Mikroprozessor Praktikum
Fr. HWP 2
Aufgabenblock 1

Paul Podlech
3910583

Max Wisniewski
4370074

6. Dezember 2011

Block 1: I/O Ports

Aufgabe 2: Output

A1.1.1

Klären Sie die Funktion folgender Register:

P4Sel: Mit diesem Register kann man das Signal von oder an ein internes Modul weiterleiten. Das sorgt dafür, dass man entweder Daten direkt verarbeiten kann ohne das die CPU mitarbeiten muss, oder Signale automatisch rausgeschickt werden können, wie eine Lampe die im Takt blinken soll. Dabei steht **0** für I/O Funktionalität und **1** für Modul Funktionalität.

P4Dir: Legt fest, ob man von diesem Port liest oder auf ihn raus schreiben möchte. Dabei steht:
0 für *IN*, das heißt man will lesen und
1 für *OUT*, das heißt man will schreiben.

P4Out: Hier schreibt man die Daten hinein, wenn man etwas senden möchte. Sobald Direction auf *OUT* steht, ist dieses Signal aussehn sichtbar.

P4In: Von hier kann gelesen werden, welches Signal am vierten Port anliegt, wenn die Direction am *P4Dir* auf *IN* steht. Allerdings reflektiert es auch im *OUT* Fall den Wert, den wir selber setzen.

Dies ist für für alle Ports gleich, wobei man für die anderen Ports die 4 durch die entsprechende Portnummer ersetzen muss.

A 1.1.2

Erstellen Sie eine Liste von Bitoperationen auf und geben Sie Operationen zum Setzen, Toggeln, Rücksetzen an.

Name	Beschreibung	Symbol
AND	Verunded die die Zahlen Bitweise	&
OR	Verodert die Zahlen Bitweise	|
XOR	VerXORt die zaheln Bitweise	HOCH
Komplement	Bildet das Bitweise Komplemet	~
Rechtshift	Shiftet die Bits nach rechts und füllt mit 0en	»
Linksshift	Shiftet die Bits nach links und füllt mit 0en	«

Setzen: $P4OUT \mid = 1 << k$. Setzt das k-te Bit auf 1. Um mehrere Bits zu setzen, können wir mit der Bitmaske verodern, die an allen Stellen eine 1 hat, die wir setzen wollen und eine 0 sonst.

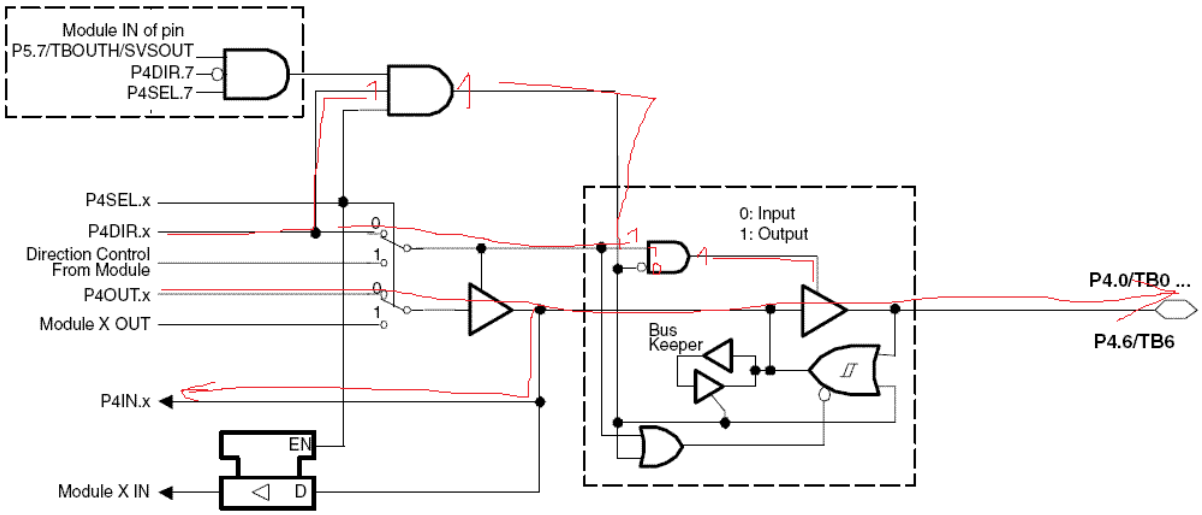
Toggeln: $P4OUT\ XOR = 1 << k$ Setzt das k-te Bit auf sein Komplement. Um mehrere Bits zu setzen, können wir mit der Bitmaske verodern, die an allen Stellen, die wir Toggeln wollen eine 1 und sonst eine 0.

Zurücksetzen: $P4OUT\ \&= \tilde{}(1 << k)$. Die Bitmaske hat an der k-ten Stelle eine 0 und sonst 1en. Damit Wird die k-te Stelle auf 0 gesetzt. Um mehrere Bits zu setzen, können wir mit der Bitmaske verunden, die an jeder Stelle eine 0 hat, die wir zurück setzen wollen und sonst 1en.

A1.1.3 Erläutern Sie anhand der Abbildung der internen Struktur einer Portleitung für die folgende Registerbelegung den Signalpfad und den Logikpegel der Portleitung P4.0

- define LED_ROT(0x01)
- P4SEL = 0x00
- P4DIR = 0x0F
- P4OUT |= LED_ROT

Daraus könen wir herraus lesen, dass Select für alle auf 0 steht, dass heißt es ist als I/O Port geschaltet. Die Direction von 0x0F heißt, dass für Bit 0 eine 1 anliegt (da die unteren 4 Bit gesetzt sind). Wenn LED_ROT mit unserem OUT verodert wird, kann man, wie in der letzen Aufgabe gezeigt, heißt dass, dass das 0te Bit gesetzt wird.



Aus unserem Bild kann man herrauslesen, das an P4.0 eine 1 herraus kommt.

A 1.1..4 Die LED leuchted, wenn wir eine Potentialdifferenz über der LED haben. Da auf der rechten Seite der LED eine Betriebsspannung (von 3V) anliegt, wie man aus dem Schaltbild lesen kann, benötigen wir ein niedrigeres Potential, damit ein Strom fließen kann. Diese wird bei 0 erreicht, da die Spannung zwischen den beiden Potentialen dann groß genug ist. Bei einem Highpegel wird diese Differenz 0 sein.

A 1.1.5 Erläutern Sie inhaltlich die Bedeutung 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;
```

Direction wird 2 mal gesetzt. Nach der ersten Anweisung wird alles auf eingang gesetzt und außen ist nichts zusehen. Nach der zweiten können wir an Leitung P4.0, P4.1, P4.2 ein Signal sehen.
Die Zeilen 5,6 haben keine Auswirkungen, da sie nicht sichtbar sind nach außen.
Nach Zeile 7 können wir die 1 auf P4.0 sehen, die in 6 gesetzt wurde.
Nach Zeile 8 können wir von außen an P4.0, P4.1, P4.2 eine 0 sehen.
Nach Zeile9 wurde auf P4.0 eine 1 gesetzt.
Nach Zeile 10 hat sich nichts geändert, da das Bit 0 schon vorher gesetzt wurde.
Nach Zeile 11 wurde mit dem komplement von LEDRT getoggelt. Das heißt, dass P4OUT nun 0xFF ist. Jedes Bit ist 1 und damit ist an P4.0, P4.1, P4.2 diese auch zu sehen. Die anderen Bits sind immer noch nicht auf OUT geschaltet.
Nach Zeile 12 wurde Bit 0 wieder getoggelt. Das bedeutet das an P4.0 eine 0 anliegt und an P4.1, P4.2 weiter die 1 bleibt.

A 1.1.6 Schreiben Sie ein Programm, dass die Phasen ein Ampel durchläuft. Und zwischen den einzelnen Phasen immer etwas Zeit lässt.
Wir haben uns dafür Makros für die Werte von Rot, Gelb und Grün geschrieben. Diese werden immer Program immer wieder kombiniert und dan auf die Outleitung gesetzt. Beim Aufruf haben wir als erstes darauf geachtet, dass die Leitungs Direction erst auf OUT gesetzt wird, wenn wir im OUT was etwas reingeschrieben haben.

```
1  #define ROT   (0x01) // Bitmaske fuer rote LED
2  #define GELB    (0x02) // Bitmaske fuer gelbe LED
3  #define GRUEN (0x04) // Bitmaske fuer gruene LED
4
5  void aufgabe1(){
6    P4SEL   = 0x00;  // setze den gesammten Port auf OUT
7    P4DIR   = 7  // setzt LED Leitungen auf OUT
8    P4OUT = ~ROT; //setzt die rote LED
9    wait(50000);
10   wait(50000);
11   P4OUT = ~( ROT | GELB); //setzt sowohl die rote als auch
         die gelbe LED
12   wait(50000);
13   P4OUT =   ~(GRUEN);  //setzt die gruene LED
14   wait(50000);
15   wait(50000);
16   P4OUT = ~(GELB)  // gelbe LED auf an
17   wait(50000);
18 }
```

Wir haben uns an dieser Stelle für die einfache Lösung entschieden und überschreiben alle Bits, die wir nicht setzten wollen. Dies könnte man umgehen indem man die Bits an sich setzt. Da in unserem Programm aber nicht mehr passiert, ist das an dieser Stelle vertretbar.

Unsere Beobachtungen waren: Es funktioniert.

Aufgabe 2: Input

A 1.2.1 Erläutern Sie unter Nutzung des *User Guides* die Funktionalität der sieben Register.

P1DIR wurde schon in A 1.1.1 erklärt und bezeichnet die Richtung des Ports (lesen oder schreiben)

P1IN Aus diesem Registern kann man lesen welches Bit auf P1 gesendet / geschrieben / angelegt ist.

P1OUT Wurde schon in A 1.1.1 beschrieben. In dieses Register schreiben wir einen Wert, den wir nach außen sichtbar machen wollen. Dieses Bit ist genau dann sichtbar, wenn P1DIR auf OUT steht.

P1SEL Wurde schon in A 1.1.1 beschrieben. Wählt aus, wer diesen Wert lesen soll. Kann an ein Modul direkt geleitet werden.

P1IE Schaltet für dieses Bit ein, ob es einen Interrupt werfen darf oder nicht.

P1IES Legt fest, ob der Interruptbit bei High-To-Low (Bit:1) oder bei Low-to-High (Bit:0) gesetzt werden soll.

P1IFG Ist die Interruptflag, die zeigt an, ob der Trigger ausgelöst wurde. Dieses Register sollte nach dem aufruf der InterruptServiceRoutine wieder auf 0 gesetzt werden, damit man auf ein neuen Interrupt warten kann.

A 1.2.2 Erläutern Sie die Funktionalität des Operators *AND* zur Bitmanipulation. Diskutieren Sie die Einsatzmöglichkeiten am Beispiel einer IF-Anweisung (IF (PIN1 & Taster)) ...

Die Bitweiseverundung ist folgender Maßen definiert:

$$\begin{aligned} AND: \quad & \mathbb{B}^n \times \mathbb{B}^n \rightarrow \mathbb{B}^n \\ & ((a_0, a_1, ..., a_n)\ ,\ (b_0, b_1, ...b_n)) \mapsto ((a_0 \wedge b_0), (a_1 \wedge b_1), ...(a_n \wedge b_n)) \end{aligned}$$

Bei unseren 8-Bit breiten Ports ist n üblicherweise 8.

In der Anwendung für die If Anweisung, können wir über eine Bitmaske (PIN1) überprüfen, ob ein Taster gesetzt gerückt wurde. Dies können wir realisieren, wenn Taster das Register ist, in das die Taster über den Port in IN schreiben (oder eine Interruptflag gesetzt haben). Nun steht in PIN1 eine Maske, die 1 an der Stelle ist, an der der Taster seine Flag setzt, und sonst 0en hat. Verunden wir nun die beiden, erhalten wir:
0, wenn der Taster nicht gesetzt wurde. Das gesuchte Bit ist 0, und alle anderen werden über die Maske genullt.
$\neq 0$, wenn der Taster gedrückt wurde. Das gesuchte Bit ist 1 und alle anderen 0.

Dies hat zur Folge, dass bei gedrücktem Taster in das IF Konstrukt gesprungen wird und sonst nicht (da if bei arg $\neq 0$/ betreten wird).

A 1.2.3 Nutzen Sie die obere Schaltung zur Erklärung der nachfolgenden dargestellten Befehlszeilen und geben Sie an, welchen Wert die Variable a (unsigned char) in den einzelnen Zeilen annimmt. Beachten Sie dabei den Tastenzustand im Kommentar.

```
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 gedrueckt
10 a = P1IN & 0x00; //Taste rechts gedrueckt
11 a = P1IN & 0x01; //Taste rechts gedrueckt
12 a = P1IN & 0x02; //Taste rechts gedrueckt
13 a = P1IN & 0x03; //Taste links gedrueckt
14 a = P1IN & 0x03; //beide Tasten gedrueckt
15 P4OUT = P1IN & Taster_rechts; //Taster an P1.0 nicht
       gedrueckt
16 P4OUT = P1IN & Taster_links; //Taster an P1.0 gedrueckt
```

Wir beschreiben im folgenden, was die einzelnen Zeilen tun:

Zeile 1 Definiert die Maske, welches Bit der Taster rechts setzt.

Zeile 2 Definiert die Maske, welche Bit der Taster links setzt.

Zeile 3 Der gesammte Port 1 (jedes Bit) wird auf IN gesetzt.

Zeile 4 Wir setzten den gesammten Port 4 auf OUT (die Bits die bisher drauf liegen werden sichtbar)

Zeile 5 Wir zeigen Port 4 auf jedem Bit eine 0

Zeile 6 $a = 7$

Zeile 7 Wir setzen die untersten 3 Bits des Ports auf 1, sollte SEL = 0 und das richtige Modul verbunden sein, würden nun alle LEDs leuchten.

Zeile 8 Wir setzen P1OUT auf a, was über das Programm hinweg nicht sichtbar wird, da P1OUT auf INT steht.

Zeile 9 a ist 0, da beide Taster die unteren beiden Bits sind. Die verundung macht daraus insgesammt eine 0.

Zeile 10 Verundung mit 0 ist immer 0.

Zeile 11 $a = 1$, da Taster 1 in 0x01 schreibt und gedrückt ist.

Zeile 12 $a = 0$, da Taster 2 nicht gedrückt ist.

Zeile 13 $a = 2$, da Taster 2 gedrückt (Wert 2) und Taster 1 nicht gedrückt (Wert 0).

Zeile 14 $a = 3$, das beide gedrückt sind und die werte sich addieren.

Zeile 15 $a = 0$, da P1.0 das Bit ist, das rechten Taster gesetzt wird. Dieses ist nicht gesetzt wird also 0 sein

Zeile 16 a ist nicht bestimmt, da wir den linken Taster überprüfen, aber P4.0 den rechten spezifiziert und auf P4.1, was den linken beschreibt, nicht eingegangen wird.

A 1.2.4 Schreiben Sie ein Programm mit folgender Funktionalität:

- gelbe LED (P4.1) an, wenn keine Taste gedrückt ist
- grüne LED (P4.2) an, wenn rechte Taste gedrückt und linke Taste nicht gedrückt ist
- rote LED (P4.0) an, wenn linke taste gedrückt und rechte Taste nicht
- gelbe LED (P4.1) an, wenn beide Tasten gedrückt sind

Unser Programm sieht folgender Maen aus:

```
1 void coolFunction(){
2
3 }
```

Auswertung:

Aufgabe 3: Ampelsteuerung

A 1.3.1 Nutzen Sie alle drei LED und den rechten Taster (P1.0), um eine Fußgängerampel (aus der Sichtweise des Autofahrers) zu programmieren. Folgender Ablauf soll dabei realisiert werden:

- Grundzustand alle LED aus
- Wenn Taste gedrückt wird, sofort gelbe LED (P4.1) einschalten
- Danach zeitverzögert gelbe LED aus und rote LED (P4.0) an
- Nach einer Pause gelbe LED (P4.1) zur roten LED (P4.0) dazuschalten
- Dann zeitverzögert nur die grüne LED (P4.2) einschalten
- Nach einer weiteren Pause alle LEDs ausschalten
- Erst danach mit einer größeren Wartezeit die Taste wieder abfragen

Unsere Lösung sieht wie folgt aus:

```
1 void myfunntion(){
2   //Deine Mudda
3 }
```

Auswertung:

Aufgabe 4: Tastenstatus abfragen

A 1.4.1 Entwickeln Sie einen Binärzähler, der folgender Beschreibung entspricht:

- Jede Tastenbetätigung der linken Taste (P1.1), soll eine Variabel um eins inkrementieren
- Jede Betätigung der rechten Taste (P1.0), soll eine Variable um eins dekrementieren
- Der Zahlenwert der Variablen soll als Binärzahl mit drei LED dargestellt werden
- Werden beide Tasten gedrückt, wird die Variable auf Null gesetzt

Die LED haben folgende Zuordnung mit dem darstellbaren Zahlenbereich von 0 bis 7:

- 2^0 ist grüne LED (P4.2) mit der Wertigkeit 1
- 2^1 ist gelbe LED (P4.1) mit Wertigkeit 2
- 2^2 ist rote LED (P4.0) mit Wertigkeit 4

Eine leuchtende LED signalisiert dabei den logischen Zustand "1ünd eine ausgeschaltete LED den logischen Zustand "0".

Eine interessante und nicht ganz einfach zu lösende Problematik ergibt sich hier durch das Prellen der Tastenkontakte. Beim Austesten ihrer Lösung werden sie auf dieses Problem stoßen. Finden sie für dieses Problem eine Lösung.

Unsere Lösung sieht folgedner Maßen aus:

```
1 void myfunction(){
2   // bla
3 }
```

Auswertung:

bla

Problem wurde so gelöst.