

Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 2

Teilnehmer:

Marco Träger, Matr. 4130515
Alexander Steen, Matr. 4357549

Gruppe: Freitag, Arbeitsplatz: HWP 1

A 2.1 Taktfrequenz

A 2.1.1 Bestimmen Sie messtechnisch die Frequenz der LFXT1CLK- und XT2CLK-Taktquelle.

Zur Lösung dieser Aufgabe werden zusätzlich zu den in Aufgabenblock 1 beschriebenen Registern das BC2CTL2-Register benutzt. In diesem Register werden einige der Bits für die Taktsteuerung verwaltet. Die zwei höchstwertigsten Bits legen die Taktquelle und die zwei nächst niederen Bits die Auswahl des Taktteilers fest. Für diese Aufgabe wird die Taktteilung ausgeschaltet (also auf den Divisor 1), damit der tatsächliche Takt gemessen wird.

LFXT1CLK Um auf P5.4 das Taktsignal messen zu können, wird nach dem Verbinden der Portleitung mit dem Takt (via P5SEL) das Messgerät angeschaltet. Nun wird das folgende Programm ausgeführt:

```
void aufgabe211() {
    P5SEL |= (1 << 4); // Leitung mit Modul verbinden
    P5DIR |= (1 << 4); // Als Ausgang fungieren
    BC2CTL2 = (BC2CTL2 & ~SELM_3) | SELM_3; // LFXTCLK
    BC2CTL2 = (BC2CTL2 & ~DIVM_3) | DIVM_0; // Divisor 1
}
```

Am Messgerät lässt sich eine Frequenz von 32,76927 kHz messen. Dies ist die Taktfrequenz der LFXT1CLK.

XT2CLK Diese Messung wird analog zur LFXT1CLK-Messung durchgeführt. Einzig die Selektion der Taktquelle ändert sich (siehe Code):

```
void aufgabe211() {
    P5SEL |= (1 << 4); // Leitung mit Modul verbinden
    P5DIR |= (1 << 4); // Als Ausgang fungieren
    BC2CTL2 = (BC2CTL2 & ~SELM_3) | SELM_2; // XT2CLK
    BC2CTL2 = (BC2CTL2 & ~DIVM_3) | DIVM_0; // Divisor 1
}
```

Aus der Messung ergibt sich eine Taktfrequenz von 7,373165 MHz für die XT2CLK-Taktquelle.

A 2.1.2 Bestimmen Sie messtechnisch die minimale und maximale Taktfrequenz des MCLK-Taktes, die sich auf Basis der LFXT1CLK-, XT2CLK- und DCOCLK-Taktquellen bereitstellen lässt. Belegen Sie die Messergebnisse mit einer Berechnung auf Basis aller Komponenten aus den Blockschaltbildern.

LFXT1CLK Als maximale Frequenz dieser Taktquelle kann das Ergebnis aus Aufgabe 2.1.1 hergenommen werden, also $f_{max} \approx 32,76927 \text{ kHz}$. Durch Modifikation mit dem größten Taktteiler (ein Achtel) können wir hier den minimalen Takt f_{min} produzieren. Also ergibt sich als Code:

```
void aufgabe212() {
    P5SEL |= (1 << 4); // Leitung mit Modul verbinden
    P5DIR |= (1 << 4); // Als Ausgang fungieren
    BC2CTL2 = (BC2CTL2 & ~SELM_3) | SELM_3; // LFXTCLK
    BC2CTL2 = (BC2CTL2 & ~DIVM_3) | DIVM_3; // Divisor 8
}
```

Aus der Messung ergibt sich ein Takt von 4,096158 kHz.

Dies deckt sich mit den Erwartungen, da sich für die minimale Taktfrequenz rechnerisch ergibt:

$$f_{min} = f_{max} \cdot \frac{1}{8} \approx 32,76927 \text{ kHz} \cdot \frac{1}{8} \approx 4,096159 \text{ kHz}$$

XT2CLK Analog zur LFX1CLK-Taktquelle, können wir die maximale Frequenz aus Aufgabe 2.1.1 nehmen, die minimale Frequenz wiederum durch den Takteiler mit Divisor 8 erreichen. Der Code dazu ist ebenfalls analog, wird darum nicht wiederholt.

Also ergibt sich für die maximale Frequenz $f_{max} \approx 7,373165$ MHz.

Die Messung des minimalen Taktes ergibt eine Frequenz von 921,6456 kHz.

Auch dies stimmt mit dem rechnerischen Ergebnis überein:

$$f_{min} = f_{max} \cdot \frac{1}{8} \approx 7,373165 \text{ MHz} \cdot \frac{1}{8} \approx 921,6456 \text{ kHz}$$

DCOCLK Um die minimale und maximale Frequenz der DCOCLK-Taktquelle zu messen, binden wir in der `main.c` wieder die DCO-Quelle ein (Funktionsaufruf `DCO()`). Bei dieser Taktquelle können wir zusätzlich mit Hilfe des DCOR-Bits die Frequenz manipulieren. Dies kann ebenfalls im BCSCCTL2-Register gesetzt werden. Die erste Messung wurde mit `DCOR = 0` durchgeführt:

```
void aufgabe212() {
    P5SEL |= (1 << 4); // Leitung mit Modul verbinden
    P5DIR |= (1 << 4); // Als Ausgang fungieren
    BCSCCTL2 = (BCSCCTL2 & ~SELM_3) | SELM_0; // DCO
    BCSCCTL2 = (BCSCCTL2 & ~DIVM_3) | DIVM_0; // Divisor 1
    BCSCCTL2 &= ~DCOR; // DCOR auf 0
}
```

Die Messung ergibt einen Takt von $f_{DCOR=0} \approx 1,7026$ MHz. Mit gesetztem DCOR-Bit ergibt die Messung $f_{DCOR=1} \approx 7,3684$ MHz, was also die maximale Frequenz f_{max} ist.

Für die Messung der minimalen Frequenz wird der Takteiler wie in den vorigen Aufgaben genutzt, deshalb wird der Code nicht gezeigt. Die Messung für die minimale Frequenz ergibt einen Takt von ca. 212,8 kHz. Dies bestätigt das rechnerische Ergebnis:

$$f_{min} = f_{DCOR=0} \cdot \frac{1}{8} \approx 1,7026 \text{ MHz} \cdot \frac{1}{8} = 212,825 \text{ kHz}$$

A 2.1.3 An P2.5 ist ein Oszillatorwiderstand R_{osc} von 39kOhm angeschlossen. Erläutern Sie, wie der externe Widerstand für den DCOCLK-Taktgenerator nutzbar gemacht wird.

A 2.1.4 Welchen Einfluss hat der Widerstand auf den DCOCLK-Taktgenerator?

A 2.2 Stromverbrauch AB HIER REST NOCH ALT

A 1.2.1 Erläutern Sie unter Nutzung des User's Guide die Funktionalität der sieben 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 Bit für Bit 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 (Taster) als I/O-Input
9      // verwenden
10     P1SEL &= ~0x03;
11     P1DIR &= ~0x03;
12     // Letzten drei Pins von Port 4 (LEDs) als I/O-Output verwenden
13     P4SEL &= ~0x07;
14     P4DIR |= 0x07;
15
16     if (~((P1IN & Taster_rechts) ^ (P1IN & Taster_links))) {
17         // Beide Tasten bzw. keine von beiden
18         P4OUT &= ~gelb;
19     } else if ((P1IN & Taster_rechts) & ~(P1IN & Taster_lins)) {
20         // Rechte Taste
21         P4OUT &= ~gruen;
22     } else if (~(P1IN & Taster_rechts) & (P1IN & Taster_lins)) {
23         // Linke Taste
24         P4OUT &= ~rot;
25     }
26 }
```

A 2.3 Taktumschaltung

A 1.3.1 Nutzen Sie alle drei LED und den rechten Taster (P1.0), um eine Fußgängerampel zu programmieren.

```
1  #define Taster_rechts (0x01)
2  #define rot (0x01)
3  #define gelb (0x02)
4  #define gruen (0x04)
5
6  void aufgabe_1_3_1() {
7      // Letzten beiden Pins von Port 1 (Taster) als I/O-Input
        verwenden
8      P1SEL &= ~(Taster_rechts);
9      P1DIR &= ~(Taster_rechts);
10     // Letzten drei Pins von Port 4 (LEDs) als I/O-Output verwenden
11     P4SEL &= ~(rot+gelb+gruen);
12     P4DIR |= (rot+gelb+gruen);
13
14     // alle LEDs ausschalten
15     P4OUT |= (rot+gelb+gruen)
16
17     if (P1IN & Taster_rechts) {
18         // Gelbe LED an
19         P4OUT &= ~(gelb);
20         wait(3000);
21         // Gelb aus, rote LED an
22         P4OUT |= gelb;
23         P4OUT &= ~rot;
24         wait(3000);
25         // Zusaetzlich gelbe LED an
26         P4OUT &= ~gelb;
27         wait(3000);
28         // Rot und Gelb aus, gruen an
29         P4OUT |= (rot+gelb);
30         P4OUT &= ~gruen;
31         wait(3000);
32         // Gruen aus
33         P4OUT |= gruen;
34         wait(5000);
35     }
36 }
```

In diesem Programm werden einfach nacheinander die richtige LEDs an- bzw. ausgeschaltet, sodass wir eine Ampelablauf simulieren. Die waits verzögern dabei die Auswertung so, dass wir wahrnehmen können, in welcher Reihenfolge die LEDs an- und ausgeschaltet werden.

A 2.4 Codezeile

A 1.4.1 Entwickeln Sie einen Binärzähler

```
1  #define Taster_rechts (0x01)
2  #define Taster_links (0x02)
3  #define rot (0x01)
4  #define gelb (0x02)
5  #define gruen (0x04)
6
7  unsigned char counter = 0;
8
9  void aufgabe_1_4_1() {
10     // Letzten beiden Pins von Port 2 (Taster) als I/O-Input
        verwenden
11     P1SEL &= ~(Taster_rechts+Taster_links);
12     P1DIR &= ~(Taster_rechts+Taster_links);
13     // Letzten drei Pins von Port 4 (LEDs) als I/O-Output verwenden
14     P4SEL &= ~(rot+gelb+gruen);
15     P4DIR |= (rot+gelb+gruen);
16
17     if (P1IN & Taster_rechts) {
18         // Rechte Taste gedrueckt
19         if (counter < 7) {
20             // Mehr als 7 geht nicht
21             ++counter;
22         }
23     } else if (P1IN & Taster_links) {
24         // Rechte Taste gedrueckt
25         if (counter > 1) {
26             // Weniger als 1 geht nicht
27             --counter;
28         }
29     }
30     // Korrekte LEDs setzen:
31     // Wertigkeiten der LEDs sind gespiegelte
32     // Wertigkeit der Bits von counter
33     P4OUT &= ~(((counter & 0x01) << 2) + ((counter & 0x02) << 1) +
        (counter & 0x04))
34     wait(100);
35 }
```

Das Prellen der Kontakte bewirkt, dass mehr Phasenübergänge (Flanken) wahrgenommen werden, als tatsächlich vorkommen sollen. Dieses Problem kann man dadurch umgehen, dass man nach dem Verarbeiten eines Tastendrucks eine gewisse Zeit wartet um auf die Stabilisierung des Tasterpegels zu warten. Wir haben die Wartezeit experimentell ermittelt, sodass die Wartezeit die kleinste ist, bei der das Problem nicht mehr auftritt.