and SRQ are always unidirectional, assuming that the controller is always the PC.

To control a bit that can set the 3-state components direction, two ways can be considered:

- The direction signal could be issued from a combination of other lines. The ATN line takes a value to indicate if the message is a command or a data, but it is not sufficient to define who is the talker. Other lines values also don't indicate the direction, so this method is not possible.
- The assembly code defines the talker in every step. So it can also assert a spare line to be the direction signal for the adapter box components. Unfortunately the parallel port has no output left. The only solution is to use another port output (serial port).

5.2.2.2. Final adapter box design

According to the previous analysis, the adapter box design is:



3
10
10
14
1
1
14
Parallel port interface (DB25 to DB25)
GPIB interface (hybrid cable, DB25 to GPIB connector)
25
13
8
13
25
2
2
Serial port output (TX)
GPIB transmitters
Parallel port transmitters
Direction control line

Illustration 13 : Adapter box design

Components choice :

- The best choice is to use transmitters available in GPIB and parallel port transceivers (SN74160, SN74162, 74ACT1284). Nevertheless, those specific transceivers are not usual and may be difficult to get.
- Another possible choice is to use the classical 74LS244 octal buffer / line driver with 3-state outputs. Its performances will be less ideal, but its transmitters are able to output currents up to -15 mA, and +24 mA. So it should be able to communicate with 8 GPIB devices.

Complementary notes:

- The power supply for ICs can be taken from a free PC internal power line, or from an adapter like printers ones, with a power regulator.
- Grounds from power supply, serial port, and ICs do not appear on the last design, but of course they have also to be connected altogether.

6.Multi-devices testing and further proceedings

6.1.Multi-devices testing

As concluded in the paragraph 4.4.2, the parallel port and the GPIB interfaces are int theory electrically compatible for few devices. This has been tested in the following conditions:

- PC:

winclnt4 PC (Pentium 4) under Linux Red Hat 9.

- Spectrum analysers (SPA #1 & 2):

HP 8590L with GPIB address set at 17 and 18 respectively.

- Signal Generator (SGN):

HP 8714C.

- Application program:

C file listed in appendix 1 "testprogram.c".

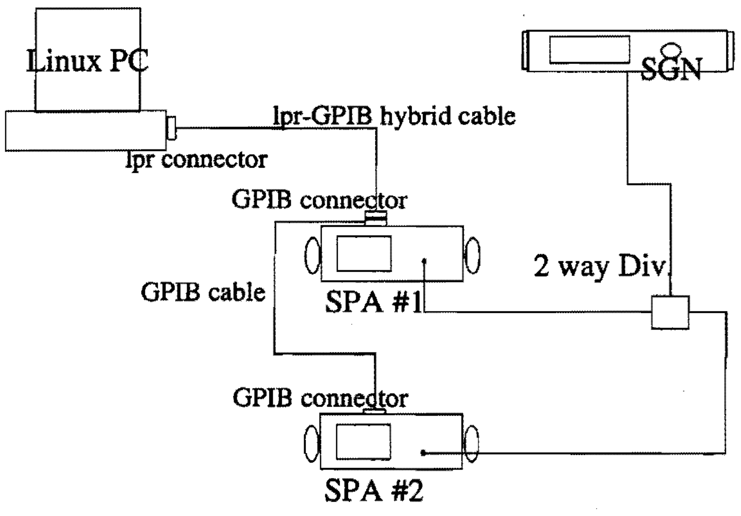


Illustration 14 : Test setting

The parallel port driver is successfully working. Data displayed in the terminal shell, according to the application program, correspond at each time to the RF signal send by the signal generator and measured by the two spectrum analysers.

6.2.Further proceedings

This work was then attempted to be extended to different type of devices.
First configuration tested only one voltmeter (HP 8508A) measuring a signal voltage on a single channel directly connected to a signal generator. During this test, writing functions (configuration, RST...) were working but reading operations failed.
Such problem can come from a software limitation, or some GPIB additional configuration for a voltmeter. No solution was straightly found and this work thus needs additional concentrated efforts.

7.Conclusion

The parallel port-to-GPIB driver is a program that allows a PC to be connected with GPIB programmable instruments, to control them and to acquire their data according to the GPIB protocol.

Nowadays, PC parallel ports are compliant to the level 2 electrical specifications of the IEEE1284 standard, that ensures that they are powerful enough to safely communicate with around 4 GPIB devices. The driver and a C application program have successfully been tested under Linux with the Winclnt4 PC (Pentium) and 2 spectrum analysers.

For some weak parallel ports or for more devices, this report also gives a design of an adapter box, placed between the PC and the GPIB devices, which gives a better electrical compatibility between the two interfaces. The only possible design for such a box requires to use one more PC output, coming from the serial port for instance.

8.References

Books

1. Parallel Port Complete, *Jan Axelson*, Penram
2. NI-488.2 User Manual for DOS, *N.I. January 1996 Edition*, Part Number 320700-01.
3. PET and the IEEE 488 bus (GPIB), *Eugene Fisher – C. W. Jenson,* Website
4. IBM PC Assembly Language and Programming, 5th edition, *Peter Abel*, Pearson Education

Data sheets

5. SN74160B / SN74162B Octal GPIB transceiver, *Texas Instrument*, May 1995.
6. DS75160A / DS75161A IEEE-488 GPIB Transceivers, *National Semiconductor*, May 1999.
7. 74ACT1284 IEEE1284 Transceiver, *Fairchild Semiconductor*, November 2000.
8. 74LVXZ161284 (B) IEEE 1284 Transceiver, *Fairchild Semiconductor*, August 2003.
9. SN74LS244 buffer / line driver with 3-state control, *Motorola.*

9.Appendix

Testprogram.c

```
#include<stdio.h>
#include<asm/io.h>
#include<time.h>



extern int initialise(int,int);
extern int iowrite(int,char*,int);
extern int ioread(int,char*);





long int  mytime(void)
{  time_t t;
   t=time(NULL);
   return t;
};

main()
{ char buf[3200];
  int i,n;
  if(iopl(3))
      {  printf("\npermission denied\n");
         exit(1);
      }
   n=initialise(0x378,0);
  if(n==0)   printf("\ndevice initialised\n");
   else
     {  if(n==-1)    printf("\ndevice not initialised\n");
           else   printf("\nsome handshaking problem or device not attached\n");
      }
iowrite(18,"CF 130MZ;",9);
iowrite(18,"SP 50MZ;",8);
for(i=0;i<3200;i++)   buf[i]='#';
iowrite(18,"SNGLS;",6);
iowrite(18,"TS;",3);
iowrite(18,"TRA?;",5);
n=ioread(18,buf);
iowrite(18,"CONTS;",6);
if(n<0) printf("reading ERROR\n");
if (n>0)  buf[n]='\0';
printf("\nThe dumped screen data 1 is as follows:\n\n");
printf("\n %s  \n %d\n",buf,n);

iowrite(17,"CF 130MZ;",9);
iowrite(17,"SP 50MZ;",8);
for(i=0;i<3200;i++)   buf[i]='#';
iowrite(17,"SNGLS;",6);
iowrite(17,"TS;",3);
iowrite(17,"TRA?;",5);
n=ioread(17,buf);
iowrite(17,"CONTS;",6);
if (n>0)  buf[n]='\0';
printf("\nThe dumped screen data 2 is as follows:\n\n");
printf("\n %s  \n %d\n",buf,n);

return 0;
}
```