Beispiel zu Pointern

```
int     *a, b;          *a → Pointer
b = 17;
a = &b;                 & → Adressoperator
*a = 9;                 * → Inhaltsoperator
a++;                    Folgeadresse aufrufen
*a = 17;
```

Alternative Darstellung:

Adresse	**Wert**
0x2000	0x11 0x09
0x2004	?? 0x11

0x2000 → 0x2004
Typ = Integer → Inkrement um 4

Bibliotheken

#include <stdio.h> → Inhalt z.B. Funktionen scanf, printf

Speicheraufteilung des LPC 2148

Prozessorfamilie LPC 2000 (ganze Reihe von Controllern mit nur wenigen Unterschieden)

LPC 2148 → Name: $\underbrace{ARM7}_{Kernarchitektur}\ \underbrace{T}_{Thumb-Befehle}\ \underbrace{D}_{Debugport}\ \underbrace{M}_{64-Bit\ Multiplier}\ \underbrace{I}_{ICE-Modul}\ -\ \underbrace{S}_{Fertigungstechnik}$

ICE-Modul → In Circuit Emulator

Speicheraufteilung → s. Vorlesungsfolien

C-Programmierung von Mikrokontrollern

Jedes C-Programm enthält auf jeden Fall eine Assembler-Datei (startup.s). Diese wird assembliert, die C-Dateien werden von einem Compiler kompiliert.

Ablauf beim Start einer C-Applikation

- Systeminitialisierung (nach Reset):
  - Initialisieren der Interrupt-Vektoren
  - Speicherreservierung
  - Stack initialisieren
- Anschließend cinit
  - Initialisieren statischer Variablen
- Dann: main Funktion des C-Programms

Basistypen

➔ S. Folie 53 der Vorlesungsunterlagen

Attribut const

In Assembler: CODE, READONLY als Pendant

Bsp.:

```
const   unsigned        InputCodeTab[] = {1,12,…}
const   unsigned        OutputCodeTab[] = {36,48,…}
```

Attribut volatile

Bsp.:

```
Volatile        unsigned int    geraetestatus
```

Zugriff auf Register von Peripherieeinheiten

Bsp.:

```
#define         IOCLR   (*((volatile  unsigned  long*)    0xE002801C;
                IOCLR = 0x7;    //Variable kann entsprechend verändert werden
```

Einbinden von Funktionen und Variablen für den LPC 2148 mit der entsprechenden Biblothek:

```
#include        <LPC214x.h>
```

Digitale Eingabe- und Ausgabe Portpins

- GPIO = General Purpose Input / Output (Standard-Digitalport)
- Portpins können verschiedene Funktionen (Digital I/O, PVM, Timer, …) einnehmen

Portpin-Konfiguration

- 45 Portpins (von 64 Pins insgesamt)
- PORT0.0 bis PORT0.31 (reserviert sind P0.24, P0.26 und P0.27)
- PORT1.16 bis PORT1.31 (JTAG-Schnittstelle an P1.26 bis P1.31)
- PINSEL0 → Pin-Select-Register → je zwei Bit entscheiden über Portbelegung
- Beispiele:
  a) Belegung PORT0.0:
     00 → GPIO Port 0.0
     01 → TxD (UMT0)
     10 → PVM1
     11 → Reserved

  b) PINSEL0 = 0xC0;
     Binäre Umformung zur Analyse: …$\underbrace{11}_{7:6}\underbrace{00}_{5:4}\underbrace{00}_{3:2}\underbrace{00}_{1:0}$
     Portpin P0.3 ist als EINT1 (ext. Interrupt) konfiguriert

  c) PINSEL0 = 0x80000008 (1000 … 0000 1000)
     Portpin 0.1 → PWM3
     Portpin 0.15 → EINT2
     Restliche Portpins 0.x → GPIO 0.x

- PINSEL1 konfiguriert die Portpins P0.16 – P0.31 (Ausnahmen beachten!)
- PINSEL2 konfiguriert die Portpins P1.16 – P1.31
- Vorlesungsinhalte am LPC2148:
    - Digitale Ein-und Ausgabe (GPIO)
    - Externe Interrupts (EINT)
    - Timer, zeitgesteuerte Abläufe (TIMER)
    - Serielle Schnittstelle (SPI)

<u>Kontroll-Register für Ein- und Ausgabeportpins</u>

$\underbrace{IODIR0}_{\text{für Port 0}}, \underbrace{IODIR1}_{\text{für Port 1}}$

Bsp.:
Pin P0.7 als Ausgang konfigurieren → IODIR0 = 0x80

IOPIN0, IOPIN1 → Auslesen des Status der entsprechenden Portpins
Bsp.:
schalter = (IOPIN0 & 0x8);

IOSET0, IOSET1 → setzen Ausgangspegel (3.3 V, digitale 1)
Bsp.:
```
IODIR0 = 0x80;          //P0.7 als Ausgang setzen
IOSET0 = 0x80;          //P0.7 auf digitale 1 setzen
```

IOCLR0, IOCLR1 → setzen Ausgangspegel (0 V, digitale 0)
Bsp.:
```
IOCLR = 0x80;           //P0.7 auf digitale 0 setzen
```

Übungsbeispiel:
```
IODIR1 = IODIR1 | 0x10000;    //Logisches ODER, um nur einzelne Stelle in IODIR1 zu verändern
IODIR1 = 0x10000;             //Gesamte Konfiguration in IODIR1 mit 0x10000 überschreiben
```

<u>Erweiterung des Übungsbeispiels (Folie 57 d. Vorlesungsunterlagen)</u>
LED 1, LED 3 und LED 8 einschalten auf Knopfdruck
LEDs liegen an den Portpins P1.16, P1.18 und P1.24

Änderungen im Programmablauf:
```
IODIR1 = IODIR1 | 0x1050000;
IOSET1 = 0x1050000;
IOCLR1 = 0x1050000;
```

Externe Interrupts durch Portpins

EXTMODE → Umschalten von Flanke / Zustand
Bsp. (Flanke einstellen):

```
EXTMODE = 0x2;      //EINT1
EXTMODE = 0x1;      //EINT0
EXTMODE = 0x4;      //EINT2
EXTMODE = 0x8;      //EINT3

EXTMODE = 0x3;      //EINT0 & EINT1 → Flanke
```

EXTPOLAR → Interrupt schalten
Bsp.:

```
EXTPOLAR = 0x2;     //HIGH-Pegel an EINT1 löst Interrupt aus
EXTMODE = 0x4;      //EINT2 flankenabhängig machen
EXTPOLAR = 0x4;     //steigende Flanke löst Interrupt aus
```

EXTINT
Bsp.:

```
EXTINT = 0x8;       //EINT3 würde quittiert
```

Dies muss am Ende der ISR (Interrupt Service Routine) geschehen.

Zusammenfassung

Digitale Ein- und Ausgabe:

- IODIR0, IODIR1 → Eingang oder Ausgang (je 32 Pins)
- IOPIN0, IOPIN1 → Spannungspegel auslesen → Status
- IOSET0, IOSET1 → Ausgänge aktiv auf HIGH setzen (U = 3.3 V)
- IOCLR0, IOCLR1 → Ausgänge aktiv auf LOW setzen (U = 0 V)

Externe Interrupts (32 Bit Register, erste 2 Bit verwendet):

- EXTMODE → entscheidet, ob Pegel oder Flanke auslöst
- EXTPOLAR → entscheidet, ob
    - HIGH-Pegel oder LOW-Pegel Interrupt auslöst
    - Steigende Flanke oder fallende Flanke
- EXTINT → Prüfen, ob Voraussetzung vorliegt und wird für das quittieren der ISR genutzt