Beispiel ATmega32: Port-Register

- Pro Port x sind drei Register definiert (Beispiel für $x = D$)
  - `DDRx` **Data Direction Register:** Legt für jeden Pin i fest, ob er als Eingang (Bit i=0) oder als Ausgang (Bit i=1) verwendet wird.

7	6	5	4	3	2	1	0
DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

  - `PORTx` **Data Register:** Ist Pin i als Ausgang konfiguriert, so legt Bit i den Pegel fest (0=GND sink, 1=Vcc source). Ist Pin i als Eingang konfiguriert, so aktiviert Bit i den internen Pull-Up-Widerstand (1=aktiv).

7	6	5	4	3	2	1	0
PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

  - `PINx` **Input Register:** Bit i repräsentiert den Pegel an Pin i (1=*high*, 0=*low*), unabhängig von der Konfiguration als Ein-/Ausgang.

7	6	5	4	3	2	1	0
PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
R	R	R	R	R	R	R	R

Verwendungsbeispiele: ↪ 3–5 und ↪ 3–8 [1, S. 66]

Peripheriegeräte – Register (Forts.)

- Memory-mapped Register ermöglichen einen komfortablen Zugriff
  - Register ↦ Speicher ↦ Variable
  - Alle C-Operatoren stehen direkt zur Verfügung (z. B. `PORTD++`)
- Syntaktisch wird der Zugriff oft durch Makros erleichtert:

```
#define PORTD ( * (volatile uint8_t*)( 0x12 ) )
```

Adresse: int

Adresse: volatile uint8_t* (Cast ↪ 7–17)

Wert: volatile uint8_t (Dereferenzierung ↪ 13–4)

`PORTD` ist damit (syntaktisch) äquivalent zu einer `volatile uint8_t`-Variablen, die an Adresse `0x12` liegt

- Beispiel

```
#define PORTD (*(volatile uint8_t*)(0x12))

PORTD |= (1<<7);          // set D.7
uint8_t *pReg = &PORTD;   // get pointer to PORTD
*pReg &= ~(1<<7);         // use pointer to clear D.7
```

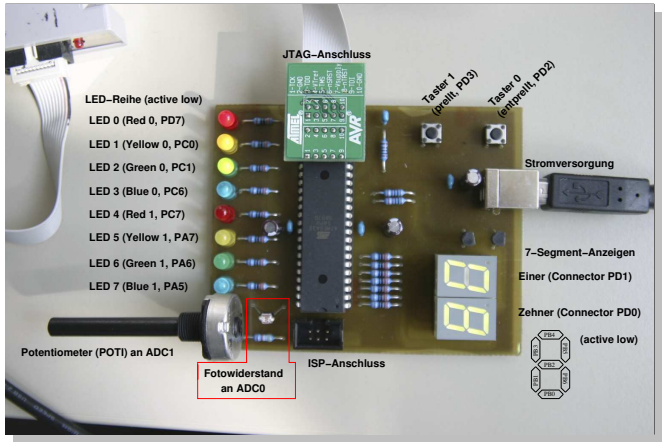

Das erste C-Programm für einen μ-Controller

- „Hello World“ für AVR ATmega (vgl. ↪ 3–1)

```
1  #include <avr/io.h>
2
3  void main() {
4    // initialize hardware: LED on port D pin 7, active low
5    DDRD  |= (1<<7);  // PD7 is used as output
6    PORTD |= (1<<7);  // PD7: high --> LED is off
7
8    // greet user
9    PORTD &= ~(1<<7); // PD7: low --> LED is on
10
11   // wait forever
12   while(1){
13   }
14 }
```


Übungsplattform: Das SPiCboard

- ATmega32-μC
- JTAG-Anschluss
- 8 LEDs
- 2 7-Seg-Elemente
- 2 Taster
- 1 Potentiometer
- 1 Fotosensor



- Ausleihe zur Übungsbearbeitung möglich
- Oder noch besser ↪ selber Löten

Module in C – Header (Forts.) [≠Java]

Modulschnittstelle: `foo.h`

```
// foo.h
#ifndef _FOO_H
#define _FOO_H

// declarations
extern uint16_t a;
void f(void);

#endif // _FOO_H
```

Modulimplementierung `foo.c`

```
// foo.c
#include <foo.h>


// definitions
uint16_t a;
void f(void){
  ...
}
```

Modulverwendung `bar.c`
(vergleiche ↪ 12–11)

```
// bar.c
extern uint16_t a;
void f(void);
#include <foo.h>

void main() {
  a = 0x4711;
  f();
}
```