Skriptum Mikrocontroller

Teil 2 Programmieren der Pins

DI Wolfgang Zukrigl, HTL Litec
DI(FH) Andreas Pötscher, HTL Litec

Inhaltsverzeichnis

Datum	Beschreibung der durchgeführten Änderung	Autor
08.09.2018	V2.0 . . . vollständige Neu-Erstellung des Skriptums, Verwenden der Bibliothek LitecAVRTools	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
16.02.2019	V2.1 . . . direktes Programmieren der Special Function Register ausführlicher erklärt	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
16.02.2019	V2.2 . . . Aufteilung des Gesamtskriptums in drei Teile	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
26.08.2024	V2.3 . . . übernahme im Markdown und allgemeine Überarbeitung	Andreas Pötscher, HTL Linz–Paul-Hahn-Straße (LiTec)

Special-Function-Register für GPIO-Pins

Special-Function-Register (SFRs) sind im Microcontroller das Bindeglied zwischen CPU und Peripherie. Mit ihnen lassen sich nicht nur GPIO-Pins programmieren, sondern auch z.B: der Analog-Digitalwandler des ATmega2560, die seriellen Schnittstellen, usw. Damit SFRs im C-Programm verwendet werden können, muss folgender Header inkludiert werden:

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

Für jeden GPIO-Port existieren drei Special-Function-Register. Für Port L beispielsweise heißen diese drei Register.

```
DDRL
PORTL
PINL
```

Für Port A (also die GPIO-Pins PA0 bis PA7) heißen diese drei Register dementsprechend `DDRA`, `PORTA` und `PINA`. Wenn im Folgenden die Registernamen `DDRn`, `PORTn` und `PINn` verwendet werden, so steht „n“ für einen der Port-Kennbuchstaben A, B, ...

Die allermeisten Register der AVR-Microcontroller sind 8 Bit, also ein Byte, breit. In ein Register können also 256 (=2^8) verschiedene Bitkombinationen geschrieben werden. Allerdings stellen diese Bitkombinationen normalerweise keine Zahlen dar, sondern jedes einzelne Bit hat seine eigene Bedeutung. Die 8 Bits eines Registers müssen also als acht einzelne, voneinander unabhängige Bits betrachtet werden, von denen jedes den Wert 0 oder 1 annehmen kann. Allerdings können in der Programmiersprache C die Register immer nur Byte-weise angesprochen werden. Das heißt, es können immer nur alle 8 Bits eines SFRs gemeinsam gelesen oder beschrieben werden.

> **Wichtig!**
>
> *Zum jedem Port gibt es drei Special-Function-Register namens `DDRn`, `PORTn` und `PINn` („n“ ist da-bei der Kennbuchstabe des Ports). Jedes dieser Register beinhaltet acht voneinander unabhängige Bits.*

Hexadezimal System

Um SFR zu setzen wird üblicherweise das hexadezimal System verwendet. Diese System hat die Basis 16. Mit einer Stelle können also Zahlen von 0 bis 15 dargestellt werden. Diese werden als 0 bis F(15) geschrieben. Dies hat den Vorteil gegenüber Dezimalzahlen, dass mit einer Stelle im hexadezimal System genau 4 Bit beschrieben werden könnnen. Ein 8 Bit breites Register, so wie beim ATMega2560, kann also mit einer zweistelligen hexadezimal Zahl beschrieben werden. Wobei die erste Stelle die höherwertigen 4 Bits darstellt und die zweite Stelle die 4 niederwertigsten Bits.

> **Beispiel:**
>
> Es sollen die Bits 0, 3 und 5 (auf den Wert 1) gesetzt werden. Im Code wird um Hexadezimal Zahlen darzustellen die Vorsilbe `0x` verwendet.

```
//Stelle        7654 3210
//Binär         0010 1001
//Wertigkeit    8421 8421
//Dezimal       2    9
//Hex         0x2    9
DDRA = 0x29;
```

Bei der ersten Stelle ist nur das Bit 5 gesetzt. Es hat die Wertigkeit 2. Bei der zweiten Stelle sind die Bits 0 und 3 gesetzt. Die Wertigkeiten ergeben zusammen 8+1 = 9.

Beispiel:

Es sollen die Bits 0, 2, 3, 5 und 7 (auf den Wert 1) gesetzt werden.

```
//Stelle        7654 3210
//Binär         1010 1101
//Wertigkeit    8421 8421
//Dezimal       10   13
//Hex         0xA    D
DDRA = 0xAD;
```

Bei der ersten Stelle ist sind die Bits 5 und 7 gesetzt. Es hat die Wertigkeit ergeben als Dezimalzahl 10. Das ist als Hexadezimalzahl A. Bei der zweiten Stelle sind die Bits 0, 2 und 3 gesetzt. Die Wertigkeiten ergeben zusammen als Dezimalzahl 13. Als Hexadezimalzahl ist das D.

Das DDRn Register

Die acht Bits des DDRn-Registers bestimmen, ob die 8 GPIO-Pins des Ports „n“ Ein- oder Ausgänge sind. Ist ein Bit auf 1, so ist der zugehörige GPIO-Pin ein Ausgang. Ist ein Bit hingegen 0, so ist der zugehörige GPIO-Pin ein Eingang.

Das Bit Nr. 7 im DDRn-Register ist dabei für den GPIO-Pin Pn7 zuständig, das Bit Nr. 6 für GPIO-Pin Pn6, usw.

Beispiel:

Welche GPIO-Pins von Port B sind nach folgendem Befehl Eingänge, und welche Ausgänge?

```
DDRB = 0xA3;
```

Lösung:

```
//Stelle        7654 3210
//Wertigkeit    8421 8421
//Hex         0xA    3
//Dezimal       10   3
//Binär         1010 0011
```

Da die Bits Nr. 7, 5, 1 und 0 im `DDRB`-Register auf eins gesetzt wurden, sind die GPIO-Pins PB7, PB5, PB1 und PB0 Ausgänge. Die anderen Bits (Bit Nr. 6, und Bits Nr. 4 bis 2) im `DDRB`-Register wurden auf 0 gesetzt, daher sind die zugehörigen GPIO-Pins PB6, PB4, PB3 und PB2 nun Eingänge.

Aufgabe:

Wie lautet der Befehl, mit dem die GPIO-Pins PL3 und PL2 zu Eingängen gemacht werden, die anderen GPIO-Pins von Port L hingegen zu Ausgängen?

Das PORTn-Register

Auch bei diesem Register ist jedes Bit für den korrespondierenden GPIO-Pin „zuständig“. Zum Beispiel bestimmt Bit Nr. 3 im Register `PORTD` bestimmte Eigenschaften von GPIO-Pin PD3.

Allerdings haben die Bits in einem `PORTn`-Register eine Doppelfunktion, je nachdem, ob die zugehörigen GPIO-Pins Ein- oder Ausgänge sind:

Ist ein GPIO-Pin ein Ausgang, so gibt eine 0 im zugehörigen Bit des `PORTn`-Registers einen Low-Pegel (0 Volt) an diesem Ausgangspin aus. Ein 1-Bit im `PORTn`-Register gibt einen High-Spannugnspegel (5 Volt) am Ausgangspin aus.

Wenn ein GPIO-Pin ein Eingang ist, so aktiviert bzw. deaktiviert das zugehörige Bit im `PORTn`-Register den internen pullup-Widerstand des Eingangspins. Mit einem auf 1 gesetzten Bit im `PORTn`-Register wird der zugehörige interne pullup-Widerstand aktiviert, mit einem auf 0 gesetzten Bit hingegen deaktiviert.

Beispiel:

Was bewirken die beiden folgenden Befehle?

```
DDRD = 0xF0;
PORTD = 0xCC;
```

Lösung:

```
//Stelle          7654 3210
DDRD = 0xF0;    //1111 0000
PORTD = 0xCC;   //1100 1100
```

Bei den GPIO-Pins PD7 und PD6 sind sowohl im `DDRD`, als auch im `PORTD`-Register Einsen an den Bit-Positionen 7 und 6. Die Einsen im `DDRD`-Register machen die beiden GPIO-Pins zu Ausgängen, die Einsen im `PORTD`-Register geben High-Pegel (5 Volt) an den Ausgangs-Pins aus. PD7 und PD6 sind also Ausgänge, die einen High-Pegel ausgeben

An den Bit-Positionen 5 und 4 sind im `DDRD`-Register Einsen, wodurch die Pins PD5 und PD4 zu Ausgängen werden. Im `PORTD`-Register sind die Bits Nr 5 und 4 hingegen Nullen, was für Ausgangspins das Ausgeben einen Low-Pegels (0 Volt) bewirkt. Die GPIO-Pins PD5 und PD4 sind daher Ausgänge, die einen Low-Pegel (0 Volt) ausgeben

Die Bits Nr 3 bis 0 im `DDRD`-Register sind Nullen, wodurch die Pins PD3, PD2, PD1 und PD0 Eingänge werden. Die zugehörigen Bits im `PORTD`-Register bestimmen für Eingangspins, ob der interne pullup-Widerstand aktiviert ist, oder nicht. Bei den Eingangs-Pins PD3 und PD2 ist der interne pullup-Widerstand aktivert

Die Eingangspins PD1 und PD0 sind Eingänge ohne internem pullup-Widerstand, da die Bits Nr. 1 und 0 im `PORTD`-Register null sind

Aufgabe:

Wie lauten die beiden Befehle, die die GPIO-Pins von Port A in jene Betriebsart versetzen, die in der folgenden Tabelle angegeben ist?

PIN	Einstellung
PA0	Eingang mit aktiviertem internen pullup-Widerstand
PA1	Ausgang, der einen Low-Pegel ausgibt
PA2	Ausgang, der einen High-Pegel ausgibt
PA3	Eingang ohne aktiviertem internen pullup-Widerstand
PA4	Ausgang, der einen High-Pegel ausgibt
PA5	Eingang mit aktiviertem internen pullup-Widerstand
PA6	Eingang ohne aktiviertem internen pullup-Widerstand
PA7	Ausgang, der einen Low-Pegel ausgibt

Das PINn-Register

Die beiden bisher behandelten Register (`DDRn` und `PORTn`) werden vom C-Programm aus mit einem Wert beschrieben, und die GPIO-Peripherie versetzt die GPIO-Pins dementsprechend in eine bestimmte Betriebsart. Das `PINn`-Register arbeitet in die umgekehrte Richtung: Bits dieser Register werden von der GPIO-Peripherie auf null oder eins gesetzt, und der Wert dieser Register wird im C-Programm eingelesen.

Im Skriptum „Skriptum-GPIOs_Teil-1_V2.2“ wurde gesagt, dass bei Eingangs-Pins der Pufferverstärker (input-Buffer) den tatsächlich am Pin vorhandenen Spannungspegel (0 Volt oder 5 Volt) „misst“, und diesen in eine logische 0 bzw. eine logische 1 für das Programm „übersetzt“. Diese logische 0 für einen Low-Pegel bzw. die logische 1 für einen High-Pegel schreibt die GPIO-Peripherie in ein Bit von einem der `PINn`-Register. Im Programm wird der Wert dieses `PINn`-Registers gelesen.

Übrigens misst der Pufferverstärker nicht nur den Spannungspegel von Eingangs-Pins, sondern auch von Ausgangs-Pins. Die acht Bits eines `PINn`-Registers enthalten also die Information, an welchen der acht Pins eines Ports ein High- und an welchen ein Low-Pegel anliegt.

Beispiel:

Der Pin PC6 wird mit einem Taster und einem externen pullup-Widerstand beschaltet: Der pullup-Widerstand ist zwischen PC6 und VCC geschaltet, und der Taster zwischen PC6 und GND (Skriptum-GPIOs_Teil-1_V2.2). Im Programm soll der Anweisungsblock einer if-Verzweigung nur dann ausgeführt werden, wenn der Taster gedrückt ist. Wie lautet der entsprechende if-Befehl? ***Lösung:*** Das Bit Nr. 6 im Special-Function-Register PINC nimmt folgende beiden Werte an: Eins, wenn der Taster nicht gedrückt ist, da in diesem Fall der Pin PC6 über den pullup-Widerstand mit VCC (positive Versorgungsspannung, +5V) verbunden ist, und am Widerstand kein Spannungsabfall entsteht, da kein Strom über den Widerstand fließt. Null, wenn der Taster gedrückt ist, da in diesem Fall der Pin PC6 über den geschlossenen Taster direkt mit GND (Minuspol der Versorgungsspannung, 0V) verbunden ist. Am pullup-Widerstand fallen in diesem Fall 5V ab, da er zwischen VCC und GND liegt.

Als erstes werden alle Bits des PINC-Regsiters ausmaskiert, mit Ausnahme des Bits Nr. 6. Dieses Ausmaskieren erfolgt mit einer bitweisen UND-Verknüpfung mit dem Bitmuster `0x40`. Dieses Bitmuster enthält lauter Nullen, nur in Bit Nr. 6 ist eine Eins. Falls der Taster nicht gedrückt ist, ist Bit Nr. 6 im PINC-Register eins. Das Ergebnis dieser UND-Verknüpfung lautet in diesem Fall:

```
// PINC         ?1?? ????
// 0x40         0100 0000
// PINC & 0x40  0100 0000
```

Ein Fragezeichen in der obigen Tabelle bedeutet, dass das entsprechende Bit im PINC-Register entweder 1 oder 0 ist, sein Wert ist also unbekannt. Da alle diese Bits jedoch mit einer 0 UND-verknüpft werden, ist das zugehörige Ergebnisbit auf jeden Fall gleich null (Eins & Null ergibt Null, Null & Null ergibt ebenfalls Null).

Falls der Taster gedrückt ist, ist Bit Nr. 6 im PINC-Register null. Die bitweise UND-Verknüpfung `PINC & 0x40` liefert daher das Bitmuster `0x00`:

```
// PINC         ?0?? ????
// 0x40         0100 0000
// PINC & 0x40  0000 0000
```

Zusammengefasst: Bei gedrücktem Taster liefert `PINC & 0x40` als Ergebnis `0x00`, bei nicht gedrücktem Taster ist das Ergebnis hingegen `0x40`. Der Anweisungsblock der folgenden if-Verzweigung wird also nur bei gedrücktem Taster ausgeführt:

```
if((PINC & 0x40) == 0x00)
{
   //Button pressed ...
}
```

Aufgabe:

Ein Taster soll an den GPIO-Pin PL3 so angeschlossen werden, dass ein Drücken des Tasters einen High-Spannungspegel an PL3 hervorruft.

- Wie muss der Taster an den Pin angeschlossen werden?
- Welche Werte müssen in das PORTL- und DDRL-Register geschrieben werden? Die anderen Pins von Port L sollen Eingänge mit deaktiviertem internen pullup-Widerstand bleiben (wie nach einem Reset des Mikrocontrollers).
- Wie lautet eine if-Abfrage, deren Anweisungsblock nur bei gedrücktem Taster ausgeführt wird?