

Software control of the parallel interface

The line scan cameras of the series *CCDL*, as well as various other devices of the *EURECA Messtechnik GmbH*, are attached to the parallel interface. Generally the *EPP protocol* according to the *IEEE-1284* standard is used, since this enables a good hardware implementation. Almost each PC has an interface, which can be configured to the EPP mode. Depending upon the manufacturer of the I/O chip these interfaces are to be treated for somewhat differently. If the used interface should not support an EPP mode, then the protocol can be emulated also with an interface in the BPP -, PS2 or ECP mode.

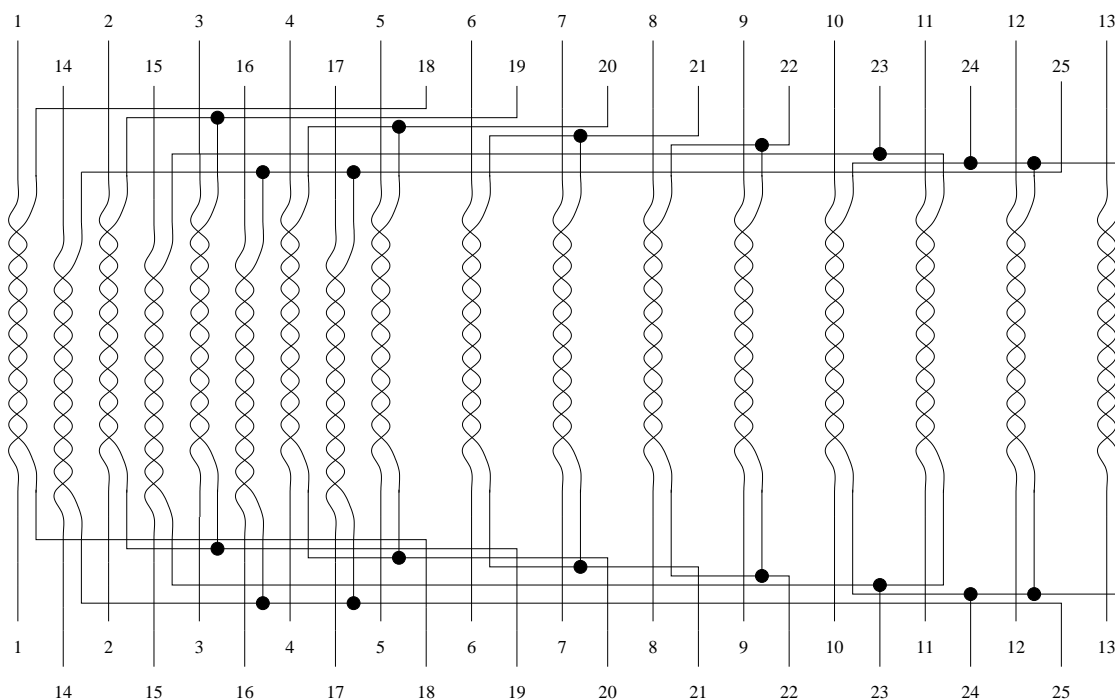
In the first section the physical function of the parallel interface and their logical control is described. Afterwards the different interface types, as well as the interrupt handling, are discussed and in the end programming examples are shown, which can be implemented easily (quick and dirty). All sample programs refer, if not differently proven, on the line scan camera of the type *CCDL-20LH1-8*. For all other line scan cameras the sample programs can be taken over, as within the declaration area the values for the pre-pixel, pixel number and post-pixel are adapted. With all program examples in the quick and dirty style timeouts are not considered!

Important:

For a programming as simple as possible the interface should be configured in the BIOS to the EPP mode!

Note: ECP+EPP requires already a increased programming effort!

To use the high transmission speed you should use always cables after IEEE-1284 standard. For cables with Sub-D-plug the allocation looks as follows:



Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | The physical behavior of the parallel interface: | 3 |
| 1.1 | Data lines | 3 |
| 1.2 | Control lines | 3 |
| 1.3 | User lines | 3 |
| 1.4 | Timing | 4 |
| 1.4.1 | Write access | 4 |
| 1.4.2 | Read access | 4 |
| 1.4.3 | Times | 5 |
| 2 | The logical interface | 5 |
| 2.1 | Port addresses | 5 |
| 2.1.1 | Data port | 6 |
| 2.1.2 | Status port | 6 |
| 2.1.3 | Control port | 7 |
| 2.1.4 | EPP Address port | 7 |
| 2.1.5 | EPP data port | 7 |
| 2.1.6 | ECP data port | 7 |
| 2.1.7 | ECP status port | 7 |
| 2.1.8 | ECP control port | 7 |
| 3 | Software control | 9 |
| 3.1 | BPP (SPP bidirectional) | 9 |
| 3.2 | PS2 | 10 |
| 3.3 | Combination: BPP + PS2 | 11 |
| 3.4 | EPP | 12 |
| 4 | Interrupt | 13 |
| 4.1 | Allocation IRQ→Vector | 13 |
| 4.2 | Interrupt controller | 13 |
| 4.3 | Setting the interrupt vectors | 14 |
| 4.4 | IRQ of the parallel interface | 14 |
| 4.5 | Program example | 14 |
| 5 | Operating systems | 15 |
| 6 | Quick and Dirty | 16 |
| 6.1 | EPP 1.9 hardware direction protocol | 16 |
| 6.2 | EPP 1.9 software direction protocol | 17 |
| 6.3 | EPP 1.7 | 18 |
| 6.4 | Polling the INTR line | 19 |

1 The physical behavior of the parallel interface:

1.1 Data lines

The interface possesses 8 data lines, by which the data communication is made into both directions. These can be set and read both by the computer, as well as by the peripheral device. You have to take care, that at no point in time both the computer and the peripheral device set the data lines.

1.2 Control lines

The data communication on the parallel interface is controlled by 5 physical control lines, whose meaning are described briefly in the following:

1. **nWRITE:**

This line determines whether the operation is a writing (data from the computer to the peripheral device) or a reading access (data from the peripheral device to the computer). Is this line physical 0 ($< 0.8V$), then it is writing access, otherwise ($> 2.0V$) a reading access. This negated behavior is signaled by the n before the WRITE.

2. **nDSTRB:** This line signals a data exchange and possesses during the access time a 0 value.

3. **nASTRB:**

This line signals an address exchange and is in principle equivalent to the data exchange, can however be treated differently by the peripheral device (e.g. controlling several devices at a parallel interface). Only one of the both STRB-lines can be used at one time.

4. **WAIT:**

This line is active high, contrary to the others, and signals the end of the access. The line is set, if the peripheral device is finished with the taking over of the data (write access) or the supplying of the data (read access) and held until the peripheral device is ready to process new data.

5. **INTR:**

This line releases a hardware interrupt in the computer.

1.3 User lines

Additionally to the control lines there are still 4 further lines:

1. **nInit:** This line is active low and releases a software reset with some peripherals devices.

2. **User-1:** This line can be used freely as an input

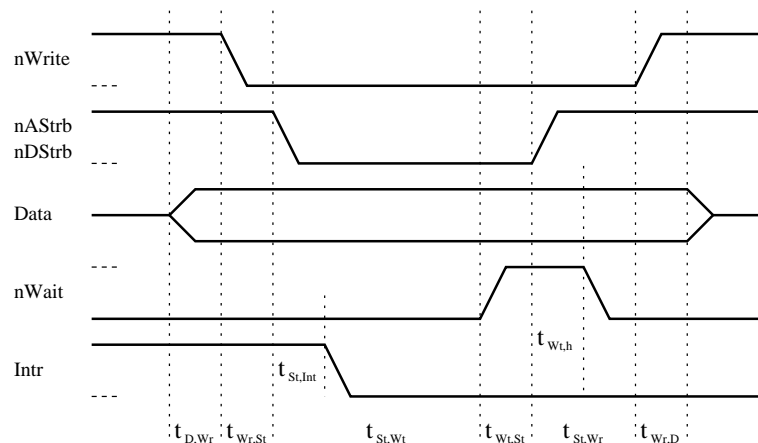
3. **User-2:** This line can be used freely as an input

4. **User-3:** This line can be used freely as an input

1.4 Timing

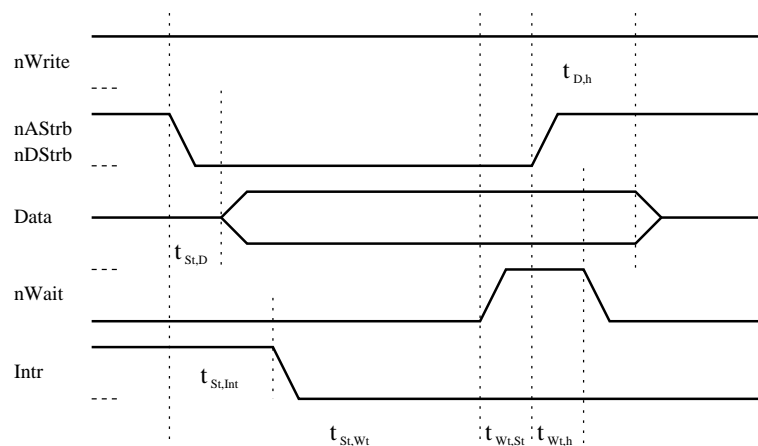
The timing diagram shows the temporal behaviour of the signals:

1.4.1 Write access



First the $nWRITE$ line is set. Afterwards the computer puts the data on the bus. Short time later follows the setting of the $nDSTRB$ line. Now the computer waits, until the $WAIT$ line of the peripheral device is set, at the most however $10\mu s$. Thereafter the $nDSTRB$ line is reset. Shortly after that the computer switches the data lines into the high impedance status and resets thereafter the $nWRITE$ line.

1.4.2 Read access



The $nWRITE$ line remains unset! The computer sets the $nDSTRB$ line, on which the peripheral device puts the data on the bus. Now the computer waits, until the $WAIT$ line is set by the peripheral device, at the most however $10\mu s$. Thereafter the $nDSTRB$ line is reset and the peripheral device switches the data lines into the high impedance status.

Note: With the line scan cameras of the EURECA Messtechnik GmbH after the read out of the sensor information no more $nWait$ signal is set. This means, that with the last 16 bytes a timeout occurs. This is to be considered by the software with BPP and PS2 interfaces!

1.4.3 Times

In the context of the EPP protocol the following times must be kept:

| Symbol | Description | Source | min. | typ. | max. | Unit |
|--------------|---|----------|------|------|------|---------|
| $t_{D,Wr}$ | Data before Write | Computer | 0 | | | ns |
| $t_{Wr,St}$ | Write before Strobe | Computer | 0 | | | ns |
| $t_{St,D}$ | Data after Strobe | Camera | 20 | 30 | 50 | ns |
| $t_{St,Int}$ | Interrupt reset after Strobe | Camera | 50 | 70 | 100 | ns |
| $t_{St,Wt}$ | Wait after Strobe, at 125kByte/s 250kByte/s 500kByte/s 1MByte/s 2MByte/s | Camera | 7 | 8 | 8,5 | μs |
| | | | 3,5 | 4 | 4,25 | μs |
| | | | 1,7 | 2 | 2,1 | μs |
| | | | 0,9 | 1 | 1,1 | μs |
| | | | 0,4 | 0,5 | 0,6 | μs |
| $t_{Wt,St}$ | Strobe after Wait | Computer | 0 | | | ns |
| $t_{Wt,h}$ | Wait hold | Camera | 0 | 20 | 30 | ns |
| $t_{D,h}$ | Data hold | Camera | 0 | 20 | 30 | ns |
| $t_{St,Wr}$ | Write after Strobe | Computer | 0 | | | ns |
| $t_{Wr,D}$ | Data after Write | Computer | 0 | | | ns |

2 The logical interface

2.1 Port addresses

As interface between the program and the physical lines 11 port addresses are available:

| | Address | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-------------------|-------------|----------|-------|-------|---------|--------|---------|--------|--------|
| Data port | PORT+0 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| Status port | PORT+1 | WAIT | INTR | USER1 | USER3 | USER2 | X | X | TMOUT |
| Control port | PORT+2 | X | X | PCD | IRQEN | nASTRB | nINIT | nDSTRB | nWRITE |
| EPP addr. port | PORT+3 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| EPP data port 0 | PORT+4 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| EPP data port 1 | PORT+5 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| EPP data port 2 | PORT+6 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| EPP data port 3 | PORT+7 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| ECP fifo port | PORT+ 0x400 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| ECP con-fig port | PORT+ 0x401 | COMPRESS | INTR | IRQ2 | IRQ1 | IRQ0 | 1 | 1 | 1 |
| ECP con-trol port | PORT+ 0x402 | Mode2 | Mode1 | Mode0 | ErrIntr | dmaEn | Service | Full | Empty |

2.1.1 Data port

The data port is the interface between the program and the data lines. This port can be read and written both. During reading the momentary physical status of the data lines is indicated.

During writing the data lines are set on the bit values corresponding the physical statuses. There are two possibilities, how this is implemented:

1. The outputs of the driver are open collectors with pull ups. This is the old standard solution and available on many computers. Both, computers and peripheral device, drive the lines against ground, while a high signal is produced only over pull up resistances. Thus the danger of short-circuits is prevented.

Disadvantage: In order to read the information of the peripheral device, all outputs must be set on High by the computer first, since they form a wired AND gate.

2. The outputs of the driver are Tri-State. This means, that the outputs can possess a high impedance status or one of the two logic levels. This version was introduced by the company IBM for the Personal-Systems-2 (PS2) and forms the today's standard.

Disadvantage: For reading the peripheral device data the driver must be switched explicitly into the high impedance status. Note: If the driver is not in the high impedance status, it can come to short-circuits with the peripheral device!

2.1.2 Status port

The status port shows the physical levels of the status inputs. In general this port can be only read. An exception forms here the timeout bit. This is set with EPP operation, if the peripheral device did not set the WAIT line within 10 μ s. In order to delete this timeout bit, there are again two possibilities:

1. The company XIRCOM introduced, that the TIMEOUT bit is deleted by the writing of a 1 on bit 0 of the status port. Writing a 0 on the bit does not cause modification.
2. The company MircoSoft introduced, that the TIMEOUT bit takes over the status of the bit 0 during writing: If one writes a 1, then the bit is set and reseted only with writing a 0.

In order to consider both possibilities, one proceeds as follows: first one writes a 1 (delete the TIMEOUT bits with XIRCOM) and afterwards a 0 (delete the TIMEOUT bits at Microsoft and no modification with XIRCOM). In general the other bits of the status word do not have a meaning, since they can be only read. It is safer to write these with the statuses which one read before:

```
buffer = inportb (port+1);
outportb (port+1, buffer | 0x01);
outportb (port+1, buffer & 0xfe);
```

The second important bit of the status port is the bit 6 (INTR). This releases, if the interrupt vector is correctly initialized in the computer, an interrupt. Since this is normally quite much programming effort, one can poll the bit for simple applications:

```
...
while (inportb (port+1) & 0x40 == 0); ...
```

2.1.3 Control port

One can in general only write to the control port. Reading normally gives back the last written byte (exception the two bits with X, which are not certain, but normally 1). The written bits are put inverted on the appropriate signal lines. This applies to the lower 4 bits of the control byte.

The bit PCD is used with the PS2-Port and EPP port (without hardware direction protocol) to switch the driver chip into the high impedance status. A 1 in this bit means, that the driver is high impedance, a 0 means, that the driver is on output. The bit `IRQEN` is used in order to activate the interrupt. A 0 means, that the interrupt is not used, the `INTR` line can however be polled.

2.1.4 EPP Address port

The EPP address port is available only, if in the BIOS the EPP function of the interface is activated. By reading or writing on this port the entire protocol, as described above, will pass through automatically. As strobe the address strobe line is used.

2.1.5 EPP data port

The EPP data port is available only, if in the BIOS the EPP function of the interface is activated. By reading or writing on this port the entire log, as described above, will pass through automatically. As strobe the data strobe line is used. With an word access (16 bits) or a 32 bit access the protocol is passed through 2 or 4 times according to the appropriate bytes. This starts with the LSB (least significant byte). As address for the port the EPP data port 0 is always to be used!

2.1.6 ECP data port

The ECP data port is available only, if in the BIOS the ECP function of the interface is activated and is not necessarily for the operation of EPP able components.

2.1.7 ECP status port

The ECP status port is available only, if in the BIOS the ECP function of the interface is activated and is not necessarily for the operation of EPP able components.

2.1.8 ECP control port

The ECP control port is available only, if in the BIOS the ECP function of the interface is activated and needed, if the ECP+EPP mode is used. The bit 7 sets thereby the EPP mode. First however the standard mode must be set:

```
outportb (port+0x402 ,0x00);  
outportb (port+0x402 ,0x80);
```

In the following you receive a summary of the single ports and their meaning with different interfaces:

| Port- offset | SPP + ECP-Mode 000 | | | PS/2 + ECP-Mode 001 | | | FIFO = ECP-Mode 010 | | | ECP = ECP-Mode 011 | | | EPP + ECP-Mode 100 | | | TEST = ECP-Mode 110 | | | CONFIG = ECP-Mode 111 | | |
|-----------------|---------------------|---------------------|---------|---------------------|---------------------|---------------------|---------------------|-------|---------|---------------------|---------------------|---------|----------------------|----------------------|----------------------|---------------------|---------------------|---------------------|-----------------------------|---------------------|--|
| | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | |
| 0x000 | — | Data | — | Data | — | Status | — | — | — | ECP address FIFO | Status | — | Data | Status | Timeout Reset | — | Status | — | — | — | |
| 0x001 | Status | — | — | Status | — | — | — | — | — | — | Status | — | Status | — | Control Puffer | Control Puffer | Control Puffer | Control Puffer | Status | — | |
| 0x002 | Control Puffer | Control Latch | — | Control Puffer | Control Latch | Control Puffer | Control Latch | — | — | Control Puffer | Control Latch | — | Control Puffer | Control Latch | Control Puffer | Control Puffer | Control Puffer | Control Puffer | Control Puffer | Control Latch | |
| 0x003 | — | — | — | — | — | — | — | — | — | — | — | — | EPP address | EPP address | EPP address | — | — | — | — | — | |
| 0x004 | — | — | — | — | — | — | — | — | — | — | — | — | EPP data | EPP data | EPP data | — | — | — | — | — | |
| 0x005 | — | — | — | — | — | — | — | — | — | — | — | — | EPP data (16 bit) | EPP data (16 bit) | EPP data (16 bit) | — | — | — | — | — | |
| 0x006 | — | — | — | — | — | — | — | — | — | — | — | — | EPP data (32 bit) | EPP data (32 bit) | EPP data (32 bit) | — | — | — | — | — | |
| 0x007 | — | — | — | — | — | — | — | — | — | — | — | — | EPP data (32 bit) | EPP data (32 bit) | EPP data (32 bit) | — | — | — | — | — | |
| 0x008 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x009 | — | — | — | — | — | — | — | — | — | ECP data FIFO | ECP data FIFO | — | — | — | — | Test | Test | Test | Configuration Register A | — | |
| 0x00A | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | Configuration Register B | — | |
| 0x00B | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x00C | ECP ext. Control | ECP ext. Control | — | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | — | — | ECP ext. Control | ECP ext. Control | — | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | ECP ext. Control | |
| 0x00D | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x00E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x00F | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x010 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x011 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x012 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x013 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x014 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x015 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x016 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x017 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x018 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x019 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01A | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01B | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01C | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01D | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |
| 0x01F | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | |

| Host- Pin-Nr. | SPP + ECP mode 000 | | | PS/2 + ECP mode 001 | | | FIFO = ECP mode 010 | | | ECP = ECP mode 011 | | | EPP + ECP mode 100 | | | TEST = ECP mode 110 | | | CONFIG = ECP mode 111 | | |
|------------------|--------------------------|--------------------------|---------|--------------------------|--------------------------|---------|--------------------------|--------------------------|---------|--------------------------|--------------------------|---------|--------------------------|--------------------------|--------------------------|---------------------|-------|---------|-----------------------|-------|---------|
| | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | Control | read | write | Control |
| 1 | nSTRB Output | nSTRB Output | — | nSTRB Output | nSTRB Output | — | nSTRB Output | nSTRB Output | — | nSTRB Output | nSTRB Output | — | nWrite Output | nWrite Output | nWrite Output | — | — | — | — | — | — |
| 2 - 9 | PD<0:7> Bidirectional | PD<0:7> Bidirectional | — | PD<0:7> Bidirectional | PD<0:7> Bidirectional | — | PD<0:7> Bidirectional | PD<0:7> Bidirectional | — | PD<0:7> Bidirectional | PD<0:7> Bidirectional | — | PD<0:7> Bidirectional | PD<0:7> Bidirectional | PD<0:7> Bidirectional | — | — | — | — | — | — |
| 10 | nACK Input | nACK Input | — | nACK Input | nACK Input | — | nACK Input | nACK Input | — | nACK Input | nACK Input | — | Intr Input | Intr Input | Intr Input | — | — | — | — | — | — |
| 11 | BUSY Input | BUSY Input | — | BUSY Input | BUSY Input | — | BUSY Input | BUSY Input | — | PeriphAck Input | PeriphAck Input | — | nWait Input | nWait Input | nWait Input | — | — | — | — | — | — |
| 12 | PE Input | PE Input | — | PE Input | PE Input | — | PE Input | PE Input | — | nAckReversion Input | nAckReversion Input | — | User-1 Input | User-1 Input | User-1 Input | — | — | — | — | — | — |
| 13 | SLCT Input | SLCT Input | — | SLCT Input | SLCT Input | — | SLCT Input | SLCT Input | — | Slot Input | Slot Input | — | User-3 Input | User-3 Input | User-3 Input | — | — | — | — | — | — |
| 14 | nAFD Output | nAFD Output | — | nAFD Output | nAFD Output | — | nAFD Output | nAFD Output | — | HostAck Output | HostAck Output | — | nDStrb Output | nDStrb Output | nDStrb Output | — | — | — | — | — | — |
| 15 | nERR Input | nERR Input | — | nERR Input | nERR Input | — | nERR Input | nERR Input | — | nPeriphAck Input | nPeriphAck Input | — | User-2 Input | User-2 Input | User-2 Input | — | — | — | — | — | — |
| 16 | nINIT Output | nINIT Output | — | nINIT Output | nINIT Output | — | nINIT Output | nINIT Output | — | nReverseRqst Output | nReverseRqst Output | — | nBit Output | nBit Output | nBit Output | — | — | — | — | — | — |
| 17 | nSLIN Output | nSLIN Output | — | nSLIN Output | nSLIN Output | — | nSLIN Output | nSLIN Output | — | nSlin Output | nSlin Output | — | nAStrb Output | nAStrb Output | nAStrb Output | — | — | — | — | — | — |
| 18 - 35 | GND | GND | — | GND | GND | — | GND | GND | — | GND | GND | — | GND | GND | GND | — | — | — | GND | GND | — |

Port and pin allocation of the different interface types

BPP

3 Software control

3.1 BPP (SPP bidirectional)

The most easy interface: bidirectional, not PS2.

In order to get the logical sequence described at the start, the program would have to look approximately so:

Writing a byte:

```
outportb (port+2, 0x01);
outportb (port, data);
outportb (port+2, 0x03);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10));
if (counter == 10)
    timeout = 1;
outportb (port+2, 0x01);
outportb (port+2, 0x00);
```

Reading a byte:

```
outportb (port, 0xff);
outportb (port+2, 0x02);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10))
if (counter == 10)
    timeout = 1;
data = inportb(port);
outportb (port+2, 0x00);
```

PS2

3.2 PS2

With the PS2 port the driver chip must be set explicitly for high impedance, or activated. This is made by the PCD bit of the control port (bit No. 5):

Writing a byte:

```
outportb (port+2, 0x21);
outportb (port+2, 0x01);
outportb (port, data);
outportb (port+2, 0x03);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10));
if (counter == 10)
    timeout = 1;
outportb (port+2, 0x01);
outportb (port+2, 0x21);
outportb (port+2, 0x20);
```

Reading a byte:

```
outportb (port+2, 0x22);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10));
if (counter == 10)
    timeout = 1;
data = inportb(port);
outportb (port+2, 0x20);
```

BPP + PS2

3.3 Combination: BPP + PS2

It has proven as meaningful to use a combination of BPP and PS2, since this solution functions in almost all cases:

Writing a byte:

```
outportb (port+2, 0x21);
outportb (port+2, 0x01);
outportb (port, data);
outportb (port+2, 0x03);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10));
if (counter == 10)
    timeout = 1;
outportb (port+2, 0x01);
outportb (port+2, 0x21);
outportb (port+2, 0x20);
```

Reading a byte:

```
outportb (port, 0xff);
outportb (port+2, 0x22);
counter = 0;
do {
    wait = inportb (port+1) & 0x80;
    counter += 1;
    udelay (1);
} while (wait && (counter < 10));
if (counter == 10)
    timeout = 1;
data = inportb(port);
outportb (port+2, 0x20);
```

EPP

3.4 EPP

With the introduction of the **EPP port** much became simpler: the entire control is handled now by the hardware! This became possible, as two further port addresses were added: port + 3 is the EPP address port and port + 4 is the EPP data port. These are equivalent, the one use the nASTRB line, the other the nDSTRB line. For applications with line scan camera circuit boards of the *EURECA GmbH* the latter is crucial:

- **Writing a byte:**

```
outportb (port+4, data);
```

- **Reading a byte:**

```
data = inportb(port+4);
```

The company XIRCOM brought a modification on the market, with which the driver chip must be set still explicitly into its desired status:

- **Writing a byte:**

```
outportb (port+2, 0x00);  
outportb (port+4, data);
```

- **Reading a byte:**

```
outportb (port+2, 0x20);  
data = inportb (port+4);
```

The latter possibility functions almost always, since interfaces, which control the port direction do not consider the PCD bit. Setting the direction must take place in general not with each access, but only when modifying the port direction.

Lately increased I/O chips came on the market, with those with each(!) access

- the direction must be set for reading a byte (see above)
- the timeout must be read and set back

```
buffer = inportb (port+1);  
outportb (port+1, buffer | 0x01);  
outportb (port+1, buffer & 0xfe);
```

As last possibility there is still the ECP+EPP port. This is after initialization:

```
outportb (port+0x402 ,0x00);  
outportb (port+0x402 ,0x80);
```

to used like the EPP port.

Interrupt

4 Interrupt

Many devices supply an interrupt with achieving a certain status. This is a signal, which interrupts the process on the PC, which is running at the moment, and calls a routine for the handling of the device status. After this routine is terminated, the PC returns to the interrupted process.

With the parallel interface generally 2 interrupt request lines (IRQs) are provided: IRQ-5 and IRQ-7. These are projected internally on interrupt vectors.

4.1 Allocation IRQ→Vector

These interrupt vectors are memory addresses, in which the physical address of the interrupt handling routine is situated. The 15 available IRQs are projected as follows on vectors.

| IRQ | Vector | Normal allocation | Controler | Register addr. | Bit No. | Bit value |
|-----|--------|-----------------------------|-----------|----------------|---------|-----------|
| 0 | 0x08 | Timer 18, 2Hz | 1 | 0x21 | 0 | 0x01 |
| 1 | 0x09 | Key pad | 1 | 0x21 | 1 | 0x02 |
| 2 | 0x0A | = IRQ-9 | 1 | 0x21 | 2 | 0x04 |
| 3 | 0x0B | COM-2, /dev/ttyS1 | 1 | 0x21 | 3 | 0x08 |
| 4 | 0x0C | COM-1, /dev/ttyS0 | 1 | 0x21 | 4 | 0x10 |
| 5 | 0x0D | LPT-2, /dev/lp2 | 1 | 0x21 | 5 | 0x20 |
| 6 | 0x0E | Floppy, /dev/fd0, /dev/fd1 | 1 | 0x21 | 6 | 0x40 |
| 7 | 0x0F | LPT-1, /dev/lp0, /dev/lp1 | 1 | 0x21 | 7 | 0x80 |
| 8 | 0x70 | real timer 1024Hz | 2 | 0xA1 | 0 | 0x01 |
| 9 | 0x71 | free | 2 | 0xA1 | 1 | 0x02 |
| 10 | 0x72 | free | 2 | 0xA1 | 2 | 0x04 |
| 11 | 0x73 | free | 2 | 0xA1 | 3 | 0x08 |
| 12 | 0x74 | free | 2 | 0xA1 | 4 | 0x10 |
| 13 | 0x75 | free | 2 | 0xA1 | 5 | 0x20 |
| 14 | 0x76 | IDE, /dev/hda, /dev/hdb | 2 | 0xA1 | 6 | 0x40 |
| 15 | 0x77 | 2. EIDE, /dev/hdc, /dev/hdd | 2 | 0xA1 | 7 | 0x80 |

4.2 Interrupt controller

The IRQs are collected in interrupt controllers. There each individual IRQ can be activated or deactivated (masked). This is done via the setting or deletion of a bit in the control register of the controller (see table). A set bit corresponds to an active IRQ. It is to be made sure, that already active IRQs of system components are not particularly deactivated, with exception of that one, that you want to use. Deactivating the IDE-IRQ leads e.g. to data losses and in extreme cases to the destruction of the complete hard disk contents!

After the handling of a IRQ by the interrupt controller, this stays in an inactive status, i.e. it cannot receive further IRQs. This ensures for the fact that a further interrupt is not released, while the first is still processed. In the handling routine therefore the interrupt controller must be released again (when possible right after the beginning, when the processor registers are saved). For the first controller this takes place via writing a 0x20 on the port 0x20, for the second controller by writing a 0x20 on the port 0xA0.

4.3 Setting the interrupt vectors

In order to set the interrupt vector, most high-level languages offer appropriate instruction. In Borland-C these read:

```
oldhandler = getvect (INTR);
setvect (INTR, newhandler );
```

The instruction **getvect** reads the vector indicated for the interrupt. This is needed, in order to activate the old vector again, after termination of the program.

The instruction **setvect** sets the vector to the new routine.

The routine, to which the vector points, should be explicitly defined as an interrupt routine. This is necessary, so that the processor registers become secured correctly.

4.4 IRQ of the parallel interface

The parallel interface uses generally the IRQ-5 or IRQ-7. In order to use the interrupt, this must be released by setting the Bit-4 (IRQEN) in the control port:

```
outportb (port+0x02, 0x10);
```

4.5 Program example

```

/*****
/*
/*      --- Declaration ---
/*
/*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define PORT 0x378          // Address of the EPP port
#define IRQ 7              // IRQ of the EPP port

int counter,
    nvorlauf = 50          // 50 port accesses up to the first pixel
    npixel = 2048          // 2048 pixel
    nnachlauf = 16         // 16 port accesses for the camera reset
int modus = 96,            // Port transfer rate 1000 kB/s
    integration = 15,      // Integration time 150ms
    io_out = 0,            // Outputs
    io_in;                // Inputs

char text[128];            // Array for storing the ASCII data
unsigned char result[2048]; // Array for storing the sensor information

void interrupt routine (void); // Declaration of the interrupt routine
void interrupt (* oldhandler)(void); // Pointer to the old vector
int oldmask;                // Old mask

```

```

/*****
/*
/*      ---  Program start  ---
/*
/*
/*****

void main (void)
{

    oldhandler = getvect (INTR | 0x08);           // save the old vector
    oldmask = inportb (0x21);                     // save the old mask
    setvect (INTR | 0x08, routine);               // setting the new vector
    outportb (0x21, oldmask & (0xff^(0x01 << INTR))); // setting the new mask

    outportb (PORT+2, 0x10);                      // Initialisation of the interface
                                                //   (activating the interrupt)

    routine ();                                   // "starting" the camera
    ...                                           //   because of the interrupts
    ...                                           //   the system keeps on running
    ...                                           //   with the adjusted integration time
    ...                                           //   automatically
}

void interrupt routine (void)                   // Declaration of the interrupt routine
{
    outportb (0x20, 0x20);                      // Resetting the interrupt controller

    outportb (PORT+4,odus);                      // Setting the transfer and camera mode
    outportb (PORT+4, integration);              // Setting the integration time
    outportb (PORT+4, io_out);                  // Transmitting the I/O signals OUT0 to OUT7
    io_in = inportb(PORT+4);                    // Reading the I/O signals IN0 to IN7
    for (counter = 0; counter < nvorlauf - 4; ++counter) // Reading the camera designation and version number
        text[counter] = inportb (port+4);
    for (counter = 0; counter < npixel; ++counter) // Reading the sensor information
        result[counter] = inportb (port+4);
    for (counter = 0; counter < nnachlauf; ++counter) // Reset of the line scan camera
        inportb (port+4);
}

```

5 Operating systems

The programming under DOS, Windows 95 and Windows 98 is normally no problem, since the accesses to the port addresses are not protected.

When using Windows NT the accesses must run mandatory through the Kernel! For this there are some shareware programmes which can be loaded as SYS and DLL drivers.

With UNIX systems the port addresses which can be used must be released. Up to the port address 0x3FF this is made e.g. by the **ioperm** instruction. When using ECP/EPP able interfaces they must be accessed over the instruction **iocntl** in the address PORT+0x402 . After this access one should release the port area over **ioperm**.

Quick and Dirty: Hardware direction protocol **EPP 1.9**

6 Quick and Dirty

6.1 EPP 1.9 hardware direction protocol

The control of an EPP able interface, which supports EPP 1,9 in the hardware direction protocol reduces the programming expenditure to a few instruction:

```

/*****
/*
/*      --- Declaration ---
/*
/*
/*****

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define PORT 0x378                                // Port adress of the EPP port

int counter,                                     // Variable
    nvorlauf = 50                               // 50 port accesses up to the first pixel
    npixel = 2048                                // 2048 pixel
    nnachlauf = 16                               // 16 port accesses for the camera reset
int modus = 96,                                  // Transfer rate 1000 kB/s
    integration = 15,                             // Integration time 150ms
    io_out = 0,                                   // Outputs
    io_in;                                        // Inputs

char text[128];                                  // Array for storing the ASCII data
unsigned char result[2048];                      // Array for storing the sensor information

    outportb (PORT+2, 0x04);                      // Initialisation of the interface

/*****
/*
/*      --- Program start ---
/*
/*
/*****

    outportb (PORT+4, modus);                      // Transmitting the transfer and camera mode
    outportb (PORT+4, integration);                // Transmitting the integration time
    outportb (PORT+4, io_out);                     // Setting the I/O signals OUT0 to OUT7
    io_in = inportb(PORT+4);                       // Reading the I/O signals IN0 to IN7
    for (counter = 0; counter < nvorlauf - 4; ++counter) // Reading the camera designation and version number
        text[counter] = inportb (port+4);
    for (counter = 0; counter < npixel; ++counter)   // Reading the sensor information
        result[counter] = inportb (port+4);
    for (counter = 0; counter < nnachlauf; ++counter) // Reset of the line scan camera
        inportb (port+4);

```

Lately increased I/O chips came on the market, with those with each(!) access

- the direction must be set again for reading a byte
- the timeout must be read and reseted

Quick and Dirty:

Software direction protocol **EPP 1.9**

6.2 EPP 1.9 software direction protocol

to control an EPP able interface, which provides EPP 1.9 with Software direction protocol, some instructions have to be added:

```

/*****
/*
/*      --- Declaration ---
/*
/*
/*****

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define PORT 0x378                                // Port address of the EPP interface

int counter,                                     // Variable
    nvorlauf = 50                               // 50 port accesses up to the first pixel
    npixel = 2048                                // 2048 pixel
    nnachlauf = 16                              // 16 port accesses for the camera reset
int modus = 96,                                  // Transfer rate 1000 kB/s
    integration = 15,                            // Integration time 150ms
    io_out = 0,                                  // Outputs
    io_in;                                       // Inputs

char text[128];                                  // Array for storing the ASCII data
unsigned char result[2048];                      // Array for storing the sensor information

/*****
/*
/*      --- Program start ---
/*
/*
/*****

    outportb (PORT+2, 0x04);                      // Setting the interface to output

    outportb (PORT+4, modus);                     // Setting the transfer and camera mode
    outportb (PORT+4, integration);               // Setting the integration time
    outportb (PORT+4, io_out);                   // Setting the I/O signals OUT0 to OUT7

    outportb (PORT+2, 0x24);                      // Setting the interface to input

    io_in = inportb(PORT+4);                      // Reading the I/O signals IN0 to IN7
    for (counter = 0; counter < nvorlauf - 4; ++counter) // Reading the camera designation and version number
        text[counter] = inportb (port+4);
    for (counter = 0; counter < npixel; ++counter) // Reading the sensor information
        result[counter] = inportb (port+4);
    for (counter = 0; counter < nnachlauf; ++counter) // Reset of the line scan camera
        inportb (port+4);

```

Lately increased I/O chips came on the market, with those with each(!) access

- the direction must be set again for reading a byte
- the timeout must be read and reseted

Quick and Dirty:

EPP 1.7

6.3 EPP 1.7

With the control of an EPP able interface, which supports EPP 1.7, the quick and dirty programming is identical to the EPP 1.9 with the software direction protocol (difference: EPP 1.9 supports the timeout flag in each case. This is not guaranteed with EPP 1,7 !):

```

/*****
/*
/*      --- Declaration ---
/*
/*
/*****

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define PORT 0x378                                // Port address of the EPP port

int counter,                                     // Variable
    nvorlauf = 50                               // 50 port accesses up to the first pixel
    npixel = 2048                                // 2048 pixel
    nnachlauf = 16                              // 16 port accesses for the camera reset
int modus = 96,                                 // Transfer rate 1000 kB/s
    integration = 15,                           // Integrations time 150ms
    io_out = 0,                                 // Outputs
    io_in;                                       // Inputs

char text[128];                                 // Array for storing the ASCII data
unsigned char result[2048];                    // Array for storing the sensor data

/*****
/*
/*      --- Program start ---
/*
/*
/*****

    outportb (PORT+2, 0x04);                     // Setting the interface to output

    outportb (PORT+4, modus);                    // Setting the transfer and camera mode
    outportb (PORT+4, integration);              // Setting the integration time
    outportb (PORT+4, io_out);                  // Setting the I/O signals OUT0 to OUT7

    outportb (PORT+2, 0x24);                     // Switching the interface to input

    io_in = inportb(PORT+4);                     // Reading the I/O signals IN0 to IN7
    for (counter = 0; counter < nvorlauf - 4; ++counter) // Reading the camera designation and version number
        text[counter] = inportb (port+4);
    for (counter = 0; counter < npixel; ++counter) // Reading the sensor information
        result[counter] = inportb (port+4);
    for (counter = 0; counter < nnachlauf; ++counter) // Reset of the line scan camera
        inportb (port+4);

```

Lately increased I/O chips came on the market, with those with each(!) access

- the direction must be set again for reading a byte
- the timeout must be read and reseted

Quick and Dirty:

INTR-Leitung

6.4 Polling the INTR line

The INTR line can be used on two different ways:

- Interrupt operation
- Polling operation

In the context of quick and dirty programming the polling operation is used, because of the substantially smaller programming expenditure. Also the danger of system crashes caused by wrong interrupt vectors is avoided.

With the polling operation the INTR line is queried by reading the status register. If the integration time is over, then the bit-6 is set and the camera can be read out:

```
if (inportb (port+1) & 0x40)                // INTR line active:
{
    outportb (PORT+2, 0x04);                // Integration completed
    outportb (PORT+4, modus);               // Setting the interface to output
    outportb (PORT+4, integration);         // Setting the transfer and camera mode
    outportb (PORT+4, io_out);              // Setting the integration time
    outportb (PORT+4, io_out);              // Setting the I/O signals OUT0 to OUT7
    outportb (PORT+2, 0x24);                // Switching the interface to input
    io_in = inportb(PORT+4);                // Reading the I/O signals IN0 to IN7
    for (counter = 0; counter < nvorlauf - 4; ++counter) // Reading the camera designation and version number
        text[counter] = inportb (port+4);
    for (counter = 0; counter < npixel; ++counter)      // Reading the sensor information
        result[counter] = inportb (port+4);
    for (counter = 0; counter < nnachlauf; ++counter)  // Reset of the line scan camera
        inportb (port+4);
}
```

With the Polling operation it is to take care, that the query of the status bit is made regular and is repeated in very short intervals, in order prevent a too strong variation of the integration time (setting the INTR line does not interrupt the integration. This is only terminated with the read out of the camera!).