Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 1

Teilnehmer:

Marco Träger, Matr. XXXXXXXX
Alexander Steen, Matr. 4357549

Gruppe: XXX, Arbeitsplatz: XXX

A 1.1 Output

A 1.1.1

A 1.1.2

A 1.1.3

A 1.1.4

A 1.1.5

A 1.1.6

A 1.2 Input

A 1.2.1

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

Der AND-Operator (&) führt bitweise die Verundung der Bits der Arguments aus.
Geht man für das Codebeispiel

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

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

```
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

```
1  void aufgabe() {
2    #define Taster_rechts (0x01)
3    #define Taster_links (0x02)
4    #define rot (0x01)
5    #define gelb (0x02)
6    #define gruen (0x04)
7    P1SEL = 0x00;
8    P1DIR = 0x00;
9    P4SEL = 0x00;
10   P4DIR = 0x07;
11
12   if (~((P1IN & Taster_rechts) ^ (P1IN & Taster_links))) {
```

```
13      P4OUT = gelb;
14    } else if((P1IN & Taster_rechts) & ~(P1IN & Taster_lins)) {
15      P4OUT = gruen;
16    } else if(~(P1IN & Taster_rechts) & (P1IN & Taster_lins)) {
17      P4OUT = rot;
18    }
19  }
```

A 1.3 Ampel

A 1.3.1

A 1.4 Taster

A 1.3.1