3.2 Atmega-Programmierung in C/Zugriff auf Register

3.2.1 SFR-Register in C

In der Assembler-Sprache ist der Zugriff auf die SFR-Register problemlos. Es gibt dafür spezielle Befehle. Wenn man wollte, könnte man die SFR-Register außerdem als Teil des SRAM betrachten, da sie ja über SRAM-Adressen ansprechbar sind.

Schwieriger ist der Zugriff in C. Dort gibt es keine Sonderbefehle für direkte Ein-/Ausgabe. Was kann man tun? Man kann zum Beispiel die Sprache erweitern, um per Befehl spezielle Zugriffe zu erlauben. Diese Erweiterungen packt man in spezielle Funktionen, um den Programmierer nicht zu verwirren. Das war der Fall in älteren AVR-GCC-Versionen.

Eine andere Möglichkeit ist die, dass man übliche Sprachelemente verwendet. In diesem Fall muss der Compiler in diesen Sprachelementen die Besonderheiten erkennen und die speziellen Zugriffe bauen. Dies ist in heutigen AVR-GCC-Versionen der Fall.

3.2.2 Register schreiben

```
1    PORTC=80;
2    /* frueher: outb(PORTC,80); so wie in vielen C-Umgebungen! */
```

Wie geht das? Siehe Include-Dateien (`/usr/lib/avr/include`):

a) `<avr/io.h>` ruft `<avr/iom32.h>` auf, falls ein bestimmtes define gesetzt ist.

b) `PORTC` wird dort zu `_SFR_IO8(0x15)`.

c) In `<avr/sfr_defs.h>`: `_SFR_IO8(0x15)` wird zu `0x15+_SFR_OFFSET`.

d) **Vorsicht bei mixed-language-Programmierung:** In Assembler ist PORTC=0x15, hier nicht. In `<avr/sfr_defs.h>` wird eine Möglichkeit gezeigt, damit umzugehen. Suche nach `_SFR_ASM_COMPAT` und `__ASSEMBLER__`.

```
1 #define PORTC _SFR_IO8(0x15) /* <avr/iom32.h>     */
2 #define __SFR_OFFSET 0x20    /* <avr/sfr_defs.h> */
3 #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
4 #define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)
```

Damit wird PORTC=80; zu:

```
1    *((char *)0x35)=80;
```

Der Compiler macht daraus jedoch keinen `lds`-, sondern einen `out`-Befehl.

3.2.3 Register lesen

```
1 x=PINB;
2 /* frueher: x=inb(PINB); so wie in vielen C-Umgebungen! */
```

3.2.4 Einzelbits schreiben

Besonders einfach ist es, ein Bit eines Ports zu setzen, wenn die anderen Bits geändert werden dürfen. Im folgenden Beispiel soll Bit PC7 von PORTC auf 1 gesetzt werden:

```
1 PORTC=_BV(PC7);
```

PC7 ist definiert als 7 (<avr/iom32.h>). _BV(7) ist ein Makro, das zu 1<<7=128=0x80 aufgelöst wird (<avr/sfr_defs.h>).

Schwieriger ist es, ein Bit eines Ports setzen, wenn die anderen Bits nicht geändert werden dürfen. Dann muss eine Oder-Verknüpfung mit der Bitmaske stattfinden:

```
1 PORTC=PORTC|_BV(PC7);   /* ausfuehrlich */
2 PORTC|=_BV(PC7);        /* Kurzform */
3 /* frueher: sbi(PORTC, PC7); in <compat/deprecated.h> */
```

Wenn man ein Bit eines Ports löschen, alle anderen Bits aber unverändert lassen möchte, braucht man eine Und-Verknüpfung mit der negierten Bitmaske:

```
1 PORTC&=~_BV(PC7);
2 /* alt: cbi(PORTC, PC7 */
```

In _BV(PC7) ist nur Bit Nr. 7 auf 1 gesetzt. In ~_BV(PC7) sind alle Bits außer Nr. 7 auf 1 gesetzt. Nur Bit Nr. 7 liegt auf 0. Durch die Und-Verknüpfung wird nun Bit Nr. 7 als einziges Bit auf 0 gesetzt.

3.2.5 Einzelbits lesen

Zum Lesen eines einzelnen Bits nimmt man eine Bitmaske für dieses Bit und verknüpft sie mit dem Port-Byte:

```
1 if(PORTC&_BV(PC7))
2     ...
3 if(!(PORTC&_BV(PC6)))
4     ...
```

Andere Schreibweise:

```
1 if(bit_is_set(PORTC, PC7)
2     ...
3 if(bit_is_clear(PORTC, PC6)
4     ...
```

In <avr/sfr_defs.h>:

```
1 #define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
2 #define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
```

3.2.6 Einzelbitschleife

```
1 do{}while(PORTC&_BV(PC7));      /* wartet auf PC7=0 */
2 do{}while(!(PORTC&_BV(PC6)));  /* wartet auf PC6=1 */
```

Mit den richtigen Makros kann man dafür eine andere Schreibweise benutzen:

```
1 loop_until_bit_is_clear(PORTC, PC7);
2 loop_until_bit_is_set(PORTC, PC6);
```

In <avr/sfr_defs.h>:

```
1 #define loop_until_bit_is_clear(sfr,bit) do{}while(bit_is_set(sfr,bit))
2 #define loop_until_bit_is_set(sfr,bit) do{}while(bit_is_clear(sfr,bit))
```