

Mikroprozessorpraktikum WS 2011/12  
Aufgabenkomplex: 1

Teilnehmer:

Marco Träger, Matr. XXXXXXXXX

Alexander Steen, Matr. 4357549

Gruppe: XXX, Arbeitsplatz: XXX

## A 1.1 Output

### A 1.1.1 Erläutern Sie unter Nutzung des User's Guide die Funktionalität der vier Register

Alle der folgenden Register haben einen Ein-Byte-Wert, besitzen also jeweils eigene Werte für jeden Pin  $x$  des Ports ( $x = 0 \dots 7$ ).

**P4SEL.x** selektiert für Bit  $x$  des Ports 4 ob der I/O-Port oder die Funktion eines Peripheriegerätes genutzt werden soll

**0:** I/O-Funktion

**1:** Funktion je nach Peripheriegerät

**P4DIR.x** legt die Richtung des I/O Pins  $x$  des Ports 4 fest

**0:** Input (um ein Signal zu lesen)

**1:** Output (um das Signal anzulegen)

**P4OUT.x** ist das Signal, das am Pin  $x$  des Ports 4 anliegen wird, wenn man **P4DIR.x** = 1 und **P4SEL.x** = 0 setzt

**P4IN.x** ist das Signal, das am Pin  $x$  des Ports 4 anliegt, wenn **P4SEL.x** = 0. Ansonsten das Signal welches am **ModuleXOut** anliegt.

### A 1.1.2 Stellen Sie eine Liste der Operatoren zur Bitmanipulation auf. Erklären Sie die Möglichkeiten zum Setzen, Rücksetzen und Toggeln einzelner bzw. mehrerer Portleitungen eines Ports am Beispiel von P4OUT.

#### Operatoren:

Operation	Wirkung
$x \& y$	Bitweises UND von $x$ und $y$
$x   y$	Bitweises ODER von $x$ und $y$
$x \wedge y$	Bitweises XOR von $x$ und $y$
$\sim x$	Bitweises NOT von $x$
$x \ll y$	Linksshift von $x$ um $y$ Bits, es werden Nullen von Rechts eingefügt
$x \gg y$	Rechtsshift von $x$ um $y$ Bits, es werden Nullen von Links eingefügt

Es gibt alle diese Operatoren auch mit einem folgenden Gleichzeichen, dabei wird das Ergebnis der Operation dem linken Operanden zugewiesen.

#### (Rück-)Setzen von Bits:

Operation	Wirkung
$P4OUT \mid= 1 \ll x$	setzt das $x$ -te Bit von P4OUT
$P4OUT \&= \sim(1 \ll x)$	setzt das $x$ -te Bit von P4OUT zurück
$P4OUT \sim= (1 \ll x)$	invertiert das $x$ -te Bit von P4OUT
$P4OUT \mid= maske$	setzt alle Bits von P4OUT, die in <b>maske</b> gesetzt sind
$P4OUT \&= \sim maske$	setzt alle Bits von P4OUT zurück, die in <b>maske</b> gesetzt sind
$P4OUT \sim= maske$	invertiert alle Bits von P4OUT, die in <b>maske</b> gesetzt sind

### A 1.1.3 Erläutern Sie anhand der Abbildung der inneren Struktur einer Portleitung für die folgenden Registerbelegungen den Signalpfad und den Logikpegel der Portleitung P4.0.

```
1 # define LED_ROT (0x01)
2 // die Select-Einstellungen aller acht Portleitungen
3 // von P4 werden auf die I/O Funktion geschaltet
4 P4SEL = 0x00;
5 // die Richtung der unteren 4 Portleitungen
6 // werden auf Out geschaltet
7 P4DIR = 0x0F;
8 // P4OUT.0 wird auf 1 gesetzt
9 P4OUT |= LED_ROT;
10 // also wird an Portleitung P4.0 eine 1 anliegen
```

Der Signalpfad ist dabei: HIER FEHLT EIN BILD

- A 1.1.4** Warum leuchtet eine LED am Port P4.0, wenn der Logikpegel "0" ist und nicht bei dem Logikpegel "1"?

Betrachten wir die obige Abbildung, P4.0 ist in der Abbildung mit P4B0 beschriftet, die LED am Port P4.0 mit LED1. An der LED1 liegt an der rechten Seite die Betriebsspannung an. Die linke Seite ist an die Betriebsspannung angeschlossen wenn der Logikpegel "1" ist, wenn der Logikpegel "0" ist die Erdung. Wenn der Logikpegel "0" anliegt, existiert eine Potentialdifferenz an der Diode, also liegt eine Spannung an. Damit fließt Strom durch die LED und darum leuchtet sie. Wenn der Logikpegel "1" anliegt, liegt keine Potentialdifferenz vor.

- A 1.1.5** Erläutern Sie inhaltlich die Bedeutung und Funktionalität der folgenden Codezeilen:

```
1 unsigned char a;
2 #define LEDRT (0x01)
3 P4DIR = 0x00;
4 a = 10;
5 P4OUT = a;
6 P4OUT = 0x01;
7 P4DIR = 0x07;
8 P4OUT = 0x00;
9 P4OUT |= 0x01;
10 P4OUT |= LEDRT;
11 P4OUT ^= ~LEDRT;
12 P4OUT ^= LEDRT;
```

**Zeile 1** Deklaration einer nicht-vorzeichenbehafteten 8-Bit Variablen mit dem Namen a

**Zeile 2** Definition eines ALIAS für die 8-Bit Zahl 1, jedes Vorkommen des ALIAS wird während der Übersetzung vom Precompiler durch die 8-Bit Zahl 1 ersetzt

**Zeile 3** Stellt jede Portleitung des Port 4 auf Eingang (IN)

**Zeile 5** Legt an Portleitung 1 und 3 von Port 4 eine 1 an, an alle anderen eine 0

**Zeile 6** Legt an Portleitung 0 von Port 4 eine 1 an, an alle anderen eine 0

**Zeile 7** Stellt Portleitung 0,1,2 von Port 4 auf Ausgang (OUT)

**Zeile 8** Legt an Portleitung allen Portleitungen von Port 4 eine 0 an

**Zeile 9** Legt an Portleitung 0 von Port 4 eine 1 an, an alle anderen eine 0

**Zeile 10** Keine Veränderung (legt an Portleitung 0 von Port 4 eine 1 an, an alle anderen eine 0)

**Zeile 11** Legt an allen Portleitungen von Port 4 eine 1 an

**Zeile 12** Legt an Portleitung 0 von Port 4 eine 0 an, an allen anderen eine 1.

- A 1.1.6** Schreiben Sie ein kleines Programm, welches den Durchlauf einer Sequenz eines Ampelsignals mit den Phasen (Rot, Rot-Gelb, Grün, Gelb, Rot) simuliert. Nutzen Sie dazu die bereitgestellte Funktion `wait()` für eine Zeitschleife. Benutzen Sie eigene Macros und Operatoren zur Bitmanipulation.

```
1 #define ROT      ~(0x01)
2 #define GELB     ~(0x02)
3 #define ROTGELB  ~(0x03)
4 #define GRUEN    ~(0x04)
5 #define WAITTIME 50000
6
7 void aufgabe116() {
8
9     P4SEL = 0x00;
10    P4DIR = 0x07;
11
12    P4OUT = ROT;
13    wait(WAITTIME);
14    P4OUT = ROTGELB;
15    wait(WAITTIME);
16    P4OUT = GRUEN;
17    wait(WAITTIME);
18    P4OUT = GELB;
19    wait(WAITTIME);
20    P4OUT = ROT;
21    wait(WAITTIME);
22 }
```

Erklärung fehlt hier

## A 1.2 Input

- A 1.2.1** Erläutern Sie unter Nutzung des User's Guide die Funktionalität der vier Register

**P1DIR** entscheidet, ob der jeweilige Pin als Eingang oder Ausgang fungiert, dabei beschreibt 0 einen Eingang, 1 einen Ausgang

**P1IN** besteht aus einem Byte, deren Bits den aktuellen Logikpegel an dem jeweiligen Pin des Ports 1 darstellen

**P1OUT** zeigt an dem jeweiligen Bit an, welcher Logikpegel an dem zugehörigen Port anliegen soll, falls P1DIR auf Ausgang und P1SEL auf I/O-Funktion geschaltet ist

**P1SEL** gibt an, ob die einzelnen Pins des Port 1 direkt als I/O benutzt werden (Wert 0) oder für ein angeschlossenes Modul (Wert 1)

**P1IE** de-/aktiviert die Intertupt-Flags (P1IFG) für die Pins des ersten Ports.

**P1IES** entscheidet, ob man Interrupt durch eine low-high-Flanke (0) oder eine high-low-Flanke (1) auf dem jeweiligen Pin ausgelöst werden soll.

**P1IFG** bezeichnet die Interrupt-Flags der Pins von Port 1. Ist ein Bit von P1IFG auf 1 gesetzt, so wurde von dem zugehörigen Pin ein Interrupt ausgelöst.

- A 1.2.2** Erläutern Sie die Funktion des Operators AND zur Bitmanipulation. Diskutieren Sie die Einsatzmöglichkeit am Beispiel einer IF-Anweisung

Der AND-Operator (&) führt bitweise die Verundung der Bits der Arguments aus.

```
if (P1IN & Taster) {...}
```

Geht man für das Codebeispiel davon aus, dass an dem Pin  $i$  von Port 1 ein Taster angeschlossen ist, so kann man durch Wahl der Variable **Taster** als Bitmaske, die nur an der Stelle  $i$  eine 1 enthält (**Taster** =  $2^i$ ), erreichen, dass die Abfrage genau dann erfolgreich ist, falls der Taster gedrückt wurde.

**A 1.2.3** Erklären Sie die nachfolgenden Befehlszeilen und geben Sie an, welchen Wert die Variable **a** in den einzelnen Zeilen annimmt.

```
1 #define Taster_rechts (0x01)
2 #define Taster_links (0x02)
3 P1DIR = 0x00;
4 P4DIR = 0xFF;
5 P4OUT = 0;
6 a = 7;
7 P4OUT = a;
8 P1OUT = a;
9 a = P1IN & 0x30; //beide Tasten gedr.
10 a = P1IN & 0x00; //Taste rechts gedr.
11 a = P1IN & 0x01; //Taste rechts gedr.
12 a = P1IN & 0x02; //Taste rechts gedr.
13 a = P1IN & 0x03; //Taste links gedr.
14 a = P1IN & 0x03; //beide Tasten gedr.
15 P4OUT = P1IN & Taster_rechts; //Taster an P1.0 nicht gedr.
16 P4OUT = P1IN & Taster_links; //Taster an P1.0 gedr.
```

**Zeile 1,2** Definiert Bitmasken, auf welche Bits der Register der rechte bzw. linke Taster zugreift

**Zeile 3** Alle Pins von Ports 1 werden auf Eingang geschaltet

**Zeile 4** Hier werden nun alle Pins von Ports 4 auf Ausgang geschaltet

**Zeile 5** An alle Pins von Port 4 werden die Logikpegel 0 angelegt

**Zeile 6** **a** wird auf 7 gesetzt

**Zeile 7** Setzt die unteren drei Bits von **P4OUT** auf 1. Wenn **P4SEL** für die unteren drei Pins auf I/O-Funktion gestellt ist, liegt an diesen Pins nun jeweils eine 1 an (die LEDs leuchten nicht).

**Zeile 8** Setzt die unteren drei Bits von **P1OUT** auf 1. Da **P1DIR** auf Eingang steht, ändert sich nichts.

**Zeile 9** Da die beiden Tasten die beiden untersten Bits sind, ist das Ergebnis der Verundung 0, also wird **a** = 0 gesetzt

**Zeile 10** Hier wird mit Und auf 0 ausgeführt, also wird **a** = 0 gesetzt

**Zeile 11** **a** = 1, da Taster gedrückt

**Zeile 12** **a** = 0, da mit der Bitmaske für den linken Taster verglichen wird

**Zeile 13** **a** = 2, weil der Wert des linken Tasters genommen wird (an der zweiten Stelle in der Maske)

**Zeile 14** **a** = 3, da sowohl der Wert des linken Tasters (2) und des rechten (1) genommen wird

**Zeile 15** **P4OUT** wird auf 0 gesetzt, da kein Taster gedrückt ist

**Zeile 16** P4OUT wird auf 0 gesetzt, da die Bitmaske den Tasterwert von P1.0 nicht berücksichtigt

**A 1.2.4** Schreiben Sie ein Programm, das die Ampelphasen simuliert.

```
1  #define Taster_rechts (0x01)
2  #define Taster_links (0x02)
3  #define rot (0x01)
4  #define gelb (0x02)
5  #define gruen (0x04)
6
7  void aufgabe_1_2_4() {
8      // Letzten beiden Pins von Port 1 als I/O-Input verwenden
9      P1SEL &= ~0x03;
10     P1DIR &= ~0x03;
11     // Letzten drei Pins von Port 4 als I/O-Output verwenden
12     P4SEL &= ~0x07;
13     P4DIR |= 0x07;
14
15     if (~((P1IN & Taster_rechts) ^ (P1IN & Taster_links))) {
16         // Beide Tasten bzw. keine von beiden
17         P4OUT &= ~gelb;
18     } else if((P1IN & Taster_rechts) & ~(P1IN & Taster_lins))
19     {
20         // Rechte Taste
21         P4OUT &= ~gruen;
22     } else if(~(P1IN & Taster_rechts) & (P1IN & Taster_lins))
23     {
24         // Linke Taste
25         P4OUT &= ~rot;
26     }
27 }
```

## A 1.3 Ampel

### A 1.3.1

## A 1.4 Taster

### A 1.4.1