Elementare E-A-Zugriffsfunktionen in C
– Kurzübersicht –

1. Physische Ein- und Ausgabe

Physische E-A-Adressen verwenden!

Ausgabe Richtungssteuerregister: **DIROUT**

void dirout(int adrs, int databyte)

- adrs: logische Portadresse,
- databyte: Portbelegung.

Ausgabe Datenregister: **DATOUT**

void datout (int adrs, int databyte)

- adrs: logische Portadresse,
- databyte: Portbelegung.

2. Logische Ein- und Ausgabe

Logische Portadressen verwenden!

Byteausgabe: **OUT**

void out (int adrs, int data)

- adrs: logische Portadresse,
- data: Portbelegung.

Byteeingabe: **IN**

int in (int adrs)

- adrs: logische Portadresse,
- Rückgabe: Portbelegung.

Bitausgabe: **BITOUT**

void bitout (int adrs, int bitpos, int data)

- adrs: logische Portadresse,
- bitpos: Bitposition im Byte (7...0).
- data: 0 ergibt Bitwert Null, ungleich 0 ergibt Bitwert 1

Biteingabe: **BITIN**

int bitin (int adrs, int bitpos)

- adrs: logische Portadresse,
- bitpos: Bitposition im Byte (7...0).
- Rückgabe: 0 oder 1.

E-A-Adressierung im Überblick

1. Physische Adressen des Portapters im PC

– Bei Wechsel auf eine andere IDE/ATA-Schnittstelle nur hier ändern –

```
#define portadrs    0x176    //  Geräte- und Kopfauswahlregister (DH; LBA 27...24)
#define dir_reg     0x175    //  Zylinder-Nr. hoch (LBA 23...16)
#define dat_reg     0x174    //  Zylinder-Nr. niedrig (LBA 15...8)
```

Aufbau des Portadapters:

E-A-Adresse	ATA-Register	Inhalt				
0x174	Zylinder-Nr. niedrig	Datenregister				
0x175	Zylinder-Nr. hoch	Richtungssteuerregister				
0x176	Geräte- und Kopfauswahl	0	0	0	1	Portadresse (0x1...0xF)

Rücklesen vom Port über E-A-Adressen beider Zylinder-Nr.-Register. Rücklesen vom Geräte- und Kopfauswahlregister ergibt stets 0xFF.

2. Physische E-A-Adressen

```
#define   ioport_a    0x1c
#define   ioport_b    0x1d
#define   ioport_c    0x1e
#define   ioport_d    0x1f
```

3. Logische Portadressen

```
#define   dir_a       0
#define   dat_a       1
#define   port_a      2
#define   dir_b       3
#define   dat_b       4
#define   port_b      5
#define   dir_c       6
#define   dat_c       7
#define   port_c      8
#define   dir_d       9
#define   dat_d       10
#define   port_d      11
```

Achtung:

- Datenausgabe auf das jeweilige **dat**-Register.
- Datenausgabe auf die Anschlüsse (**port)** ist wirkungslos.
- Ausgabe zwecks Richtungssteuerung auf das jeweilige **dir**-Register.
- Dateneingabe von den Anschlüssen, d. h. vom jeweiligen **port**. (Bei Eingabe von **dat** wird nur der alte Inhalt des **dat**-Registers zurückgelesen.)

Quelltexte der E-A-Routinen

Physische E-A-Zugriffsfunktionen für Borland-Compiler.

// * 1. Physische Ein- und Ausgabe (physische E-A-Adressen verwenden!)*****

```
void dirout(int adrs, int databyte)        // Ausgabe Richtungssteuerregister
{
     outportb (portadrs, adrs);                        // Portadresse laden
     outportb (dir_reg,databyte);                      // Richtungssteuerregister laden
}


void datout (int adrs, int databyte)       // Ausgabe Datenregister
{
     outportb (portadrs, adrs);                        // Portadresse laden
     outportb (dat_reg, databyte);                     // Datenregister laden
}


int datin (int adrs)                       // Porteingabe
{
     outportb (portadrs, adrs);                        // Portadresse laden
     return  inportb (dat_reg);                        // Port lesen
}
```

// * 2. Logische Ein- und Ausgabe (logische Portadressen verwenden!) *****

```
void out (int adrs, int data)              // Byteausgabe
{

switch ((portctl [adrs] & 0xff00))                     // Registertyp
{
     case 0x0100:                                      // Richtungssteuerregister
     portregs [adrs] = data;                           // Kopie speichern
     outportb (portadrs, (portctl [adrs] & 0xff));     // Register in Adapter adressieren
     outportb (dir_reg,data);                          // Ausgabe in Register
     break;

     case  0x0200:                                      // Datenregister
     portregs [adrs] = data;                           // Kopie speichern
     outportb (portadrs, (portctl [adrs] & 0xff));     // Register in Adapter adressieren
     outportb (dat_reg,data);                          // Ausgabe in Register
     break;
}
return;
}
```

int in (int adrs) // Byteeingabe

```
{

if ((portctl [adrs] & 0xff00) == 0x0300)          // Registertyp, Nur von Port lesen
      {
      outportb (portadrs, (portctl [adrs] & 0xff));  // Register in Adapter adressieren
      portregs [adrs] =  inportb (dat_reg);          // Eingabe in die Registerkopie
      }

return portregs [adrs];                           // Rückgabe der Registerkopie
}
```

void bitout (int adrs, int bitpos, int data) // Bitausgabe

```
{

switch ((portctl [adrs] & 0xff00))                // Registertyp
  {

      case 0x0100:                                // Richtungsregister
      if (data == 0) portregs [adrs] = (portregs [adrs] & ~bitmasks [bitpos] & 0xff); // Bit = 0
          else   portregs [adrs] = (portregs [adrs] | bitmasks [bitpos]);             // Bit = 1
      outportb (portadrs, (portctl [adrs] & 0xff));  // Register in Adapter adressieren
      outportb (dir_reg, portregs [adrs]);           // Ausgabe in Register
      break;

      case  0x0200:                               // Datenregister
      if (data == 0) portregs [adrs] = (portregs [adrs] & ~bitmasks [bitpos] & 0xff); // Bit = 0
          else portregs [adrs] = (portregs [adrs] | bitmasks [bitpos]);               // Bit = 1
      outportb (portadrs, (portctl [adrs] & 0xff));  // Register in Adapter adressieren
      outportb (dat_reg, portregs [adrs]);           // Ausgabe auf Register
      break;
    }

return;
}
```

int bitin (int adrs, int bitpos) // Biteingabe

```
{

if ((portctl [adrs] & 0xff00) == 0x300)           // Registertyp, Nur von Port lesen
      {
      outportb (portadrs, (portctl [adrs] & 0xff));
      portregs [adrs] =  inportb (dat_reg);          // Eingabe in die Registerkopie
      }

if ((portregs [adrs] & bitmasks [bitpos]) == 0) return 0;   // Bit in Registerkopie abfragen
      else return 1;
}
```

Initialisierung im Hauptprogramm

Datenstrukturen der Ein- und Ausgabe:

- portregs = Kopien der Portregister (Lesen und Schreiben). Es gibt 4 Ports mit je 3 Adressen:
  - dir = Richtungssteuerung,
  - dat = Daten (ausgabeseitig),
  - port = Anschlüsse (Eingabe von außen).
- portctl = Steuer- und Adreßangaben für alle Portregister (Festwerte).
- bitmasks = Verknüpfungsmasken für alle Bitpositionen im Byte (Festwerte).

// portregs

```
// Kopien der Portregister werden gelöscht:

    for (n=0; n<=11; n++)
    portregs [n] = 0;
```

// portctl

```
// Array zur Steuerung der Portzugriffe
// niederes Byte: Portadresse im IDE-Adapter
// höheres Byte: kennzeichnet die Art des Registers: 1 = Richtung, 2 = Daten, 3 = Port

portctl  [0]  = 0x011c;              // Port A
portctl  [1]  = 0x021c;
portctl  [2]  = 0x031c;
portctl  [3]  = 0x011d;              // Port B
portctl  [4]  = 0x021d;
portctl  [5]  = 0x031d;
portctl  [6]  = 0x011e;              // Port C
portctl  [7]  = 0x021e;
portctl  [8]  = 0x031e;
portctl  [9]  // Portadresse laden= 0x011f;          // Port D
portctl [10] = 0x021f;
portctl [11] = 0x031f;
```

// bitmasks

```
bitmasks[0] = 1;        // 0000  0001
bitmasks[1] = 2;        // 0000  0010
bitmasks[2] = 4;        // 0000  0100
bitmasks[3] = 8;        // 0000  1000
bitmasks[4] = 16;       // 0001  0000
bitmasks[5] = 32;       // 0010  0000
bitmasks[6] = 64;       // 0100  0000
bitmasks[7] = 128;      // 1000  0000
```