DESIGN AND REALISATION OF A SYSTEM FOR INFORMATION EXCHANGE BETWEEN A NUMBER OF COMPUTERS / MICROCONTROLLERS

SUPERVISOR: MARK SARLET
DATE OF PRESENTATION: 3 OF JUNE 1996
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1: The Physical Connection</th>
<th>page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 2: The Logical Connection</td>
<td>page 7</td>
</tr>
<tr>
<td>CHAPTER 1: The software</td>
<td>page 15</td>
</tr>
</tbody>
</table>
This project involves the realisation of a system for exchange informations between a number of computers. A system based on a master/slave principle. This system is designed for the later use of radio transceiver. So the master(s) will be able to send command for controlling the slaves-systems ( e.g. remote controlled vehicles ).

For connectivity in a room or building, we can use cables to convey digital signals directly between a Data Terminal Equipment ( DTE ) and another ( DTE ). That is the problem in direct communication between two personal computers. That we go against the grain of traditional communications. A DTE communicates properly with devices designed to act like Data Circuit - Terminating Equipment ( DCE ) like a modem or a printer and not like another terminal.

The connection between a Personal Computer and any other computers is

* **Physical connections.** That allow the computers to exchange compatible electrical signals.

* **Logical connections.** The appropriate combination of signaling schemes and protocols, that cause the electrical signals generated by one computer to result in the desired response at a second computer.

* And finally we must use **software** that supports the physical and logical features that we used.
The Physical Connection

The hardware that we have to use is: 2 PC's with serial, asynchronous communications (RS-232C or RC-232D) ports, separated by a device called a null modem, with the use of 2 RS-232 serial communications cables.

To communicate with the other PC, electric circuitry must convert the parallel bit stream used within the IBM PC system unit into a serial bit stream by using the asynchronous communications adapter. The port receives data eight bits at a time through the PC data bus. The port hardware then performs a parallel-to-serial conversion for all eight bits. The port sends the data bits sequentially, one bit at a time, through the serial communications channel. At the receiving end, a serial port converts the serial flow of data bits back into a parallel flow of bits (serial-to-parallel...
The receiving computer then processes the information through its internal parallel bus.

To get from the serial communications port to another device requires the use of specific connectors and cables.

**PC's (the AT)** have 9pin male SUB-D connectors:

![Diagram of IBM PC AT Serial Port and Asynchronous Modem](image)

- 1: Carrier Detect
- 2: Received Data
- 3: Transmitted Data
- 4: Data Terminal Ready
- 5: Signal Ground
- 6: Data Set Ready
- 7: Request To Send
- 8: Clear To Send
- 9: Ring indicator

and 25pin male SUB-D connectors:

![Diagram of 25pin SUB-D connectors](image)

- Sec. Transmitted Data
- Sec. Received Data
- Received Clock
- Local Loopback
- Sec. Request To Send
- Data Terminal Ready
- Data Set Ready
- Data Signal Rate Set
- Transmitter Clock
- Sec. Clear To Send

The most important lines are RxD, TxD, and GND. Others are used with modems, printers and plotters to indicate internal states.
Transmitted Data (TxD, pin 3/2): The data to be transmitted from the PC to a device (the other PC). The serial port maintains this circuit in the marking condition (logic condition 1, equivalent to a stop bit) when no data are available.

Received Data (RxD, pin 2/3): The data to be transmitted from a device (the other PC) to the PC's serial port. This circuit remains in a marking condition when no data are available.

Request To Send (RTS, pin 7/4): The PC asserts this line whenever it wants to send data. A modem uses this signal as a starting signal for building up its carrier.

Clear To Send (CTS, pin 8/5): Is asserted by the modem after it has built up the carrier. The PC can now start sending the data.

Remark: in PC-PC communications where the modems are absent these two signals are used for handshaking. RTS now means: “I’m ready for receiving data”.

Data Set Ready (DSR, pin 6/6): Is asserted when the COM port is activated.

Data Terminal Ready (DTR, pin 4/20): Is asserted when the modem is operational and senses a 1 on the DTR line.

(These six lines plus GND are often referred to as ‘7 wire’-connection or ‘hand shake’-connection.)

Data Carrier Detect (DCD, pin 1/8): Indicates that the modem is detecting the carrier from the other side.

Signal Ground (GND, pin 5/7): The ‘signal ground’, i.e. the reference level for all signals.

Protective ground: This line is connected to the power ground of the serial adapter. It should not be used as a signal ground, and it must not be connected to GND.
Transmitted Data (TxD, pin 3 / 2) The data to be transmitted from the PC to a device (the other PC). The serial port maintains this circuit in the marking condition (logic condition 1, equivalent to a stop bit) when no data are available.

Received Data (RxD, pin 2 / 3) The data to be transmitted from a device (the other PC) to the PC’s serial port. This circuit remains in a marking condition when no data are available.

Request To Send (RTS, pin 7 / 4) The PC asserts this line whenever it wants to send data. A modem uses this signal as a starting signal for building up its carrier.

Clear To Send (CTS, pin 8 / 5) Is asserted by the modem after it has built up the carrier. The PC can now start sending the data.

Remark: in PC-PC communications where the modems are absent these two signals are used for handshaking. RTS now means: “I’m ready for receiving data”

Data Set Ready (DSR, pin 6 / 6) Is asserted when the COM port is activated.

Data Terminal Ready (DTR, pin 4/20) Is asserted when the modem is operational and senses a 1 on the DTR line.

(These six lines plus GND are often referred to as ‘7 wire’-connection or ‘hand shake’-connection.)

Data Carrier Detect (DCD, pin 1 / 8) Indicates that the modem is detecting the carrier from the other side.

Signal Ground (GND, pin 5 / 7) The ‘signal ground’, i.e., the reference level for all signals.

Protective ground This line is connected to the power ground of the serial adapter. It should not be used as a signal ground, and it must not be connected to GND.
Three lines (RxD, TxD & ground) are at least needed to make up a bi-directional connection.

A connection that is called null modem. The null modem cables makes each computer to think that is connected with a modem.

There are several ways to connect the cable pins together to get two PC's to talk to each other. The connection that we used is illustrated here:

When moving a data signal through a communications channel, it is necessary to vary electrical energy in the channel so that the information moves from one point in the media to another. The process of varying the electrical energy is modulation. A High-amplitude, high-frequency and fixed-frequency level of energy that flows through the channel is the signal carrier.

When a communications link must be established between a PC and another computer more than 50 feet away, the most economical (in cables) method of doing so is by using the public telephone system. To do this, two devices called modem must be installed between the telephone system and at each PC system board. The modem used the data to modulate the carrier, mostly based on face modulation. By modulating a carrier signal to reflex the original signal, a modulator device can generate a combined signal that is strong enough to make it to its destination and retain its information imprint. For this process to work, the second modem to the other side of the line has to demodulate, to separate the signal that arrives at its destination from the carrier that helped get there. The modem (MOdulate-DEModulate) can perform each of the two functions: modulation and demodulation.
The Logical Connection.

Two basic channel types are used in the data communications. The analog channel that its signals transmitted by the public telephone system or a commercial radio station. And the digital type that its signals transmitted by a PC to an external device through a cable or internal to the PC through its bus.

A communications channel moves electromagnetic energy between a source and one or more destination points while retaining the information contained in the energy when it leaves the source.

Bits or Binary Digits represents one of the two energy states (Voltage levels) that use the binary system to convey information or data. One state represents a binary zero and the other a binary one.

We use two different line states to designate the two logical binary values the two line voltages represent in a binary channel.

If no data is transmitted, the line is in its quiescent 'mark' state

\[
1 = \{-5 \text{ V} \ldots -15 \text{ V} \}
\]
\[
0 = \{+5 \text{ V} \ldots +15 \text{ V} \}
\]

But in practice we have:

\[
1 \approx -10 \text{ V}
\]
\[
0 \approx +10 \text{ V}
\]

That negative voltage, as is called, is indicates the data line voltage. In the other line we have positive voltage e.g.:

- when DTR = 1 \[\rightarrow\] DTR = +5 \ldots +15 \text{ V} \quad \text{Positive voltage.}
- But when TxD = 0 \[\rightarrow\] TxD = +5 \ldots +15 \text{ V} \quad \text{Negative voltage.}
Data looks like

<table>
<thead>
<tr>
<th>mark</th>
<th>0</th>
<th>10 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>0</td>
<td>-10 V</td>
</tr>
</tbody>
</table>

(1) start bit  (2) data bits  (3) stop bit(s)

But in most books these is kind of diagrams are drew the other way round, althow it seems to us that a zero should be the quiescent state, and the one an active state. But because the first teletypes used a current loop to continuously monitor the state of the line, and thus current flow was regarded as a 1 and it is “Mark”-ing time, and a signal then left a “Space” in the graph of current flow designating a zero.

Data looks like

<table>
<thead>
<tr>
<th>space</th>
<th>0</th>
<th>10 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>mark</td>
<td>1</td>
<td>-10 V</td>
</tr>
</tbody>
</table>

(1) start bit  (2) data bits  (3) stop bit(s)

Thus the bits following the start bit at level zero were true to their bit values. So if there is an interruption on the line it will cause the pause of Marking and in this way it’s easier to search for the problem.

The physical capabilities of the communications media limit both analog and digital communications. Physical connections media alters electromagnetic signals by resisting their flow and by adding noise to them.

Although a single electrical signal propagates in a given direction in a communication channel, it is possible to have two signals flowing in different directions in a channel.

The half-duplex communications channel - bi-directional data bus - that we use allows data to flow in both directions.

The hardware and software at both ends of the channel must execute a type of ‘handshake’ called line turnaround, to stop data flow in one direction start it flowing.

Design and realisation of a system for information exchange between a number of computers/microcontrollers.
in the other. So the receiver will become the transmitter, and the transmitter will become the receiver.

A digital channel has a capacity to transmit a specific number of information bit in a specified period called *bit rate*. The measurement is in *bits per second (bps)*.

The use of RS-232 C limit the size of the wires to 20 meters and minimise the transmission rate to 19200 bits per second.

*Band* is the number of *signals events* - changes in the frequency - transmitted per second. So one baud equals one such signal event per second, but the event may communicate more than one bit to the receiver.

For communications below 600 bps, the baud rate and bps transmitted between devices will be the same. In other words, a 300 baud modem is the same as a 300 bps modem. The rates differ at 600 bps and above because the analog transmission techniques that vendors have supply are such that each signal event carries two or more binary value.

The bps rating of a digital channel is the actual maximum rate it can convey digital bits. The actual data throughput of the channel, called the channel data rate, is usually less than the raw bit rate because the channel must provide overhead signals to synchronise communications between devices.

Digital communications depend upon exact timing of signal generation and reception to be successful. If the transmitter sends a signal and the receiver starts to examine the signal at the wrong time, the receiver will get meaningless information. The receiver must look at the signal at the appropriate times to detect the proper transition from one energy level to another, a process called *synchronisation*.

Synchronisation between a sending and receiving device requires an agreement on *bit time* between the two devices. To determine bit time, the receiver must *regenerate the clock signal* used by the sender to modulate the original signal. The receiver must extract the clock signal from the total signal it receives. After the receiver determines the bit time of the signal, it can then regenerate the digital signal as long as the bit time of its clock remains synchronised with the bit time the sender used to produce the signal.

Devices at each end of a digital channel can synchronise using one of two techniques. First, the transmitter and the receiver can work independently of each other and exchange a specified signal pattern at the start of each signal exchange, a process called *asynchronous* communications. Second, the transmitter and receiver can exchange initial synchronising information, then continuously exchange a digital signal stream that keeps them in lock step, a process called *synchronous* communications.

We are using asynchronous communication technique.

For asynchronous transmission, the bit stream itself must contain its own synchronising data for each byte of data. There are no separate clock signals transmitted between an asynchronous port and the DCE at the other end of the cable.
Asynchronous communication requires clock regeneration during the first bit period of each transmission frame. The technology that makes this clock regeneration possible is precise and high-frequency signal sampling. A receiver must monitor the communications channel and measure the voltage in that channel at a frequency that allows it to figure out the width of a bit period at the start of a transmission frame of data.

The sampling rate of an asynchronous receiver had to be at least 16 times the bit rate of the digital signal coming through the channel. By reading the signal amplitude in the channel at a rate 16 times the bit rate, the receiver can accurately locate the centre of the first signal transition that occurs in a channel.

The asynchronous, serial port hardware gives each group of bits - 5, 6, 7, or 8 bits from the data stream - it transmits a frame to delimit one transmitted group from another and to provide synchronisation between the transmitter and the receiver.

Each asynchronous byte begins with a start bit that tells the receiving device to begin measuring the subsequent data for the presence of 1s and 0s. A high-voltage signal on the data line always precedes the start bit. This marking line or marking state provides a clear contrast for the start bit and allows the receiving device to detect the beginning of a new start bit.

The asynchronous receiver must identify and measure the width of the digital signal that marks the start of an asynchronous frame. This start bit must have a value of zero. Its width in milliseconds is the bit time of the digital signal. The receiver must set its signal clock to match this bit time for the duration of the asynchronous transmission frame.

After successfully identifying and measuring the interval of an asynchronous start bit, an asynchronous receiver measure the voltage in the channel after each subsequent bit time. For eight-bit serial communications, the receiver does this sampling eight times and translates the signal strength it reads into a series of eight bit stores in a holding register. The receiver then does one final sampling of the signal strength in the channel after the ninth bit time to verify the proper end of a transmission frame. This sampling must result in a bit value of one called the marking state or stop bit that indicates synchronisation is still correct at the end of the frame.

Asynchronous communications require that each transmitted frame end with at least one stop bit. This last bit period must contain a value of one. This mark original provided the receiver time to process received data, and its duration had to be long enough for slow mechanical devices to finish their work. This bit period returns the signal in the channel to the marking state, which allows the receiver to find the centre of the next start bit.

The presence of a stop bit at the end of the asynchronous transmission frame allows the receiver to verify proper synchronisation with the digital signal stream. If the ninth bit sample does not produce a value of one, the receiver assumes there is a synchronisation problem and discards the eight bits it collected earlier as data. If the ninth bit is a mark, the receiver assumes synchronisation is correct and processes the eight data bits. While processing the eight data bits, however, the receiver must continue to monitor the channel for the arrival of the next asynchronous frame. The
next frame may arrive any time after the preceding one. A new start bit may begin at the end of the stop bit from the previous frame.

The transmitter determines how close to its maximum capacity an asynchronous link operates. The asynchronous channel is operating at maximum efficiency when frames arrive contiguously at the receiver. The bits in each frame must be contiguous. The frames do not have to be contiguous.

Asynchronous communications requires a high volume of synchronising signal compared with synchronous communications. Each character or element of data that goes through the communications channel must have a flag to indicate the beginning and end of data or text. These flags have the equate length of one bit each. When you add these flags to the eight data bits required to transmit a byte of data, you add a 20 percent overhead to each byte you transmit.

A protocol is a clear description of the logical method of transmitting information. There is no national or international standard for an asynchronous protocol that requires contiguous communications of asynchronous data frames.

The framing protocol is usually described by a sequence of numbers and letters. The frame that we are using is '8n1' means 1 start bit (always the same, thus omitted), 8 bits of data, no parity bit, 1 stop bit.

As we have say the start bit indicates the beginning of a new data word (this means one single character). It is used to synchronise transmitter and receiver and is always a logical ‘0’ (so the line goes ‘high’ or ‘space’).

Data is transmitted LSB to MSB, which means that the least significant bit (LSB) is transmitted first with 4 to 7 bits of data following, resulting in 5 to 8 bits of data.

A parity bit can be added to the data bits to allow error detection. There are two — actually there are five — kinds of parity: odd and even (plus none, mark and space). Odd parity means that the number of ‘low’ or ‘mark’ steps in the data word (including an optional parity bit, but not the framing bits) is always odd, and even parity that is always even, so the parity bit is set accordingly. It is also possible to set the parity bit to a fixed state or to omit it.

The stop bit does not indicate the end of the word (as it could be derived from its name), it rather separates two consecutive words by putting the line into the quiescent state for a minimum time (that means the stop bit is a logical ‘1’ or ‘mark’) in order for the next start bit to be clearly visible. Called the stop bit or the stop bits, depending on the size of the stop step, normally 1, 1.5 or 2 times the size of a data bit.)
The command-frames that we are using by

1 byte 1 byte 1 byte 1 byte maximum 10 2 bytes

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
<th>Control character</th>
<th>length of the command</th>
<th>Command</th>
<th>Checksum</th>
</tr>
</thead>
</table>

General size of the command-frame is 16 Bytes And its separate in :

1 byte The address of the destination device.
1 byte The address of the source device.
1 byte A control character that define what kind is the frame that transmitted For a command-frame the control character is 1, and for data-frames is 0.
10 bytes The command.
2 bytes The checksum for the detection of errors in the transmission.

The Data-frame description is :

1 byte 1 byte 1 byte maximum 59 2 bytes

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
<th>Control character</th>
<th>Data</th>
<th>Checksum</th>
</tr>
</thead>
</table>

General size of the data-frame is 64 Bytes And its separate in :

1 byte The address of the destination device.
1 byte The address of the source device.
1 byte A control character that define what kind is the frame that transmitted. Of a command-frame the control character is 1, and for data-frames is 0.
10 bytes The data.
2 bytes The checksum for the detection of errors in the transmission.
The Response's frame - that the secondary PC sends - description is:

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
<th>Control charact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

General size of the data-frame is 64 Bytes. And it's structured as:

1 byte  The address of the destination device.
1 byte  The address of the source device.
1 byte  A control character that defines if the transmission was successful or not. For a successful transmission a 0 is transmitted, and for a bad transmission a 1 is transmitted.

Data is transmitted bit-synchronously and word-asynchronously, which means that the size of the bits, the length of the words, is clearly defined while the time between two words is undefined.

The attenuation of the electrical signals and emission of electromagnetic signals that have no meaning can cause the receiver unable to use a signal of a very decreased level, or he cannot separate the information signal from the noise. Also when a Personal computer sends data, a bad set of hardware chips in the port can generate errors in the data stream.

So we have to ensure for error-free information transmission. And because there is no way to avoid the mistakes the only thing we can do is searching and correcting them by using transference data in the proper code.

The most useful ways of searching and correcting the errors in transmission are based on mathematical calculations. Where the sender generates this redundancy data from the bit stream of information data it sends. The sender then appends the redundancy data to the end of the bit stream. The receiver must generate its redundancy data from the information data it receives and compare the results with the redundancy data it received from the sender. A favourable comparison indicates the absence of errors. An unfavourable comparison indicates the presence of errors.

A sender and receiver must use identical techniques of generating redundancy information.

There are three kinds of error-checking techniques:

---

Saini Eleni — T.E.I. of Kavala
K.I.H.O— campus RABOT
— Vertical Redundancy Check (VRC) or Equivalence checking of characters.
— Longitudinal Redundancy Checking (LRC) or Equivalence checking of groups.
— Cyclic Redundancy Check (CRC)

VRC requires the addition of a single bit to each byte of transmitted data to produce a specific parity for the entire byte. The sender and receiver can use either even or odd parity. If the sender is using even parity, it must ensure each byte ends with a bit value that makes the entire byte contain an even number of 1 bits. If the sender is using odd parity, it must ensure each byte ends with a bit value that makes the entire byte contain an odd number of 1 bits.

LRC requires the addition of a single byte (one character) after each string of transmitted bytes (characters). The LRC byte must provide even parity for each longitudinal string of bit positions. The sender and receiver can use either even or odd parity as a VRC in combination with even parity for the LRC. This LRC character, sometimes called a checksum, checks the sum of the binary values for each bit position for all transmitted bytes.

The CRC may take several forms. The selection of a specific communications protocol often dictates the use of a specific type of CRC. The error-checking data produced by a protocol is the remainder after the protocol divides its polynomial value into all the bits that require error checking. Protocols produce CRC data by dividing a constant (derived from the CRC polynomial) into the binary values of all bits contained in a block of data it transmits. The protocol discards the resulting quotient and retains the remainder. The protocol then appends this remainder to the block of bits as a block check character (BCC) or frame check sequence (FCS). A receiving station calculates its own CRC for the block of bits it receives and compares this remainder to the BCC/FCS it receives to determine the presence of errors.

Both the LRC and CRC are more accurate error detection techniques than the VRC. The VRC is good for detecting single-bit errors in single bytes of data. The LRC is good for detecting single-bit and multiple-bit errors in individual bytes of data.

The method of exchanging signals for data flow control between computers and data sets is called handshaking. The most popular and most often used handshaking variant is called XON/XOFF, it's done by software, while other methods are hardware-based.

XON: XOFF

Two bytes that are not mapped to normal characters in the ASCII character set are called XON (DC1, Ctrl-Q, ASCII 17) and XOFF (DC3, Ctrl-S, ASCII 19). Whenever either one of the sides wants to interrupt the data flow from the other (e.g., full buffers), it sends an XOFF (‘Transmission Off’). When its buffers have been
purged again, it sends an XON ("Transmission On") to signal that data can be sent again. (With some implementations, this can be any character.)

XON / XOFF is of course limited to text transmission. It cannot be used with binary data since binary files tend to contain every single one of the 256 characters...

That's why hardware handshaking is normally used with modems, while XON/XOFF is often used with printers and plotters and terminals.

**DTR / DSR**

The ‘Data Terminal Ready’ and ‘Data Set Ready’ signals of the serial port can be used for handshaking purposes, too. Their names express what they do: the computer signals with DTR that it is ready to send and receive data, while the data set sets DSR. With most modems, the meaning of these signals is slightly different: DTR is ignored or causes the modem to hang up if it is dropped, while DSR signals that a connection has been established.

**RTS / CTS**

While DTR and DSR are mostly used to establish a connection, RTS and CTS have been specially designed for data flow control. The computer signals with RTS (‘Request To Send’) that it wishes to send data to the data set, while the data set (modem) sets CTS (‘Clear To Send’) when it is ready to do one part of its job: to send data through the phone wires.

**The software**

Our first step was to find a unit that would handle the serial port, by reading data from the port, writing data to it, providing handshaking signals, and handling the pins of the port. For that purpose we used the ASYNC unit that we found in the internet.
The routines that follow comprise a Turbo Pascal UNIT that will fully implement interrupt-driven serial communications on a fully PC-compatible computer (those machines that use the 8250 or equivalent UART, mapped at the standard addresses). Full simultaneous buffering for both input and output with variable buffer sizes for each port is provided. Unlike many other libraries of a similar nature, up to 4 ports may be active simultaneously (easily modified for more).

For further details, consult the procedure/function headers within the program or check the accompanying ASYNCl1.DOC file.

This product is copyrighted (C) 1988 by Rising Edge Software, Inc. and its author, Mark Schultz. Permission is granted for non-commercial use and distribution. Refer to the file ASYNCl1.DOC for details.

**Version History:**

**V1.1 - Corrected errors that dealt with the handling of the 8259. Also added compensation for "bugs" that are present in some 8250's and equivalent gate arrays (Thanks to Ralph Schraven, Toshiba Europa Gmbh for tracking these problems down).**

**V1.0 - First release**

---

**Status byte definition (C_Status):**

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Input buffer empty</td>
<td>Input buffer full</td>
<td>Output buffer empty</td>
<td>Output buffer full</td>
<td>Input buffer overflow</td>
<td>Output buffer overflow</td>
<td>Hard handshake active (xmit stopped)</td>
<td>Soft handshake active (xmit stopped)</td>
</tr>
</tbody>
</table>

**Control byte definition (C_Ctrl):**

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enable RTS handshake</td>
<td>Enable CTS handshake</td>
<td>Enable software handshake</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Const
C MinBaud = 50;
C MaxBaud = 115200;

{ Base port addresses & interrupt usage }
C MaxPort = 4;
C MaxCom : Byte = C MaxPort;
C PortAddr : Array[1..C MaxPort] Of Word = ($03F8,$02F8,$03E8,$02E8);
C PortInt : Array[1..C MaxPort] Of Word = (4,3,4,3);

{--------------------------------}
Type
C PointerArray = Array[1..C MaxPort] Of Pointer;
C WordArray = Array[1..C MaxPort] Of Word;
C ByteArray = Array[1..C MaxPort] Of Byte;
C CharArray = Array[1..C MaxPort] Of Char;
C BooleanArray = Array[1..C MaxPort] Of Boolean;

{--------------------------------}
Var
C InBufPtr.C OutBufPtr : C PointerArray; { Input/output buffer pointers }
C InHead.C OutHead : C WordArray; { Input/output head pointers }
C InTail.C OutTail : C WordArray; { Input/output tail pointers }
C InSize.C OutSize : C WordArray; { Input/output buffer sizes }
C RTSOn.C RTSOff : C WordArray; { RTS assert/drop buffer points }
C StartChar.C StopChar : C CharArray; { Soft handshake start/stop char }
C Status.C Ctrl : C ByteArray; { STATUS and CONTROL registers }
C Xl3Ptr : C ByteArray;
C PortOpen : C BooleanArray; { Port open/close flags }
C_Temp : Word; { Used for debugging }

{--------------------------------}
Function ComReadCh(ComPort:Byte) : Char;
Function ComReadChW(ComPort:Byte) : Char;
Procedure ComWriteCh(ComPort:Byte; Ch:Char);
Procedure ComWriteChW(ComPort:Byte; Ch:Char);
Procedure SetDTR(ComPort:Byte; Assert:Boolean);
Procedure SetRTS(ComPort:Byte; Assert:Boolean);
Procedure SetOUT1(ComPort:Byte; Assert:Boolean);
Procedure SetOUT2(ComPort:Byte; Assert:Boolean);
Function CTSStat(ComPort:Byte): Boolean;

Saini Eleni — T.E.I. of Kavalas
K.I.H.O – campus RABOT
Function DSRStat(ComPort Byte) : Boolean;
Function RLSStat(ComPort Byte) : Boolean;
Function DCDStat(ComPort Byte) : Boolean;
Procedure SetRTSMode(ComPort Byte, Mode Boolean, RTSOn,RTSOff Word);
Procedure SetCTSMode(ComPort Byte, Mode Boolean);
Procedure SoftHandshake(ComPort Byte, Mode Boolean, Start,Stop Char);
Procedure ClearCom(ComPort Byte, IO Char);
Function ConBufferLen(ComPort Byte, IO Char) Word;
Procedure ComWaitForClear(ComPort Byte);
Procedure ComWrite(ComPort Byte, St String);
Procedure ComWriteLn(ComPort Byte, St String);
Procedure ComWriteWithDelay(ComPort Byte, St String, Dly Word);
Procedure ComReadln(ComPort Byte, Var St String, Size Byte, Echo Boolean);
Function ComExists(ComPort Byte) : Boolean;
Function ComTrueBaud(Baud LongInt) : Real;
Procedure ComParams(ComPort Byte, Baud LongInt, WordSize Byte, Parity Char, StopBits Byte);
Function OpenCom(ComPort Byte, InBufferSize,OutBufferSize Word) : Boolean;
Procedure CloseCom(ComPort Byte);
Procedure CloseAllComs;

IMPLEMENTATION

Uses DOS,CRT;

{$L F:\pascal\Forum\ASYNC.OBJ}$

Const
  C_LER = 1; { 8250 register offsets }
  C_LIR = 2;
  C_LCR = 3;
  C_MCR = 4;
  C_LSR = 5;
  C_MSR = 6;
  C_SCR = 7;

Var
  C OldlNTVec : C PointerArray; { Storage for old hardware INT vectors }
  X Byte; { Used by initialization code }

Procedure INT Handler, External;

{ Hardware interrupts 3 and 4 (vectors $0B and $0C) are pointed to }
{ this routine. It is for internal use only and should NOT be called }
{ directly. Written in assembly language (see ASYNC11.ASM). }

Procedure INT Handler, External.

Design and realisation of a system for information exchange between a number of computers/microcontrollers.
Procedure ComReadCh(ComPort: Byte): Char; External;

- ComPort: Byte -> Port # to use (1 - C.MaxCom)
- Returns character from input buffer of specified port. If the buffer is empty, the port # invalid or not opened, a Chr(0) is returned.
- Written in assembly language for best possible speed (see ASYNCl 1.ASM)

Function ComReadChW(ComPort: Byte): Char; External;

- ComPort: Byte -> Port # to use (1 - C.MaxCom)
- Works like ComReadCh, but will wait until at least 1 character is present in the specified input buffer before exiting. Thus, ComReadChW works much like the ReadKey predefined function. Written in assembly language to maximize performance (see ASYNCl 1.ASM)

Procedure ComWriteCh(ComPort: Byte; Ch: Char): External;

- ComPort: Byte -> Port # to use (1 - C.MaxCom)
- Ch: Char -> Character to send
- Places the character [Ch] in the transmit buffer of the specified port. If the port specified is not open or nonexistent, or if the buffer is filled, the character is discarded. Written in assembly language to maximize performance (see ASYNCl 1.ASM)

Procedure ComWriteChW(ComPort: Byte; Ch: Char): External;

- ComPort: Byte -> Port # to use (1 - C.MaxCom)
- Ch: Char -> Character to send
- Works as ComWriteCh, but will wait until at least 1 free position is available in the output buffer before attempting to place the character.
Design and realization of a system for information exchange between a number of computers/microcontrollers
Procedure SelRTS(ComPort:Byte; Assert:Boolean);

Begin
  If (ComPort<1) Or (ComPort>C MaxCom) Then Exit;
  P := C PortAddr|ComPort|;

  X := Port[P+C_MCR];
  If Assert Then
    X := X Or $02
  Else
    X := X And $FD;
  Port[P+C_MCR] := X;
End;

Procedure SetOUT1(ComPort:Byte; Assert:Boolean);

Begin
  If (ComPort<1) Or (ComPort>C MaxCom) Then Exit;
  P := C PortAddr|ComPort|;

  X := Port[P+C_MCR];
  If Assert Then
    X := X Or $04
  Else
    X := X And $FB;
  Port[P+C_MCR] := X;
End;

Sani Eleni — T.E.I. of Kavalas
K.I.H O-campus RABOT
Design and realisation of a system for information exchange between a number of computers/microcontrollers
Begin
If (ComPort<1) Or (ComPort>C_MaxCom) Then
  CTSStat := False
Else
  CTSStat := (Port[C_PortAddr|ComPort|+C_MSR| And $10] > 0);
End;

Function DSRStat(ComPort:Byte) : Boolean
Begin
If (ComPort<1) Or (ComPort>C_MaxCom) Then
  DSRStat := False
Else
  DSRStat := (Port[C_PortAddr|ComPort|+C_MSR And $20] > 0;
End;

Function RIStat(ComPort:Byte) : Boolean
Begin
If (ComPort<1) Or (ComPort>C_MaxCom) Then
  RIStat := False
Else
  RIStat := (Port[C_PortAddr|ComPort|+C_MSR And $40] > 0;
End;

The Data Set Ready (DSR) line is typically used by a remote station to signal the host system that it is on-line (although not necessarily ready to receive data yet - see CTSStat). A remote station has the DSR line asserted if DSRStat returns TRUE.

The Ring Indicator (RI) line is typically used only by modems, and indicates that the modem has detected an incoming call if RIStat returns TRUE.
Function DCDStat(ComPort: Byte) : Boolean;

Begin
If (ComPort < 1) Or (ComPort > C_MaxCom) Then
  DCDStat := False
Else
  DCDStat := (Port[C_PortAddr[ComPort] + C_MSR] And $80) > 0;
End;

Procedure SetRTSMode(ComPort: Byte; Mode: Boolean; RTSOn, RTSOff: Word);

Begin
If (ComPort < 1) Or (ComPort > C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;
X := C_Ctrl[ComPort];

If (ComPort < 1) Or (ComPort > C_MaxPort) Or (Not C_PortOpen[ComPort]) Then Exit;
X := C_Ctrl[ComPort];
If Mode Then X := X Or $01 Else X := X And $FE.
C_Ctrl[ComPort] := X;

If Mode Then
Begin
    If (RTSOFF >= C_InSize[ComPort]) Then RTSOFF := C_InSize[ComPort] - 1;
    If (RTSOFF > RTSON) Then RTSOFF := RTSON.
    C_RTSOn[ComPort] := RTSON;
    C_RTSOff[ComPort] := RTSOFF;
End;

{/*
 */
{ Procedure SelCTSMode(ComPort:Byte; Mode:Boolean); */
{  * ComPort:Byte -> Port # to use (1 - C_MaxCom); */
{  * Request ignored if out of range or unopened; */
{  * Mode:Boolean -> Set to TRUE to enable automatic CTS handshake; */
{  * */
{  * SetCTSMode allows the enabling or disabling of automated CTS handshaking. If [Mode] is TRUE, CTS handshaking is enabled, which means that: */
{  * if the remote drops the CTS line, the transmitter will be disabled */
{  * until the CTS line is asserted again. Automatic handshaking is disabled */
{  * if [Mode] is FALSE. CTS handshaking and "software" handshaking (provided by the SoftHandshake procedure) ARE compatible and may be used */
{  * in any combination. The actual logic for CTS handshaking is located */
{  * in the communications interrupt driver (see ASYNC11.ASM); */

Procedure SelCTSMode(ComPort:Byte; Mode:Boolean).

Var
    X : Byte;

Begin
    If (ComPort< 1) Or (ComPort>C_MaxPort) Or (Not C_PortOpen[ComPort]) Then Exit;
    X := C_Ctrl[ComPort],
    If Mode Then X := X Or $02 Else X := X And $FD;
    C_Ctrl[ComPort] := X;
End,

{/*
 */
{ Procedure SoftHandshake(ComPort:Byte; Mode:Boolean; Start,Stop:Char); */
{  * ComPort:Byte -> Port # to use (1 - C_MaxCom); */
{  * Request ignored if out of range or unopened; */
{  * Mode:Boolean -> Set to TRUE to enable transmit software handshake; */
{  * Start:Char -> START control character (usually ^Q); */
{  * Defaults to ^Q if character passed is >= <Space>; */
{  * Stop:Char -> STOP control character (usually ^S); */

Saini Eleni — T.E.I. of Kavalas
K.I.H.O — campus RABOT
Procedure SoftHandshake(ComPort:Byte; Mode:Boolean; Start,Stop:Char);

Var
  X : Byte;

Begin
  If (ComPort<1) Or (ComPort>C_MaxPort) Or (Not C_PortOpen(ComPort)) Then Exit;
  X := C_CtrliComPort;
  If Mode Then
    Begin
      If Start=Stop Then Begin Start := ^Q; Stop := ^S; End;
      If Start>$32 Then Start := "Q;
      If Stop>$32 Then Stop := ^S;
      C_StartChar[ComPort] := Start;
      C_StopChar[ComPort] := Stop;
    End
  Else
    X := X And $FB;
  C_Ctrli[ComPort] := X;
End

Procedure ClearCom(ComPort:Byte): IO Char;

ComPort Byte — Port # to use (1 - C_MaxCom).
* Request ignored if out of range or unopened.

IO,Char — Action code: I=Input, O=Output, B=Both
* No action taken if action code unrecognized.

ClearCom allows the user to completely clear the contents of either
* the input (receive) and/or output (transmit) buffers. The "action"
* code" passed in <IO> determines if the input (I) or output (O) buffer
* is cleared. Action code (B) will clear both buffers. This is useful
* if you wish to cancel a transmitted message or ignore part of a
* received message.

Design and realisation of a system for information exchange
between a number of computers/microcontrollers
Procedure ClearCom(ComPort: Byte; IO: Char);

Var
P, X : Word;

Begin
If (ComPort < 1) Or (ComPort > C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;

IO := Upcase(IO);
P := C_PortAddr[ComPort],

Inline($FA),

If (IO = 'I') Or (IO = 'O') Then

    C_InHead|ComPort| := 0;
    C_InTail|ComPort| := 0;
    C_Status|ComPort| := (C_Status|ComPort| And $E) Or $01;

End;
If (IO = '0') Or (IO = 'B') Then

    C_OutHead|ComPort| := 0;
    C_OutTail|ComPort| := 0;
    C_Status|ComPort| := (C_Status|ComPort| And $D3) Or $04;
    X := Port[P+C_LSR] + Port[P+C_MSR] + Port[P+C_IIR];

End;

Inline($FB);

End;

Procedure ComBufferLeft(ComPort: Byte; IO: Char) : Word

Function ComBufferLeft(ComPort: Byte; IO: Char) : Word;

Begin
ComBufferLeft := 0;

Saini Eleni — T E I of Kavala
K I H O — campus RABOT
If (ComPort < 1) Or (ComPort > C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;
IO := Upcase(IO);

If IO = 'I' Then
If C_InHead[ComPort] >= C_InTail[ComPort] Then
  ComBufferLeft := C_InSize[ComPort] - (C_InHead[ComPort] - C_InTail[ComPort])
Else
  ComBufferLeft := C_InTail[ComPort] - C_InHead[ComPort];
End;

If IO = 'O' Then
If C_OutHead[ComPort] >= C_OutTail[ComPort] Then
  ComBufferLeft := C_OutSize[ComPort] - (C_OutHead[ComPort] - C_OutTail[ComPort])
Else
  ComBufferLeft := C_OutSize[ComPort] - (C_OutTail[ComPort] - C_OutHead[ComPort]);
End;

Procedure ComWaitForClear(ComPort:Byte);
Var
  Empty : Boolean;
Begin
  If (ComPort < 1) Or (ComPort > C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;
  Repeat
    Empty := (C_Status[ComPort] And $04) = $04;
    Empty := Empty And ((Port[PortAddr[ComPort] + C_IER] And $02) = $00);
  Until Empty;
End;

Procedure ComWrite(ComPort:Byte; St:String)
Var
  Empty : Boolean;
Begin
  If (ComPort < 1) Or (ComPort > C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;
  Repeat
    Empty = (C_Status[ComPort] And $04) = $04;
    Empty = Empty And ((Port[PortAddr[ComPort] + C_IER] And $02) = $00);
  Until Empty;
End;

Design and realisation of a system for information exchange between a number of computers/microcontrollers
Procedure ComWrite(ComPort:Byte; St:String);

Var X: Byte;

Begin
    If (ComPort<1) Or (ComPort>C_MaxCom) Or (Not C_P ortOp en[ComPort]) Then Exit;
    For X := 1 To Length(St) Do
        ComWriteChW(ComPort,St[X]);
End;

Procedure ComWriteln(ComPort:Byte; St: String);

If (ComPort<1) Or (ComPort>C_MaxCom) Or (Not C_P ortOp en[ComPort]) Then Exit;
For X := 1 To Length(St) Do
    ComWriteChW(ComPort,St[X]);
    ComWriteChW(ComPort,#13);
    ComWriteChW(ComPort,#10);
End;

Procedure ComWriteWithDelay(ComPort:Byte; St String; Dly:Word);

ComPort:Byte -> Port # to use (1 - C_MaxCom)
St String -> String to send
Dly Word -> Time, in milliseconds, to delay between each char.

ComWriteWithDelay will send string <St> to port <ComPort>, delaying for <Dly> milliseconds between each character. Useful for systems that cannot keep up with transmissions sent at full speed.

Saini Eleni - T.E.I. of Kavalas
K.I.H.O. - campus RABOT
Procedure ComWriteWithDelay(ComPort:Byte; St:String; Dly:Word);
Var
  X : Byte;
Begin
  If (ComPort<1) Or (ComPort>C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit;
  ComWaitForClear(ComPort);
  For X := 1 To Length(St) Do
    ComWriteChW(ComPort,St[X]);
    ComWaitForClear(ComPort);
    Delay(Dly);
End;
End;

{♦ *}
{♦ Procedure ComReadln(ComPort:Byte; Var St:String; Size:Byte; Echo:Boolean)}
{♦ *}
{♦ ComPort:Byte -> Port # to use (1 - C_MaxCom). ♦}
{♦ Exits immediately if out of range or port unopened ♦}
{♦ St: String -> Edited string from remote ♦}
{♦ Size:Byte; -> Maximum allowable length of input ♦}
{♦ Echo:Boolean; -> Set TRUE to echo received characters ♦}
{♦ ComReadln is the remote equivalent of the standard Pascal READLN pro- ♦}
{♦ cedure with some enhancements. ComReadln will accept an entry of up to ♦}
{♦ 40 printable ASCII characters, supporting ^H and ^X editing commands. ♦}
{♦ Echo-back of the entry (for full-duplex operation) is optional. All ♦}
{♦ control characters, as well as non-ASCII (8th bit set) characters are ♦}
{♦ stripped. If <Echo> is enabled, ASCII BEL ('G) characters are sent ♦}
{♦ when erroneous characters are intercepted. Upon receipt of a "M (CR), ♦}
{♦ the procedure is terminated and the final string result returned ♦}
{♦ ♦}

Procedure ComReadln(ComPort:Byte; Var St:String; Size:Byte; Echo:Boolean);

Var
  Len,X : Byte;
  Ch : Char;
  Done : Boolean;
Begin
  St := "",
  If (ComPort<1) Or (ComPort>C_MaxCom) Or (Not C_PortOpen[ComPort]) Then Exit,
  Done := False;
  Repeat
    Len := Length(St);
    Design and realisation of a system for information exchange between a number of computers/microcontrollers
Ch := Chr(Ord(ComReadChW(ComPort)) And $7F);

Case Ch Of
  ^H: If Len > 0 Then
    Dec(Len);
    St[0] := Chr(Len);
    If Echo Then ComWrite(ComPort.#8#32#8);
    End
  Else
    ComWriteChW(ComPort.^G);
  ^M: Begin
    Done := True;
    If Echo Then ComWrite(ComPort.#13#10);
    End;
  ^X: Begin
    St :=
    If Len = 0 Then ComWriteCh(ComPort.^G);
    If Echo Then
      For X := 1 to Len Do
        ComWrite(ComPort.#8#32#8);
      End;
    #32..#127: If Len < Size Then
      Inc(Len);
      St[Len] := Ch;
      St[0] := Chr(Len);
      If Echo Then ComWriteChW(ComPort.Ch);
      End
    Else
      If Echo Then ComWriteChW(ComPort.^G);
    Else
      If Echo Then ComWriteChW(ComPort.^G)
    End;

  Until Done;
End;

Function ComExist(ComPort,Byte) : Boolean
·}
ComPort,Byte -> Port # to use (1 - C MaxCom)
·}
Returns FALSE if out of range ♦}
Returns TRUE if hardware for selected port is detected & tests OK ♦}
Function ComExist performs a high-speed short loopback test on the ♦}
selected port to determine if it indeed exists. Use this function ♦}
before attempts to OPEN a port for I/O (although this function is ♦}
called by OpenCom to prevent such an occurrence). ♦}
NOTE! Although pains are taken to preserve the 8250 state before the ♦}
port test takes place. It is nonetheless recommended that this function ♦}
NOT be called while a port is actually OPEN. Doing so may cause the ♦}
port to behave erratically. ♦}

Saini Eleni — T.E.I of Kavalas
K.I.H.O — campus RABOT
Function ComExist(ComPort:Byte) : Boolean;

Const
TestByte1 : Byte = $0F;
TestByte2 : Byte = $F1;

Var
P : Word;
M,L,B1,B2 : Byte;

Begin
ComExist := False;
If (ComPort<l) Or (ComPort>C_MaxPort) Then Exit;
P := C_PortAddr|ComPort|;
M := Port[P+C_MCR];
L := Port[P+C_LCR];
Port[P+C_MCR] := $10;
Port[P+C_LCR] := $80;
B1 := Port[P];
B2 := Port[P+1];
Port[P] := TestByte1;
Delay(20);
If Port[P] <> TestByte1 Then Exit;
Port[P+1] := TestByte2;
Delay(20);
If Port[P] <> TestByte2 Then Exit;
ComExist := True;
Port[P+C_LCR] := $80;
Port[P] := B1;
Port[P+1] := B2;
Port[P+C_LCR] := L;
Port[P+C_MCR] := M;
End;

Function ComTrueBaud(Baud:Longint) : Real

*  Function ComTrueBaud(Baud:Longint) : Real ♦)
*  Baud:Longint -> User baud rate to test ♦}
*  Should be between C_MinBaud and C_MaxBaud ♦}
*  Returns the actual baud rate based on the accuracy of the 8250 divider ♦}
*  The ASYNCl 1  communications package allows the programmer to select ANY ♦}
*  baud rate, not just those that are predefined by the BIOS or other ♦}

Design and realisation of a system for information exchange between a number of computers/microcontrollers
agency. Since the 8250 uses a divider/counter chain to generate its baud clock, many non-standard baud rates can be generated. However, the binary counter/divider is not always capable of generating the EXACT baud rate desired by a user. This function, when passed a valid baud rate, will return the ACTUAL baud rate that will be generated. The baud rate is based on a 8250 input clock rate of 1.73728 MHz.

Function ComTrueBaud(Baud:Longint) : Real;

Var
X: Real;
Y: Word;

Begin
X := Baud;
If X < C_MinBaud Then X := C_MinBaud;
If X > C_MaxBaud Then X := C_MaxBaud;
ComTrueBaud := 115200 / Round($900/(X/50));
End;

Procedure ComParams(ComPort:Byte; Baud:Longint, WordSize:Byte; Parity:Char; StopBits:Byte);

Const
C_Stopbit1 = $00;  { Bit masks for parity, stopbits }
C_Stopbit2 = $04,
C_NoParity = $00;
C_OddParity = $08.

ComParams is used to configure an OPENed port for the desired communications parameters, namely baud rate, word size, parity form and # of stop bits. A call to this procedure will set up the port appropriately, as well as assert the DTR, RTS and OUT2 control lines and clearly all buffers.

Procedure ComParams(ComPort Byte, Baud LongInt, WordSize Byte, Parity Char, StopBits Byte);
C  EvenParity = $18;
C  MarkParity = $28;
C  SpaceParity = $38;

Var
X  Real;
Y,P : Word;
DivMSB,DivLSB : Byte;
WS,SB,PTY : Byte;

Begin
If (CoinPort<l) Or (ComPort>C MaxPort) Or (Not C PortOpen|CoinPort]) Then Exit;

Inline($FA);
P := C_PortAddr|CoinPort];

{ Calculate baud rate divisors }

    {  Calculate baud rate divisors }
    If X < C MinBaud Then X := C MinBaud;
    If X > C MaxBaud Then X := C MaxBaud;
    Y := Round($900/(X/50));
    DivMSB := Hi(Y);
    DivLSB := Lo(Y);

{ Determine parity mask }
{ Default if unknown: No parity }
Case UpCase(Parity) Of
    'N' : PTY := C NoParity;
    'E' : PTY := C EvenParity;
    'O' : PTY := C OddParity;
    'M' : PTY := C MarkParity;
    'S' : PTY := C SpaceParity;
Else
    PTY := C NoParity;
End;

{ Determine stop-bit mask }
{ Default if out of range: 1 stop bit }
Case StopBits Of
    1 : SB := C_StopBit1;
    2 : SB := C_StopBit2;
Else
    SB := C StopBit1;
End;

{ Determine word-size mask }
{ Default if out of range: 8 bit word size }
If (WordSize >= 5) And (WordSize <= 8) Then
    WS := WordSize - 5

Design and realisation of a system for information exchange between a number of computers/microcontrollers
{ Initialize line-control register }
Y := Port[P] + Port[P+C_LSR];
Port[P+C_LCR] := WS + SB + PTY;

{ Initialize baud rate divisor latches }
Port[P+C_LCR] := Port[P+C_LCR] Or $80;
Port[P] := DivLSB;
Port[P+1] := DivMSB;

{ Assert RS232 control lines (DTR,RTS,OUT2) & exit }
Port[P+C_MCR] := $0B;
ClearCom(ComPort,'B');
InLine($FB);

Function OpenConi(ConiPort:Byte; InBufrerSize,OutBufrerSize:Word):Boolean

ConiPort:Byte -> Port # to OPEN (1 - C MaxCom)
InBufferSize:Word -> Requested size of input (receive) buffer
OutBufferSize:Word -> Requested size of output (transmit) buffer
Returns success/fail status of OPEN request (TRUE if OPEN successful)

OpenCom must be called before any activity (other than existence check) takes place. OpenCom initializes the interrupt drivers and serial communications hardware for the selected port, preparing it for I/O. Memory for buffers is allocated on the Pascal "heap", thus freeing data-segment memory for larger more data-intensive programs. Once a port has been OPENed, a call to ComParams should be made to set up communications parameters (baud rate, parity and the like). Once this is done, I/O can take place on the port.

OpenCom will return a TRUE value if the opening procedure was successful, or FALSE if it is not.

Function OpenCom(ComPort,InBufferSize,OutBufferSize:Word):Boolean

Var
TempVec : Pointer;
P : Word;
IntLn.X : Byte;

Begin
{ Ensure that port was not previously open }

OpenCom = False;
If (ComPort<1) Or (ComPort>C_MaxPort) Or C_PortOpen[ComPort] Then Exit;
If Not ComExist(ComPort) Then Exit;

{ Clear any pending activity from 8250 interrupt queue }

{ Inline($FA), }
P := C_PortAddr[ComPort];
Port[P+C_IER] := $0D;

{ Set up interrupt vectors & 8259 P(C }

IntLn := C_PortInt[ComPort];
GetIntVec(8+IntLn, TempVec);
If C_OldINTVec[IntLn] <> TempVec Then

C_OldINTVec[IntLn] := TempVec;
SetIntVec(8+IntLn,(a)Int_Handler);
Port[$21] := Port[$21] And ((01 SHL IntLn) XOR $FF);
X := Port[$21];
End;

{ Allocate memory for I/O buffers }

C_InSize(ComPort) := InBufferSize;
C_OutSize(ComPort) := OutBufferSize;
GetMem(C_InBufPtr[ComPort], InBufferSize);
GetMem(C_OutBufPtr[ComPort], OutBufferSize);

{ Set up default parameters for port }

C_RTSOn[ComPort] := InBufferSize - 2,
C_RTSOff[ComPort] := InBufferSize - 1;
C_StartChar[ComPort] := "Q",
C_StopChar[ComPort] := "S",
C_PortOpen[ComPort] := True;
OpenCom := True;

{ End }

{..........................................................}

(* Procedure CloseCom(ComPort Byte) *)
(*
(* ComPort Byte -> Port # to close *)
(* Request ignored if port closed or out of range *)
(*
(* CloseCom will un-link the interrupt drivers for a port, deallocate it's *)
(* buffers and drop the DTR and RTS signal lines for a port opened with *)

Design and realisation of a system for information exchange between a number of computers/microcontrollers
Ihc OpenCom function. It should be called before exiting your program to ensure that the port is properly shut down.

NOTE: CloseCom shuts down a communications channel IMMEDIATELY, even if there is data present in the input or output buffers. Therefore, you may wish to call the ComWaitForClear procedure before closing the ports.

Procedure CloseCom(ComPort: Byte):

Var
    ClosePort: Boolean;
    IntLn: Word;

Begin
    If (ComPort<l) Or (ComPort>C_MaxPort) Or (Not C_PortOpen[ComPort]) Then Exit;

    \ Drop RS232 control lines (DTR,RTS,OUT2) and reset 8250 interrupt mode \\
    Inline($FA);
    P := C_PortAddr(ComPort);
    Port[P+C_MCR] := 0;
    Port[P+C_IER] := 0;
    C_PortOpen[ComPort] := False;

    \ Reset INT vectors & 8259 PIC if all COMs on selected INT are closed \\
    IntLn := C_PortInt[ComPort];
    ClosePort := True;
    For X := 1 To C_MaxCom Do
        If C_PortOpen[X] And (C_PortInt[X] = IntLn) Then
            ClosePort := False;
    If ClosePort Then
        Begin
            Port[$21] := Port[$21] Or ($01 SHR IntLn);
            X := Port[$21];
            SetIntVec(8+IntLn,C_OldIntVcc[11]);
        End;

    \ Deallocate buffers \\
    FreeMem(C_InBufPtr[ComPort],C_InSize[ComPort]);
    FreeMem(C_OutBufPtr[ComPort],C_OutSize[ComPort]);
    Inline($FB);

End;

Procedure CloseAllComs:

\ Procedure CloseAllComs will CLOSE all currently OPENed ports. See the CloseCom function. 

Sami Eleni — T.E.I. of Kavala
K.I.H.O — campus RABOT
Design and realisation of a system for information exchange between a number of computers/microcontrollers.
Which contains some of the procedures in assembly language

Status byte definition (C Status):

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td></td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control byte definition (C Ctrl):

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IDEAL

SEGMENT DATA WORD PUBLIC

Externally accessed (TP program) variables

Note: Most of these variables are declared as arrays in the main program; i.e. C InSize : Array[1, 4] Of Word

EXTRN C lnBufPtr DWORD : Pointer to input buffers
EXTRN C OutBufPtr DWORD : Pointer to output buffers
EXTRN C InSize WORD : Size (bytes) of input buffers

Saini Eleni — T.E.I. of Kavalas
K.I.H.O — campus RABOT
EXTRN C OutSize WORD  ; Size (bytes) of output buffers
EXTRN C InHead WORD   ; Input (receive) head pointers
EXTRN C OutHead WORD   ; Output (transmit) head pointers
EXTRN C InTail WORD    ; Input (receive) tail pointers
EXTRN C OutTail WORD   ; Output (transmit) tail pointers
EXTRN C RTSOn WORD     ; Point at which RTS line is asserted
EXTRN C RTSOff WORD    ; Point at which RTS line is dropped
EXTRN C StartChar BYTE ; Start character for soft handshake
EXTRN C StopChar BYTE  ; Stop character for soft handshake
EXTRN C Status BYTE    ; Status byte (see above)
EXTRN C Ctrl BYTE      ; Control byte (see above)
EXTRN C PortOpen BYTE  ; Port-open flags
EXTRN C PortAddr WORD   ; Base address of ports
EXTRN C MaxCom BYTE    ; Highest port # defined (single byte)
EXTRN C XL3Ptr BYTE    ; String index: detection of XL3 ID
EXTRN C Temp WORD      ; Used for debugging

IEE EQU 1   ; Interrupt enable register
IER EQU 2   ; Interrupt identification register
LCR EQU 3   ; Line control register
MCR EQU 4   ; Modem control register
LSR EQU 5   ; Line status register
MSR EQU 6   ; Modem status register
SCR EQU 7   ; 8250 scratch register

ENDS DATA

Code segment declaration

SEGMENT CODE BYTE PUBLIC
ASSUME CS CODE, DS DATA

Externally accessible procedures defined here

PUBLIC INT Handler
PUBLIC ComReadCh
PUBLIC ComReadChW
PUBLIC ComWriteCh
PUBLIC ComWriteChW

Subroutines that are used internally

Subroutine: ChkPort
Function: Check port parameter(s), ensure that port is OPEN
Entry: AL <- Port # (1 - C_MaxCom)
Exit: AL -> Adjusted port # (0 - 3)

Design and realisation of a system for information exchange between a number of computers/microcontrollers
AH -> Status register
BX -> Byte array index
DX -> Base address of port

PROC ChkPort	FAR

Determine if port # is valid

CMP AL, [C MaxCom] ; Port # > Maximum port # ?
JA ChkErr ; Yes, exit w/error
CMP AL, 0 ; Port # = 0 (invalid port #)
JZ ChkErr ; Yes, exit w/error
DEC AL ; AL <- Adjusted port #

Check if port open

XOR BH, BH ; BX <- Byte array index
MOV BL, AL
MOV AH, [C PortOpen+BX] ; AH <- Port-open flag
CMP AH, 0 ; Port open ?
JZ ChkErr ; No, exit w/error

Get status register and base port address in DX

MOV AH, [C Status+BX] ; AH <- Status register
SHL BL, 1 ; BX <- Word array index
MOV DX, [C PortAddr+BX] ; DX <- Port address
SHR BL, 1 ; BX <- Byte array index
STC ; Set carry (valid return)
RET ; Exit

Here if error

ChkErr CLC ; Clear carry (invalid return)
RET

ENDP ChkPort

*******************************************************************************

: Subroutine: PutChar
: Function: Place character in transmit buffer (transmit a character)
: Entry: AL <- Port # (0-3)
AH <- Status register
BX <- Byte array pointer
CH <- Character to put in buffer
DX <- Base address of port

: Destroyed: ES, SI, DI

: Note: Port address passed is NOT checked for validity.
: Subroutine ChkPort will create the proper entry environment

PROC PutChar	FAR

Saini Eleni — T.E.1 of Kavalas
K I H O — campus RABOT
Design and realisation of a system for information exchange between a number of computers/microcontrollers.
Interrupt service routine for INT3, INT4

INT3 typically used by COM2, COM4
INT4 typically used by COM1, COM3

PROC INT_Handler FAR

PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
PUSH DS
PUSH ES
PUSH BP

MASM
MOV AX, SEG DATA ; AX <- Current data segment

IDEA1
MOV DS, AX ; Set new data segment

L1:
XOR BX, BX ; BX <- Port # (start at 0)

Identify active port

MOV AL, [C PortOpen + BX] ; AL <- Port-open flag
CMP AL, 0 ; Port open ?
JZ IntID2 ; No, don't check
SHL BL, 1 ; BX <- Word array index
MOV DX, [C PortAddr + BX] ; DX <- Base address of port
SHR BL, 1 ; BX <- Byte array index
ADD DL, IR ; Add in offset for IR
IN AL, DX ; AL <- COMn IR
TEST AL, 0000000b ; Interrupt active on this port ?
JZ INT Active ; (Bit 0 = 0 if active)
INC BL ; Bump port #
CMP BL, [C MaxCom] ; All ports checked ?
JB IntID1 ; No, continue

Here to leave interrupt handler

MOV AL, 20h ; AL <- EOI Acknowledge code
OUT 20h, AL ; To 8259 PIC
POP BP ; Restore environment
POP ES

Saini Eleni — T E.I. of Kavalas
K.I.H.O — campus RABOT
Design and realisation of a system for information exchange between a number of computers/microcontrollers.
Modem status change

ComMsc: NOP

Get modem status & control/status registers

Msc0: ADD DL,MSR :DX <- 8250 MSR port address
IN AL,DX :AL <- 8250 Modem status
SUB DL,MSR :Back to base

MOV CL,[C_Status+BX] :CL <- Status register
MOV CH,[C_Ctrl+BX] :CH <- Control register

Control CTS status flag based on CTS status and handshake enable

TEST CH,00000010b ;CTS handshake enabled?
JZ Msc2 ;No, ignore CTS status
TEST AL,00010000b ;CTS asserted?
JNZ Msc1 ;Yes, enable transmit
OR CL,01000000b ;CTS inactive - hard handshake on
JMP Msc2

Msc1: AND CL,10111111b ;CTS active - hard handshake off
Msc2: MOV [C_Status+BX],CL ;Save status flags

Determine if transmitter should be enabled

Msc3: ADD DL,IER :DX <- 8250 IER port address
IN AL,DX :AL <- Interrupt enable register
OR AL,00000010b ;Enable transmitter by default
TEST CL,11000100b ;Enable transmitter?
JZ Msc4 ;Yes
AND AL,11111101b ;No - disable transmitter
JMP INT_Id

Msc4: OUT DX,AL ;Update IER
JMP INT_Id

Character transmitted

Register usage:
AH: Status register
DX: Array pointer
DL: Port address
SI: Output tail pointer
DI: Output head pointer

Get status and control registers

ComXmt: MOV AH,[C_Status+BX] ;AH <- Status register
TEST AH,00000010b ;Buffer empty?
JNZ Xmt2 ;Yes, stop transmitter
Character received

Register usage:
AL : Status register
AH : Control register
BX : Array index
CL : Character received
CH : Temporary storage
DX : Port address
SI : Input tail pointer
DI : Input head pointer

ComRev: IN AL,DX ;CL <- Received character
MOVC CL,AL

Check for software handshake

MOV AL,[C_Status+BX] ;AL <- Status byte
MOV AH,[C_Ctl+BX] ;AH <- Control byte

Design and realisation of a system for information exchange between a number of computers/microcontrollers
Software handshake enabled?

No, don't check software handshake

STOP TRANSMIT character?

START TRANSMIT character?

Soft-handshake character received.
Activate or deactivate transmitter depending on software handshake enabled?

Yes, deactivate transmitter

No, don't check software handshake

STOP TRANSMIT character?

START TRANSMIT character?

Check for XL3 verification sequence

Note: This code is here to provide the capability to communicate with a Pyrotronics XL3 fire alarm panel (since this is why these routines were developed in the first place). In almost all cases they will never be needed by the "normal" user.

XL3 mode active?

No, ignore

Save status & exit

Set soft-handshake flag (stop)

Routines in MSC2 require status in Cl

Reset soft-handshake flag (start)

Save status & exit

End of string?

Yes, exit now

Save string index

Exit

Possible XL3 string match

CH <- String index

Recover array index

Bump index

End of string?

Yes, send answerback

Save string index

Exit

XL3 identify sequence detected. Send answerback

CH <- Pointer to answerback string

Put status byte in AH (for PutChar)

Put port # in AL (for PutChar)

CH <- Char from answerback string

End of string?

Yes, exit now

Save SI (modified by PutChar)

Transmit character

Recover SI

Point to next char in string

Continue until string sent

Put status byte in AH (for PutChar)

Put port # in AL (for PutChar)

Save SI (modified by PutChar)

Transmit character

Recover SI

Point to next char in string

Continue until string sent

Put status byte in AH (for PutChar)

Put port # in AL (for PutChar)

Save SI (modified by PutChar)

Transmit character

Recover SI

Point to next char in string

Continue until string sent
Resel siring index

XL3 identify string mismatch - reset pointer & continue

Recv3d: MOV BX, DI
MOV [C_XL3Pr+BX].0

.Recover array index
.Reset string index

Clear buffer empty flag / check for buffer overflow

Recv3e: AND AL, 11111110b
TEST AL, 00000001b
JZ Rev4
OR AL, 000010000b
JMP Rev10

.Clear buffer-empty flag
.Buffer full?
.No, continue
.Set overflow flag
.Exit

Bump receive buffer pointer

Recv4: SHL BL, 1
MOV DI, [C_InHead+BX]
MOV SI, [C_InTail+BX]

.DBX <- Word array index
.DI <- Input head pointer
.SI <- Input tail pointer

INC DI

.Bump buffer pointer

CMP DI, [C_InSize+BX]
JB Rev5
XOR DI, DI

.Head > buffer size?
.No, continue

Store character in buffer

Recv5: PUSH BX
MOV [C_InHead+BX].DI
SHL BL, 1
LES BX, [C_InBufPtr+BX]

.Save word array pointer
.Save updated input head pointer
.BX <- Doubleword array index
.ES BX <- Pointer to buffer

MOV [ES BX+DI], CL

.Save character in buffer
.Recover word array pointer

Check for full buffer

CMP SI, DI
JNZ Rcv6
OR AL, 000000100b

.Tail = Head?
.If Tail <> Head, buffer is full
.Set buffer-full flag
.Reset RTS and exit

Check for near-full buffer (buffer used >= RTSOff)

Recv6: CMP SI, DI
JBE Rev7
SUB SI, DI
MOV DI, [C_InSize+BX]

.Tail <= Head?
.Yes, use standard formula
.SI <- Tail - Head
.DI <- Input buffer size

Rev7: SUB DI, SI
CMP DI, [C_RTSOff+BX]
JB Rev9

.DI <- Head - Tail (amt used)
.Used < Limit?
.Yes, leave RTS on

Design and realisation of a system for information exchange between a number of computers/microcontrollers
Buffer is (near) full, force RTS off & exit

: RTS handshake enabled?
  : No, exit now

MOV CH, AL
: Keep status byte
ADD DL, MCR
: DX <- Address of modem control reg.
IN AL, DX
: AL <- MCR
AND AL, 1111110b
: Disable RTS
OUT DX, AL
: Update MCR
SUB DL, MCR
: DX <- Base of port
MOV AL, CH
: Recover status byte

Rev9: SHR BL, 1
: BX <- Byte array index
Rev10: MOV [C_Status+BX], AL
: Save status byte

The following code corrects a "bug" present in some 8250's

ADD DL, IER
: DX <- Interrupt Enable Register
IN AL, DX
: AL <- IER
MOV AH, AL
: Save IER
AND AL, 1111110b
: Mask off transmit interrupt
OUT DX, AL
: Send to IER
MOV AL, AH
: Restore original IER state
OUT DX, AL
JMP INT_1d
: Check for pending INTs and exit

ENDP INT Handler

Start of Pascal low-level procedures

Function ComReadCh(ComPort:Byte) : Char

PROC ComReadCh VAR

Check port # for validity

CLI
: Interrupts disabled
MOV BX, SP
: BX <- Stack pointer
MOV AL, [SS:BX+4]
: AL <- Port #
CALL ChkPort
: Check port # for validity
JNC ComRd1
: Invalid port, exit

Check buffer, return null if empty

Saini Eleni — T.E.I. of Kavalas
K.I.T.H.O — campus RABOT
MOV AH,[C_Status+BX]  ;AH <- Status byte
TEST AH,00000001b      ;Buffer empty?
JZ ComRd2             ;No, continue normally

Buffer empty, port not open or port # invalid - exit

ComRd1: MOV AX,0100h    ;Return null (port error or empty)
          AL <- Char from buffer
          EBX <- Char from buffer
          Enable interrupts

Increment tail pointer
NOTE Entry point for ComReadChW

ComRd2:  SHR BL,1        ;BX <- Word array index
          MOV SI,[C_InTail+BX]       ;SI <- Tail pointer
          MOV DI,[C_InHead+BX]       ;DI <- Input head pointer
          INC SI                     ;Bump tail pointer
          CMP SI,[C_InSize+BX]       ;Tail past end of buffer?
          JB ComRd3                 ;No, continue
          XOR SI,SI                  ;Yes, reset pointer
ComRd3:  MOV [C_InTail+BX],SI ;Save updated tail pointer
          Get character from buffer
          SHR BL,1                  ;BX <- Pointer array index
          LES BX,[C_InBufPtr+BX]     ;ES.BX <- Pointer to input buffer
          MOV CH,[ES.BX+SI]          ;CH <- Character from buffer

Clear FULL and OVERFLOW flags
Check for empty buffer

XOR BH,BH                ;Byte array index
MOV BL,AL                 ;Reset FULL and OVERFLOW status bits
AND AH,11011011b          ;Reset FULL and OVERFLOW status bits
CMP DI,SI                 ;Head = Tail (buffer empty?)
JNZ ComRd4                ;No, continue normally
OR AH,00000001b           ;Yes, set empty flag
ComRd4:  MOV [C_Status+BX],AH ;Save status byte

Check for RTS assert (Used <= RTSOn)
Variable USED (# of used bytes in buffer) calculated as:
IF (Head >= Tail) THEN
  Used = Head - Tail
ELSE
  Used = BufferSize - (Tail - Head)

SHL BL,1                ;BX <- Word array index
CMP DI,SI               ;Head >= Tail?
JAE ComRd5              ;Yes, use alternate formula
SUB SI,DI                ;Sl <- Tail - Head
MOV DI,[C_InSize+BX]    ;DI <- Input buffer size

Design and realization of a system for information exchange between a number of computers/microcontrollers
SUB DI, SI  ; DI <- Amount of buffer used
CMP DI, [C RTSOn+BX]  ; Used > Limit ?
JA ComRdX  ; Yes, not ready for receive

Here to assert RTS

SHR BL, 1  ; BX <- Byte array index
MOV AL, [C Ctrl+BX]  ; AL <- Control byte
TEST AL, 00000001  ; RTS handshaking enabled ?
JZ ComRdX  ; No. exit now

ADD DL, MCR  ; DX <- 8250 MCR port address
IN AL, DX  ; AL <- 8250 MCR
OR AL, 00000010  ; Assert RTS
OUT DX, AL  ; Send to port
JMP ComRdX  ; Exit

ENDP ComReadCh

Function ComReadChW(ComPort: Byte) : Char

PROC ComReadChW FAR

Check for valid port

MOV BX, SP  ; BX <- Stack pointer
MOV AL, [SS:BX+4]  ; AL <- Port #
CALL ChkPort  ; Port # valid ?
INC ComRWX  ; No. exit now

Wait for character

MOV AH, [C_Status+BX]  ; AL <- Status byte
TEST AL, 00000001  ; Input buffer empty ?
JNZ ComRdW1  ; Yes, continue waiting
CLI  ; Disable interrupts
JMP ComRd2  ; Proceed with normal read

Here if error

MOV AL, 0  ; Return null result
RET 2

ENDP ComReadChW

Procedure ComWriteCh(ComPort: Byte, Var Ch: Char)
PROC ComWriteCh FAR

Check port # for validity

MOVBX, SP ;Point BX at parameters
MOV AL, [SS: BX+6] ;AL <- Port #
MOV CH, [SS: BX+4] ;CH <- Character to send
CALL ChkPort ;Check port for validity
JNC ComWriteCh ;Exit if not valid

CLI ;Disable interrupts
CALL PutChar ;Place character in appropriate buffer
STI ;Enable interrupts

ComWriteCh: RET 4 ;Exit

ENDP ComWriteCh

PROC ComWriteChW FAR

Check port # for validity

MOVBX, SP ;Point BX at parameters
MOV AL, [SS: BX+6] ;AL <- Port #
MOV CH, [SS: BX+4] ;CH <- Character to send
CALL ChkPort ;Check port for validity
JNC ComWriteChW ;Exit if not valid

: Wait for buffer to open up

ComWriteChW1: MOV AH, [C Status+BX] ;AH <- Status byte
TEST AH, 00101000b ;Buffer filled ?
JNZ ComWriteChW1 ;Yes, loop until open

CLI ;Turn off interrupts
CALL PutChar ;Place character in buffer
STI ;Enable interrupts

ComWriteChW2: RET 4 ;Exit

ENDP ComWriteChW

XL3Len EQU 3 ;Length of XL3 string

Design and realisation of a system for information exchange between a number of computers/microcontrollers
Here are some programs which using this unit:

```pascal
program testreadch;
uses async.crt;
var
  pi : boolean;
  c : char;
begin
  pi := OpenCom(1,1000,1000);
  ComParams(1,4800,8,'n',1);
  repeat
    ComWrite(1,' aek ');
    c:=ComReadCh(1);
    if ord(c)<>0 then write(c);
  until keypressed;
end;

program testframe; { slave }
uses async.crt;
var
  pi : boolean;
  r : byte;
  ch : C_CharArray;
  summ : Word;
begin
  pi := OpenCom(1,1000,1000);
  ComParams(1,4800,8,'n',1);
  for r:=1 to 4
    Do
    begin
      ch[r]:=ComReadCh(1);
      summ :=summ + ord(ch[r]);
    end;
    if summ<>0
      then ComWriteLn(1,'There is an error. send the message again!')
      else writeln(ch[3]);
end;
```

Simi Elena — T.E.I. of Kavalas
K.I.H.O. — campus RABOT
program testreadch;
uses async,crt;

pi : boolean;
begin
  pi := OpenCom(1,1000,1000);
  ComParams(1,110,8,'n',1);
repeat
  ComWriteCh(1,'a'),
until keypressed;
end.

program testreadln;
uses async;

var
text : string;
d1 : boolean.
begin
d1 := OpenCom(1,1000,1000);
  ComReadln(1,text,40,true);
end.

program testframe. { Master }
uses async,crt;

var
Port num : Byte;
p1 : Boolean;
Sum : Longint;
C,els : Char;
Text : String;

begin

Function ReadWrite(Ports:Byte) : Longint;
Var
  r : Byte,
a : Char;
  O_a,Summ : Longint;
begin
  a := 'a';
  O_a := Ord(a);
  Summ := 0;
  for r:=1 to 3
    Do begin
Design and realisation of a system for information exchange between a number of computers/microcontrollers
Summ := Summ + O a;
ComWrite(Port num,Char(O a));
O a = O a+1;
end{do};
ReadWrite := Summ.
end.

procedure Epalithefs(sum:Longint);
var
  epal : Longint;
begin
  epal := sum - 256;
  ComWrite(Port_num,Char(epal));
end;

program testframe;
uses async, crt;
const
  Port num = 2;
var
  P1,check : boolean;
  ch : CCharArray;
Psumm,Esumm,Asumm : Byte;
begin
  gotoXY(27,7); write('givene the number of your port
  gotoXY(27,41); readfPort num);
  repeat
    P1 := OpenCom(Port_num,1000,1000);
    ComParams(Port_num,2400,8,'n',1);
    sum := (ReadWrite(Port num)),
    Epalithefs(sum);
    C:=ComReadchW(Port_num);
    ComReadLnt(Port_num,Text,40,TRUE);
    if c = 'Q' then
      begin
        gotoXY(13,9),write('There is an error, send the message again');
        end
      else
      begin
        gotoXY(13,9),write('OK');
        end;
    gotoXY(13,13),write ('Do you want to send anything else? Y/N'),
    read(ch);
    until (ch = 'n') or (ch = 'N');
  end;
Function Check_adds(r,byte; ch,char) : Boolean.

Var
m add,s add : char;
Begin
m add := 'a';
s add := 'b';
Check adds := FALSE;
IF ((r = 1) and (ch = m add)) or ((r = 2) and (ch = s add)) THEN
    Check_adds := TRUE;
End;

Function Athroisma(ch,char) : Byte.

Var
Asumun : Byte;
Begin
writeln('O'.Asunim);
Asunun := Asumun + ord(ch);
Athroisma := Asumun;
writeln('S '.Asumun);
End;

Procedure Elenxos_Echo(Esumm,Byte);
Begin
IF Esumm <> 0 THEN
    Begin
    WriteLn('Esumm := '.Esumm);
    ComWriteLn(Portnum.'Q');
    ComWriteLn(Port_num.'There is an error. send the message again!');
    End
ELSE
    Begin
    Write(ch[3]);
    write('M');
    ComWriteChW(Port num.'O');
    ComWriteLn(Port_num.'O.K.');
    End;
End;

BEGIN
PI := OpenCom(Port_num,1000,1000);
ComParams(Port nuni,2400,8,'n'. 1);

Design and realisation of a system for information exchange
between a number of computers/microcontrollers.
Psumm := 0;
Assumm := 0;
REPEAT
  r := r + 1;
  writeln('r =', r);
  ch[r] := ComReadChW(Port num);
  writeln(ch[r], Ord(ch[r]));
  IF ( r = 1 ) or ( r = 2 ) THEN
    check := check adds(r, ch[r]);
  IF ( check = FALSE ) and ( r = 3 ) THEN
    Begin
      Psumm := 205;
UNTIL r = 4;
Eleaxos Echo(Psumm),
4 THEN
  Psumm := Athroisma(ch[r]);
  writeln('P '.Psumm);
end;

program lestframe; { Master }
uses async.crt.Dos;
Const
  Port num = 2;
Type
  Buffer = Array[1..1000] of Byte;
  Message = String[60];
  BufPtr = ^Buffer;
  MessPtr = ^Message;

  pi : Boolean;
  Sum : Longint;
  C : Char;
  Text,Source,Destination : String;
  Aframe : Array[1..64] of String;
  Len,ij : Byte;
  Buf1 : BufPtr;
  Mess : Message;
  MPtr : MessPtr;

Procedure WriteFrame( Ports Byte; MP Pointer; L:Integer; DA String );

Saini Eleni — T.E.I. of Kavalas
K.I.H O— campus RABOT
Design and realization of a system for information exchange between a number of computers/microcontrollers.


BuPr1 = 'Buffer1';
Bu1 = Buffer1;
Bu11 = BufPr1;
Imgang = String;
Pl.chck = boolean;
r.Psuum = byte;
ch = C CharArray;
Bleg.lleg = Integer;

Function Check_adds(ch1,ch2:char) : Boolean;
Var
  m_add,s_add: char;
Begin
  m_add := 'a';
  s_add := 'b';
  Check_adds := FALSE;
  IF ( ch1 = s_add ) and ( ch2 = m_add ) THEN
      Check_adds := TRUE;
{[IF]}
End;

Procedure PutBuf(insert:String, Bleg Integer, ch3:Byte);
Var
  i,j: Integer;
Begin
  IF ch3 = 1 THEN
      Begin
        {FOR j := 1 TO length[insert][4] DO
          Begin
          End;
        }
        FOR i := 1 TO length(insert) DO
          Begin
            Buf11 Bleg + i := insert [i+3];
          End;
      Bleg := Bleg + ( length(insert) - (5+ch3));
  End;

Procedure Elenxos_Echo(Esumni:Byte);
Begin
  IF Esumni <> 0 THEN
      Begin
        Writeln ('Esumni = ',Esumni);
        ComWriteLn(Port_num,'Q');
        ComWriteLn(Port_num,'There is an error. send the message again!');
      End;

Sanini Eleni — T.E.I. of Kavalas
K I H O — campus RABOT
This two last programs are not finished completely but there are may help.

But finally we found out that this unit wasn't what we were looking for. And the reason is simple because in ComReadCh function, that Reads data from the port the returned character, in case something is not correct (invalid or not open port) or if the buffer is empty, is a zero. Which is very easy to be confused with a zero contained in the data.

There is also another unit that is very interesting, also founded in internet, but and also rejected for the same reason. The address is:

So finally we found the UART unit, that we have been used but the time left was so little...

And I can only offer you the following programs, after the unit:

{ Unit UART - serielle I/O v3 07/91,08/92,01/93 }
{ by Peter Mandrellia, P Mandrellia@HOT gun de }
{ Dieser Quelltext ist Public Domain. }
{$B-,R-,S-,V-,F-,I-,A+}


Das Senden von Daten erfolgt mit SendByte (ohne CTS-Handshake) oder mit HSendByte (mit CTS-Handshake).

Sobald die Funktionen RRing und Carrier genutzt werden, ob ein Klingelzeichen bzw. ein Carrier am Modem anliegt.


---

interface

uses dos;

{SIFNDEF DPMI}
const Seg0040 = $40;
{SENDIF}
c

const coms = 4;  { Anzahl der unterstützten Schnittstellen }

ua array[1, coms] of word = ($3f8, $2f8, $3e8, $2c8),
datainout = 0;  { UART-Register-Offsets }
intenable = 1;
inrds = 2;     { Read }
ifocrl = 2;    { Write }
linecntl = 3;
modemcntl = 4;
linesstat = 5;
modemstat = 6;
scratch = 7;
UartNone = 0.  { Ergebnisse von ComType }

Saimi Eleni — T.E.I. of Kavalas
K I H O — campus RABOT
Uart8250 = 1;
Uart16450 = 2;
Uart16550 = 3;
Uart16550A = 4.

NoFifo = $00;  \{ Triggerlevel bei 16550-Chips \}
FifoTL1 = $07;
FifoTL4 = $47;
FifoTL8 = $87;
FifoTL14 = $C7;

type partype = (None, Podd, Pxxxx, Peven);  \{ mögliche Paritats-Typen \}

{ Parameter für Schnittstelle einstellen }
{ no : Nummer (1-4)  }
{ address : I/O-Adresse, 0 -> Adresse wird beibehalten }
{ _irq : Interrupt-Nummer (z.B. 3 für IRQ3, 4 für IRQ4, 0, 15)  }

procedure SetComParams(no: byte, address: word, _irq: byte);

{ Schnittstelle aktivieren }
{ Schnittstelle Nummer des UART-Chips ermitteln }

function ComType(no: byte): byte;  \{ Typ des UART-Chips ermitteln \}

procedure SetUart(conno: byte, baudrate: longint; parity: partype;  
{ Schnittstelle aktivieren }
{ Schnittstelle Nummer der Schnittstelle 7 oder 8  
{ Stops : Stop-Bits (1 oder 2)  }

procedure ActivateCom(no: byte; buffersize: word; FifoTL: byte);  

procedure ReleaseCom(no: byte);  \{ Schnitte desaktiv. Puffer freig. \}

{ Schnittstelle aktivieren }
{ Schnittstelle Nummer der Schnittstelle 7 oder 8  
{ Stops : Stop-Bits (1 oder 2)  }

procedure Receive(no: byte, var b: byte): boolean;  \{ Byte holen, falls vorh. \}

function peek(no: byte, var b: byte): boolean;  \{ Dito, aber Byte bleibt im Puffer \}

function received(no: byte): boolean;  \{ Tauschen, ob Daten vorhanden \}

function receive(no: byte, var b: byte): boolean;  \{ Dito, aber Byte bleibt im Puffer \}

procedure SendByte(no, b: byte);  \{ Byte senden \}

procedure SendByte(no, b: byte);  \{ Byte senden, mit CTS-Handshake \}

procedure PutByte(no, b: byte);  \{ Byte im Puffer hinterlegen \}

function ringing(no: byte): boolean;  \{ Telefon klingelt \}

function carrier(no: byte): boolean;  \{ Carrier vorhanden \}

Design and realisation of a system for information exchange  
between a number of computers/microcontrollers
function getCTS(no:byte):boolean. (True = (cts=1))
procedure DropDlr(no:byte);         (DTR=0 setzen)
procedure SetDlr(no:byte);          (DTR=1 setzen)
procedure DropRls(no:byte);         (RTS=0 setzen)
procedure SetRls(no:byte);          (RTS=1 setzen)
procedure SendBreak(no:byte);       (Break-Signal)

function slrs(l:longint):string;
var s : string;
begin
  str(l,s);
srls:=s;
end;

{ Modecom-Satus-Register }
{ Ring Indicator: Klingelsignal }
{ Data Carrier Delect }
{ Modem Control Register }

{ Interrupts sperren }
{ Interrupts freigeben }

procedure cli; inline($fa);        (Interrupts sperren)
procedure sit; inline($fb);         (Interrupts freigeben)

procedure com1server, interrupt;
begin
  if inlcom2[1] then port[$a0] :=$20;
  port[$20] :=$20;     (Interrupt-Controller resetten)
  buffer[1]'[bufs][1]:=port[sta][1];
procedure com2server; interrupt;
begin
  if intcom[2] then port[f$0] = $20;
  port[f$20] = $20;
end;

procedure com3server; interrupt;
begin
  if intcom[3] then port[f$0] = $20;
  port[f$20] = $20;
end;

procedure com4server; interrupt;
begin
  if intcom[4] then port[f$0] = $20;
  port[f$20] = $20;
  buffer[4] = port[ua[4]];
end;

procedure com1FIFOserver; interrupt;
begin
  if port[ua[1]] = intids[1] and 4<>0 then
  repeat
    buffer[1] = port[ua[1]];
    until not odd(port[ua[1]] + linestat[1]);
  if intcom[1] then port[f$0] = $20;
  port[f$20] = $20;  // Interrupt-Controller resetten
end;

procedure com2FIFOserver; interrupt;
begin
  if port[ua[2]] = intids[1] and 4<>0 then
  repeat
    until not odd(port[ua[2]] + linestat[2]);
  if intcom[2] then port[f$0] = $20;
  port[f$20] = $20;
end;

procedure com3FIFOserver; interrupt;
begin
  if port[ua[3]] = intids[1] and 4<>0 then
  repeat
    Design and realisation of a system for information exchange
    between a number of computers/microcontrollers
procedure com4FIFOServer; interrupt;
begin
  if port[ua|4]+intids) and 4<>0 then
    repeat
      buffer[4]'[bufl4]' := port[ua|4];
      incr(bufl4); if bufl4 = bufsize4 then bufi(4) := 0;
      until not odd(port[ua|4]+linectrl));
  if intcom2[4] then port[$a0] :=$20;
  port|$20| :=$20;
end;

--- UART-Typ ermitteln ---------------------------------

{ Hinweis: Die Erkennung des 16550A funktioniert nur bei Chips. }
{ die weitgehend kompatibel zum Original-16550A von NS }
{ sind. Das gilt allerdings für die meisten verwendeten }
{ 16500A's - ich schätze ca. 97-99% }

function ComType(no:byte):byte; { TYP des UART-Chips ermitteln }
var
  uart, lsave, ssave : word,
  isave, iir, $5a : byte;
begi
  uart := ua|no|;
  lsave := port[uart+linectrl],
  port[uart+linectrl] := lsave xor $ff,
  if port[uart+linectrl]<>lsave xor $ff then
    ComType := UartNone
  else begin
    port[uart+linectrl] := lsave;
    ssave := port[uart+scratch];
    port[uart+scratch] := $5a;
    if port[uart+scratch]<>$5a then
      ComType := Uart8250 { kein Scratchpad vorhanden }
    else begin
      port[uart+scratch] := $5a;
      if port[uart+scratch]<>$5a then
        ComType := Uart8250 { kein Scratchpad vorhanden }
      else begin
        isave := port[uart+intids];
        port[uart+fifoclrl] := 1;
        iir := port[uart+intids];
        if isave and $80<>0 then port[uart+fifoclrl] := 0,
        if iir and $40<>0 then ComType := Uart16550A
        else if iir and $80<>0 then ComType := Uart16550
        else ComType := Uart16450;
      end;
    end;
  end;
procedure SetComParams(no:byte. address:word: irq:byte);
begin
if (no>=1) and (no<coms) then begin
  if address<>0 then ua[no] := address;
  irq[no] := irq;
  intmask[no] := (1 shl (_irq and 7));
  inlcom2[no] := (_irq>7);  { 2. Interrupt-Controller }
end;
end;

procedure setuart(comno:byte. baudrate:longint: parity:paritytype,
                   wlength:stops:byte);
var uart:word;
begin
  uart:=ua[comno];
  port(uart+linectrl) :=$80;
  port(uart+datainout) :=lo(word(15200 div baudrate));
  port(uart+datainout+1) :=hi(word(15200 div baudrate));
  port(uart+linectrl) := (wlength-5) or (stops-1)*4 or ord(parity)*8;
  port(uart+modemctrl) :=$0b;
  if port(uart+datainout)<>0 then; { dummy }
end;

procedure clearstatus(no:byte);
begin
  if port(ua[no]+datainout)<>0 then;
  if port(ua[no]+linsat)<>0 then;
  if port(ua[no]+modcmstat)<>0 then;
  if inlcom2[no] then port[$a0] :=$20;
  port[$20] :=$20;
end;

function IntNr(no:byte):byte;
begin
  if irq[no]<8 then IntNr :=irq[no]+8
  else IntNr := irq[no]+$68;
end;

procedure ActivateCom(no:byte. buffersize word: FifoTl:byte);
var p : pointer,
    i byte;
begin
if active[no] then begin
    error('Schnittstelle '+strs(no)+' bereits aktiviert!').
else if (no<1) or (no>coms) or (irq[no]=0) then
    error('Schnittstelle '+strs(no)+' (noch) nicht unterstüzt!').
else
    active[no]:=true:
    bufsiz[no|] := buffersize, { Puffer anlegen }
getmem(buffer[no], buffersize).
buf[no] := 0, bufo[no] := 0,
fillchar(buffer[no], bufsize[no], 0);

IF (fifo1 > 0)
THEN BEGIN
    Port[(ua[no] + fifoctrl)] := fifo1,
    IF ((Port[(ua[no] + intida)] AND $40) = 0)
    THEN BEGIN
        Port[(ua[no] + fifoctrl)] := 0;
        fifo1 := NoFifo;
    END;
END;

IF (fifo1 > 0)
THEN CASE no OF
    1  :  p:=com1FIFOserver,
    2  :  p:=com2FIFOserver,
    3  :  p:=com3FIFOserver,
    4  :  p:=com4FIFOserver,
END {CASE}
ELSE CASE no OF
    1  :  p:=com1server,
    2  :  p:=com2server,
    3  :  p:=com3server,
    4  :  p:=com4server,
END {CASE}

getinvoc(Innr(no), savecom[no]); { IRQ setzen }
setinvoc(Innr(no), p), { Int bei Empfang }
if incom[2] then
    port[$1] := port[$1] and (not inmask[no]) { Int freigeben }
else
clearstatus(no).

procedure releasecom(no byte);
begin
    if not active[no] then
        error('Schnittstelle '+strs(no)+' nicht aktiv!')
    else begin

Saini Eleni — T.E.I. of Kavala
K.I.H.O — campus RABOT
 Design and realisation of a system for information exchange between a number of computers/microcontrollers
peck := false
else begin
  b := buffer[no]^|bufo[no]|
  peck := true.
end;
end;

begin
  while (port[ua][no]+linesstat] and $20) = 0 do;
  port[ua][no] := b.
end;

procedure sendbyte(no, b:byte); { Byte senden }
begin
  while (port[ua][no]+linesstat] and $20) = 0 do;
  port[ua][no] := b.
end;

procedure hsendbyte(no, b:byte); { Byte senden, mit CTS-Handshake }
begin
  while (port[ua][no]+modemstat] and $10) = 0 do;
  while (port[ua][no]+linesstat] and $20) = 0 do;
  port[ua][no] := b.
end;

procedure hsendbyte(no, b:byte); { Byte senden, mit CTS-Handshake }
begin
  while (port[ua][no]+modemstat] and $10) = 0 do;
  while (port[ua][no]+linesstat] and $20) = 0 do;
  port[ua][no] := b.
end;

procedure putbyte(no, b:byte); { Byte im Puffer hinterlegen }
begin
  if bufo[no]<>0 then bufo[no] := bufs[no]
  else dec(bufo[no]);
end;

procedure putbyte(no, b:byte); { Byte im Puffer hinterlegen }
begin
  if bufo[no]<>0 then bufo[no] := bufs[no]
  else dec(bufo[no]);
end;

procedure flushinput(no:byte); { Receive-Puffer löschen }
begin
  bufo[no] := bufs[no];
end;

function ring(no:byte):boolean;
begin
  ring := (port[ua][no]+modemstat] and MS_RI)<>0
end;

function carrier(no:byte):boolean; { Carrier vorhanden }
begin
  carrier := (port[ua][no]+modemstat] and MS_DCD)<>0;
end;

procedure DropDtr(no byte), { DTR=0 setzen }
begin
  port[ua][no]+modemctrl] := port[ua][no]+modemctrl] and (not MC_DTR);
end;

procedure SelDtr(no byte); { DTR=1 setzen }
begin
  port[ua][no]+modemctrl] := port[ua][no]+modemctrl] or MC_DTR;
end;

Saini Eleni — T.E.I. of Kavalaş
K.I.H O— campus RABOT
procedure DropRls(no:byte);  \{ RTS=0 set/en \}
begin
    port[ua][no]+modemctrl := port[ua][no]+modemctrl \& (not MC RTS);
end.

procedure SelRls(no:byte);  \{ RTS=1 set/en \}
begin
    port[ua][no]+modemctrl := port[ua][no]+modemctrl \| MC RTS;
end.

{ True -> Modem (oder entsprechendes Ger., t) ist bereit, Daten zu empfangen } 

function GetCTS(no:byte):boolean;
begin
    getcts := ((port[ua][no]+modemstat \& $10)<>0) and
        ((port[ua][no]+linestat \& $20)<>0);
end.

function ticker:longint;
begin
    ticker := meml[Seg0040:$6c];
end.

procedure SendBreak(no:byte);  \{ Break-Signal \}
var
teiler, word,
    btime : longint,
    t0, longint,
begin
    CLI;
    port[ua][no]+linectrl := port[ua][no]+linectrl \| $80;
    teiler := port[ua][no] + 256*port[ua][no] + 1;
    port[ua][no]+linectrl := port[ua][no]+linectrl \& $7f;
    STI;
    btime := teiler DIV 200;
    if (btime<1) THEN btime := 1;
    t0 := ticker;
    inc(ticker, ticker);
    Port[ua][no]+linectrl := port[ua][no]+linectrl \& $40; \{ set break \}
    repeat
        until (ticker>btime) or (ticker<t0);
    Port[ua][no]+linectrl := port[ua][no]+linectrl \& $bf; \{ clear break \}
end.

begin
    exitsave := exitproc;
    exitproc := @comexit;
end.

Design and realisation of a system for information exchange between a number of computers/microcontrollers
program testreadch;
uses uart, crt,
    var
        test : byte,
        tl : boolean,
    begin
        test := 0;
        tl := False;
        ActivateCom(1,1000,0),
        SetUart(1,2400, Pnone, 8, 1);
        gotoXY(27,9); write('Send your message master!'),
        gotoXY(27,19), write('PRESS ANY KEY FOR END'),
        repeat
            tl := Receive(1, test),
            if tl = TRUE then
                begin
                    i := i+1;
                    gotoXY(7+i, 13); write(chr(test)),
                    tl := False,
                end,
            until keypressed;
        end.

program testreadch,
uses uart, crt,
    var
        a : Byte,
        pi : boolean,
        c : char,
    begin
        a := 0;
        ActivateCom(1,1000,0),
        SetUart(1,2400, Pnone, 8, 1),
        gotoXY (15,8); write('I am waiting for an ASCII character'),
        gotoXY (15,10), write(' PRESS ANY KEY FOR END '),
        repeat
            i := i+1;
            SendByte(1,i),
        end.
pi := Receive(1,a);
gotoXY (27,13).
if a<>0 then
    write(chr(a),' = ',a,' '),
until keypressed,
end

program testreadch,
uses uart,crt;
var
    a,i : Byte,
    pl : boolean;
    c : char,
begin
    a:=0;
    ActivateCom( 1,1000,0),
    SetUart( 1,2400,Pnone,8,1),
gotoXY (15,8), write('I am waiting for an ASCII character'),
gotoXY (15,10), write('PRESS ANY KEY FOR END'),
repeat
    i:=i+1,
    SendByte(1,i),
    p1 := Receive(1,a),
gotoXY (27,13),
    if a<>0 then
        write(chr(a),' = ',a,' '),
    until keypressed,
end

Those programs have been also used for a demonstration in a communication system of one terminal and two PCs, connected like following.
REFERENCE

System integration for the IBM 2 and PC

http://www.ece.uiuc.edu/ece291/class.recourses/serial.html.