

## U4 3. Übungsaufgabe

- Grundlegendes zur Übung mit dem AVR- $\mu$ C
- Register
- I/O-Ports
- AVR-Umgebung
- Peripherie

## U4-2 Register beim AVR- $\mu$ C

### 1 Überblick

- Beim AVR- $\mu$ C sind die Register
  - ◆ in den Speicher eingebettet
  - ◆ am Anfang des Adressbereichs angeordnet
- Adressen sind der Dokumentation zu entnehmen
- vollständige Dokumentation für "unseren" Mikrokontroller ATmega8:  
`/proj/i4gdi/tools/doc/`
- Für die Aufgaben benötigte Register sind auf den Folien erwähnt
- Die Bibliothek (`avr-libc`), die wir verwenden, definiert bereits sinnvolle Makros für alle Register des AVR  $\mu$ C (`#include <avr/io.h>`)

## U4-1 Grundlegendes zur Übung mit dem AVR- $\mu$ C

- Die Übungsaufgaben werden wir mit Hilfe eines Simulators testen und entwickeln
  - ▶ hierzu verwenden wir die Entwicklungsumgebung AVR Studio von Atmel
- Wer möchte, kann sein Programm auch auf echter Hardware testen
  - ▶ es wird ein **ATmega8**-basiertes Board eingesetzt
  - ▶ wurde für uns von Studenten der AGEE gebaut (**Danke!**)
- AVR Studio läuft nur unter Windows
  - ▶ Von den Linux Rechnern aus wird mit `rdesktop` (Remote Desktop) auf dem Windows Terminalserver **faiu07** gearbeitet.
  - ▶ Hierzu muss jeder Student zunächst mit dem Kommando  
`/local/ciptools/bin/setsambapw`  
ein Windows-Passwort setzen.
  - ▶ **Achtung:** Passwörter werden erst zur jeweils 12. und 42. Minute jeder Stunde übernommen!

## 2 Makros für Register-Zugriffe

- Makros mit aussagekräftigen Namen können den Umgang mit Registern deutlich vereinfachen
- Beispiel:
  - ◆ Makro für Register an Adresse 0x5:

```
#define REG1 (*(volatile unsigned char *)0x5)
```

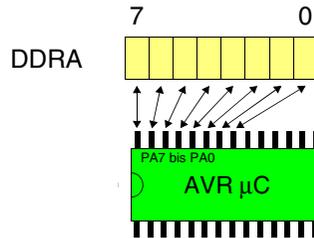
- ◆ Verwenden dieses Registers:

```
REG1 = 0;           /* schreibender Zugriff */
...
if (REG1 == 0x04)  /* lesender Zugriff */
    REG1 &= ~4;    /* lesender und schreibender Zugriff */
```

- Das `volatile`-Schlüsselwort wird später erläutert, im Moment ist es bei sämtlichen Zugriffen auf Hardwareregister zu verwenden.

## 1 Überblick

- Jeder I/O-Port des AVR- $\mu$ C wird durch drei 8-bit Register gesteuert:
  - ◆ Datenrichtungsregister (DDR<sub>x</sub> = data direction register)
  - ◆ Datenregister (PORT<sub>x</sub>)
  - ◆ Port Eingabe Register (PIN<sub>x</sub> = port input register, nur-lesbar)
- Jedem Anschluss-Pin ist ein Bit in jedem der 3 Register zugeordnet
  - Beispiel: DDR von Port A:



## 3 Beispiel: Aktivieren eines Ports

- Pin 3 von Port B als Ausgang konfigurieren und auf V<sub>CC</sub> schalten:

```
DDRB |= 0x08; /* Pin 3 von Port B als Ausgang nutzen... */
PORTB |= 0x08; /* ...und auf 1 (=high) setzen */
```

- Pin 0 von Port D als Eingang nutzen, pull-up-Widerstand aktivieren und prüfen ob ein low-Pegel anliegt:

```
DDRD &= ~0x01; /* Pin 0 von Port D als Eingang nutzen... */
PORTD |= 0x01; /*...und den pull-up-Widerstand aktivieren*/

if ( (PIND & 0x01) == 0) { /* den Zustand auslesen */
    /* ein low Pegel liegt an, der Taster ist gedrückt */
}
```

# 2 I/O-Port-Register

- **PIN<sub>x</sub>**: Bit *i* gibt den aktuellen Wert des Pin *i* von Port *x* an (nur lesbar).
- **DDR<sub>x</sub>**: hier konfiguriert man ob man einen Pin von Port *x* als Ein- oder Ausgang verwenden möchte
  - Bit *i* = 1 → Pin *i* als **Ausgang** verwenden
  - Bit *i* = 0 → Pin *i* als **Eingang** verwenden
- **PORT<sub>x</sub>**: Auswirkung abhängig von DDR<sub>x</sub>:
  - ◆ ist Pin *i* als **Ausgang** konfiguriert, so steuert Bit *i* im PORT<sub>x</sub> Register ob am Pin *i* ein high- oder ein low-Pegel erzeugt werden soll.
    - Bit *i* = 1 → high-Pegel an Pin *i*
    - Bit *i* = 0 → low-Pegel an Pin *i*
  - ◆ ist Pin *i* als **Eingang** konfiguriert, so kann man einen internen pull-up-Widerstand aktivieren
    - Bit *i* = 1 → pull-up-Widerstand an Pin *i* (Pegel wird auf high gezogen)
    - Bit *i* = 0 → Pin *i* als tri-state konfiguriert

# U4-4 AVR-Umgebung

## 1 Compiler

- Um auf einem PC Programme für den AVR-Mikrocontroller zu erstellen, wird ein **Cross-Compiler** benötigt
  - ◆ Ein Cross-Compiler ist ein Compiler, der Code für Architektur generiert, die von der Architektur des Rechners, auf dem der Compiler ausgeführt wird, verschieden ist.
  - ◆ Hier: Compiler läuft auf Intel x86 und generiert Code für AVR.
  - ◆ AVR Studio + WinAVR (Windows)
  - ◆ GNU gcc (Linux)
- Wir verwenden AVR Studio

## 2 Windows-Terminalserver

- Verbinden mit dem Windows-Terminalserver **fau107**:  
`rdesktop -d ICIP -f fau107`
- Startet eine Windows-Terminalsitzung im Vollbildmodus
  - ◆ Wechsel zwischen Fenster- und Vollbildmodus mit **CTRL+ALT+ENTER**
- Windows-Umgebung
  - ◆ UNIX-Homeverzeichnis als Laufwerk H:
  - ◆ SPiC-Projektverzeichnis (`/proj/i4gdi/siVVNNNM`) als Laufwerk P:
  - ◆ SPiC-Vorgabenverzeichnis (`/proj/i4gdi/pub`) als Laufwerk Q:
- Struktur des Projektverzeichnisses wie gehabt
- Die Abgabe erfolgt weiterhin unter UNIX!

## 4 AVR Studio-Projekteinstellungen

- Setzen der Projekteinstellungen (*Project - Configuration Options*)
  - ◆ Frequency: `1000000` Hz
  - ◆ Optimization: `-O0`
- Compileroptionen im Bereich *Custom Options*
  - ◆ `-std=gnu99` entfernen
  - ◆ `-ansi` hinzufügen
  - ◆ `-pedantic` hinzufügen
  - ◆ `-ffreestanding` hinzufügen
- Projekteinstellungen speichern

## 3 AVR Studio

- Entwicklungsumgebung von Atmel
- Simulator und Debugger für alle AVR-Mikrocontroller
- Start über das Startmenü  
*Start - Alle Programme - Atmel AVR Tools - AVR Studio 4*
- Anlegen eines neuen Projekts
  - ◆ Projekttyp: AVR GCC
  - ◆ Projektname: `aufgabe3`
  - ◆ *Create Initial File* und *Create Folder* aktivieren
  - ◆ Initial File: `laufflicht.c`
  - ◆ Location: `P:\`
- Auswahl der Plattform *AVR Simulator* mit dem Gerät *ATMega8*

## 5 Erstellen einer main()-Funktion

- Auf dem Mikrocontroller ist die *main()*-Funktion vom Typ `void main(void);`
  - Sollte niemals zurückkehren (wohin?)
  - Rückgabebetyp daher nicht sinnvoll
  - Freistehende Umgebung (`-ffreestanding`)
- Beispiel: Grüne LED einschalten

```
#include <avr/io.h>

void main(void) {
    DDRB = 0xff; /* alle Pins von Port B als Ausgang */
    PORTB = 0x01; /* ...und Pin 0 auf high setzen */

    for (;;) /* Endlosschleife */
}

```

- Kompilieren des Programmes mit F7 oder *Build - Build*

## 6 Starten des Simulators/Debuggers

- Konfiguration des Simulators  
*Debug - AVR Simulator Options - Frequency 1 MHz*
- Das Programm wird zunächst beim Betreten von `main()` angehalten
  - ◆ Normal laufen lassen mit *F5* oder *Debug - Run*
  - ◆ Schrittweise abarbeiten mit
    - *F10* (step over): Funktionsaufrufe werden in einem Schritt bearbeitet
    - *F11* (step into): Bei Funktionsaufrufen wird die Funktion betreten
- Die I/O-Ansicht (rechts) gibt Einblick in die Zustände der I/O-Register
- Breakpoints erlauben, das Programm an einer bestimmten Stelle zu stoppen
  - ◆ Codezeile anklicken, dann *F9* oder *Debug - Toggle Breakpoint*
  - ◆ Programm laufen lassen (*F5*), stoppt wenn ein Breakpoint erreicht wird

### 1 Taster-Details

- Den Pin, an dem der Taster angeschlossen ist, als Eingang konfigurieren
  - Entsprechendes Bit im DDRx-Register löschen
- Für den Taster muss der pull-up-Widerstand aktiviert werden
  - Entsprechendes Bit im PORTx-Register setzen
- Taster zieht den Pegel auf GND
  - Bit im PINx-Register ist 0, wenn Taster gedrückt

## U4-5 Peripherie

- Der ATmega8 läuft mit einem internen Takt von 1 MHz.
- Angeschlossene Peripherie:

| ATmega8                |                     |
|------------------------|---------------------|
| Taster FG-Ampel (INT0) | Port D Pin 2        |
| LED grün               | Port B Pin 0        |
| LED gelb               | Port B Pin 1        |
| LED rot                | Port B Pin 2        |
| LED blau               | Port D Pin 7        |
| Weitere LEDs           | Port D Pins 0,1,5,6 |
| Taster S1,S2 (INT1,-)  | Port D Pins 3,4     |

- Adressen der Port-Register im I/O-Adressbereich:

| Register | Adresse | Register | Adresse |
|----------|---------|----------|---------|
| DDRB     | 0x37    | DDRD     | 0x31    |
| PORTB    | 0x38    | PORTD    | 0x32    |
| PINB     | 0x36    | PIND     | 0x30    |

### 2 HapSIM

- Simulation der Peripheriegeräte unseres Ampelboards
- Verbindet sich mit dem Simulator von AVR Studio
- Start über das Startmenü  
*Start - Alle Programme - HapSIM 2.12*
- Konfiguration für das Ampelboard laden  
*Q:laufgabe3lampel-hapsim.xml*
- Richtigen Mikrokontroller auswählen (ATmega8)
- Sicherstellen, dass HapSIM mit AVR Studio verbunden ist
  - ◆ *Options - AVRStudioHook* sollte aktiv sein
- Die Statuszeile sollte *Ready* anzeigen