Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 3

Teilnehmer:

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

Gruppe: Freitag, Arbeitsplatz: HWP 1

A 3.1 Programmierung

A 3.1.1 Ermitteln Sie für den Watchdog alle möglichen Zeiten, die sich auf Basis der `ACLK`-Taktquelle mit dem `WDTCTL`-Register einstellen lassen.

Wie man dem Schaltbild entnehmen kann, führen nur die Bits `WDTIS1` und `WDTIS0` zu einer Veränderung der Watchdogzeiten (diese beiden Registerwerte dienen zum Multiplexen der Counter-Selektoren `Q6`, `Q9`, `Q13` bzw. `Q15`).

Dabei bedeutet der Counter-Selektor Qx, dass jeweils nur der Flip des (x+1)-niedrigwertigsten Bits vom Counter gezählt wird. Also können wir die Zeit bis zum Reset t_{reset} bei Selektion von `Qx` durch die Formel

$$t_{reset} = 2^{x+1} \cdot \frac{1}{\text{Takt}} \cdot \frac{1000}{2} = 2^{x+1} \cdot \frac{500}{\text{Takt}}$$

berechnen. Der Zähler des Terms $\frac{1000}{2}$ rechnet die Zeit in Millisekunden um, der Nenner teilt das Ergebnis durch zwei, da nur Flanken von 0 nach 1 gezählt werden.

Bei einem Takt von 32,768 kHz ergeben sich dann folgende Zeiten:

`WDTIS0`	`WDTIS1`	`Qx` selektiert	$t_{reset} = 2^{x+1} \cdot \frac{500}{32768\ Hz}$
0	0	`Q15`	1000 ms
0	1	`Q13`	250 ms
1	0	`Q9`	15,625 ms
1	1	`Q6`	ca. 1,953 ms

A 3.1.2 Wie verändern sich die in `A 3.1.1` bestimmten Zeiten wenn man mit den `DIVAx`-Bits im `BCSCTL1`-Register die `ACLK` Vorteiler auf 1, 2, 4, und 8 setzt?

Der Takt wird zum Zähler hin geteilt, also wird die Zeit bis zum Reset um den reziproken Wert vervielfacht.

A 3.1.3 Entwickeln Sie eine eigenständige Endlosschleife die in der Aufgabenstellung beschriebenen Schritte zyklisch ausführt.

Das nachfolgende Programm schaltet die LEDs in der vorgegebenen Form an bzw. aus.

```
1  // simuliert eine Sekunde Wartezeit
2  void waitOneSec() {
3    wait(50000); wait(50000);
4  }
5
6  void aufgabe313() {
7    //LEDs einrichten
8    P4DIR |= (0x07);
9    P4SEL &= ~(0x07);
10
11   // LEDs wie im vorgegebenen Schema
12   LED1OFF; LED2OFF; LED3OFF; waitOneSec();
13
14   LED1ON; waitOneSec(); LED1OFF;
15   LED2ON; waitOneSec(); LED2OFF;
16   LED3ON; waitOneSec(); LED3OFF;
17
18   LED1ON; LED2ON; LED3ON;
19   waitOneSec();
20   LED1OFF; LED2OFF; LED3OFF;
21 }
```

A 3.1.4 Im nächsten Schritt aktivieren Sie vor der Endlosschleife den Watchdog ohne weitere Einstellungen. Wie verhält sich das Programm jetzt? Erläutern Sie die gemachten Beobachtungen.

Der Aufruf in dem Hauptprogramm wird nun entsprechend geändert, sodass der Watchdof aktiviert wird. Dies können wir durch setzen des Passworts erreichen, da dabei das WDTHOLD-Register auf 0 gesetzt wird.

```
1  WDTCTL = WDTPW;
2  while(1) {aufgabe313();}
```

Beobachtung:
Die LEDs gehen alle an, das Modul führt direkt nach Aktivieren des Watchdogs ein RESET durch. Die Funktion aufgabe313 wird nicht ausgeführt (via Breakpoint getestet). Das kommt wohl daher, dass bereits durch das Initialisieren des Moduls der Zähler des watchdogs so groß wurde, sodass direkt nach dem aktivieren ein RESET-Befehl ausgelöst wird.

A 3.1.5 Nun wird der Watchdog wie in der Aufgabenstellung beschrieben aktiviert. Berechnen Sie für diese Einstellung die Zeit bis der Watchdog einen RESET auslöst. Wie lange dauert ein Durchlauf der Schleife mit den oben beschriebenen fünf Schritten? Zu welchem Zeitpunkt löst der Watchdog einen RESET aus? Testen das Programm und diskutieren Sie das erkennbare Verhalten.

Das Programm wird nun folgendermaßen angepasst:

```
1  BCSCTL1 |= DIVA0 + DIVA1;
2  WDTCTL = (WDTPW + WDTSSEL)
3  while(1) {aufgabe313();}
```

Da sowohl das WDTIS0-Bit und das WDTIS1-Bit nicht gesetzt sind, ergibt sich (nach Aufgabe 3.1.1) eine Zeit von einer Sekunde bis zum RESET. Da wir nun aber den Vorteiler 8 dazugeschaltet haben, sollte der Watchdog nach acht Sekunden ein RESET auslösen.
Die ungefähre Schleifendauer kann man durch Zählen der Warteoperationen abschätzen: Es wird fünf mal für eine Sekunde gewartet, also wird ein Schleifendurchlauf ca. fünf Sekunden dauern.
Ebenfalls durch Zählen können wir schätzen, dass der Watchdog im zweiten Schleifendurchlauf kurz vor dem Einschalten der gelben LED ein RESET auslösen sollte.
Nach Beobachtung des Ablaufs bestätigen sich unsere Überlegungen: Der Watchdog löst nach ca. acht Sekunden ein RESET aus.

A 3.1.6 Einen RESET durch den Watchdog innerhalb der Endlosschleife kann verhindert werden, wenn man innerhalb der Endlosschleife den Watchdog neu programmiert. Fügen Sie dazu einfach nach dem 5.Schritt eine entsprechende Codezeile ein. Testen und dokumentieren Sie das.

Da ein Schleifendurchlauf nur ca. fünf Sekunden dauert, können wir am Ende der Schleife eine Codezeile hinzufügen, die den Watchdog-Counter zurücksetzt. Dies können wir durch hinzufügen des folgenden Codes erreichen:

```
WDTCTL = (WDTPW + WDTCNTCL + WDTSSEL);
```

Mit dieser Codezeile wurde das Modul auch nach fünf Schleifendurchläufen nicht zurückgesetzt, also gehen wir davon aus, dass es funktioniert und das Modul nicht mehr neugestartet werden sollte.

A 3.2 Verteilung

A 3.2.1 Wie in der Aufgabenstellung erwähnt wird längere Arbeit durch den nachfolgenden Code simuliert:

```
while(P1IN & 0x01){
  P4OUT & =~0x01; wait(30000); P4OUT |= 0x01; wait(30000);
};
```

Erklären Sie was in der Codezeile ausgeführt wird. Welche Beobachtung machen Sie, wenn der Taster länger als 6 Sekunden gedrückt wird oder einfach dauerhaft gedrückt bleibt. Erklären Sie die Beobachtung.

In der Schleife wird zuerst die LED P4.0 eingeschaltet, dann 300 Millisekunden gewartet, die LED P4.0 wieder ausgeschaltet und wieder 300 Millisekunden gewartet.
Wird die Taste lang genug gedrückt (mindestens ca. 6 Sekunden), so wird durch den Watchdog ein `RESET` ausgelöst. Dies können wir uns damit erklären, dass durch die Schleifendurchläufe, die von dem Tastendruck ausgelöst werden, der Watchdog-Counter so weit hochgezählt wird, bis es schließlich zum `RESET` kommt. Ein Zurücksetzen des Zählers kann nicht stattfinden, da das Programm die Schleife nicht verlassen kann und somit die Zeile aus Aufgabe A.3.1.6 nicht erreicht.

A 3.2.2 Jetzt ändern Sie bitte die Codezeile so, dass während der Taster gedrückt ist, kein `RESET` durch den Watchdog ausgelöst wird. Erläutern Sie die Änderung.

Die Codeänderung schaltet den Watchdog zu Beginn der Tastendruck-Schleife einfach aus und schaltet ihn am Ende wieder an. Dies wird im nachfolgenden Code umgesetzt:

```
while(P1IN&0x01){
  WDTCTL = WDTPW + WDTHOLD;
  P4OUT &=~0x01; wait(30000);P4OUT |= 0x01; wait(30000);
  WDTCTL = (WDTPW + WDTSSEL);
};
```

A 3.2.3 In realen Systemen kann es durchaus passieren, dass aufgrund von Speicherfehlern oder Softwareproblemen ein System sich ständig neu startet. Wie kann man programmtechnisch registrieren und speichern, wann und an welcher Programmstelle der Watchdog das System neu gestartet hat. Skizzieren Sie einen Lösungsansatz. Als Hilfestellung hier folgende Stichwörter:

Ein System-Reset kann aus drei verschiedenen Gründen ausgelöst werden

1. Reset über den $\overline{RST}$/NMI Pin
2. ein Oszillator-Fehler
3. eine Zugriffsverletzung auf den Flash-Speicher

Es ist möglich programmatisch auf diese Ereignisse mit Hilfe einer Interrupt-Service-Routine vor dem Reset zu reagieren. Dazu wird die NMI-Interrupt-Service-Routine (Figure 2 – 6. NMI Interrupt Handler) verwendet, die für die einzelnen Ereignisse aktiviert werden kann, indem

1. das $\overline{RST}$/NMI Pin in NMI-Modus gesetzt wird
2. das ACCVIE Register gesetzt wird
3. das OFIE Register gesetzt wird

In der NMI-Interrupt-Service-Routine kann nun der gesamte Stack ab dem Stack Pointer (3.2.2) ausgelesen und in den Information-Memory-Teil des Flashspeichers gespeichter werden, damit kann die gesamte Aufruf-Hierarchie zurück verfolgt werden um den Programm-Ablauf zu rekonstruieren. Die ist möglich da auf dem Stack sämtliche Rücksprung-Adressen (jeweils über den Programm-Counter) gespeichert sind.

Eine Speicherung in den Information-Memory-Teil des Flashspeichers ist sinnvoll, da dieser Teil des Flash-Speichers nicht als Zwischenspeicher von Programmen genutzt wird und damit nicht sofort nach Neustart des Controllers neu beschrieben wird. Damit sind die darin gespeicherten Information nach einen Soft-Resets immer noch in Speicher verfügbar und können verarbeitet werden (zB. zur Fehleranalyse).

A 3.3 Software-Reset

A 3.3.1 Vervollständigen Sie das `MCU _ RESET;` Macro. Schreiben Sie ein kleines Programm, mit dem Sie die Funktion testen können. Das Programm soll bei Tastendruck einen RESET auslösen und über die LED ein Neustart des Controllers in geeigneter Weise signalisieren. Nutzen Sie dazu den Watchdog.

Wir können einen Neustart erzwingen, in dem wir ohne Passwort auf das Watchdogregister zugreifen, also lässt sich das Macro folgendermaßen schreiben:

```
#define MCU_RESET      (WDTCTL = 0x0000)
```

Unter Nutzung dieses Macros verwenden wir nun den Code aus den vorherigen Aufgaben um bei einem Tastendruck das Modul neuzustarten.

```
while(P1IN&0x01){
  P4OUT &=~0x01; wait(30000);P4OUT |= 0x01;
  MCU_RESET;
};
```

Dabei wird bei einem Tastendruck für eine kurze Zeit die LED4.0 angeschaltet. Sobald sie erlischt, wird ein Neustart ausgelöst.