Mikroprozessor Praktikum
Fr. HWP 2
Aufgabenblock 3

Paul Podlech 3910583
Max Wisniewski 4370074

6. März 2012

Block 3: Watchdog

Aufgabe 3.1: Programmierung

A 3.1.1

Ermitteln Sie auf Basis für den Watchdog alle mögliche Zeiten, die sich auf Basis der ACLK-Taktquelle (f=32768Hz) mit dem WDTCTL Register einstellen lassen. Fassen Sie die notwendigen Bitbelegung des WDTCL-Registers mit den dazu gehörigen Zeiten tabellarisch zusammen.

WDTIS0	WDTIS1	Selektiert	Takt
0	0	Q15	1000ms
0	1	Q13	250ms
1	0	Q9	15,625ms
1	1	Q6	1,953125ms

WDTSSEL ist in jedem Fall 1 gewesen, was der Taktquelle ACLK entspricht.

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?

Die Zeit wird vereinfacht, verdoppelt und verachtfacht. Da die Periodendauer sich, wie im letzten Aufgabeblock genau so verhält.

A 3.1.3

Entwickeln Sie eine eigenständige Endlosschleife die folgenden Ablauf von 5 Schritten zyklisch ausführt:

1. Schritt: alle LED (P4.0..2) aus , 1 Sekunden warten

2. Schritt: LED (P4.0) an, 1 Sekunden warten, LED (P4.0) aus

3. Schritt: LED (P4.1) an, 1 Sekunden warten, LED (P4.1) aus

4. Schritt: LED (P4.2) an, 1 Sekunden warten, LED (P4.2) aus

5. Schritt: alle LED (P4.0..2) an, 1 Sekunden warten, alle LED (P4.0..2) aus

Die erforderlichen Zeitverzögerungen realisieren Sie bitte mit der wait() Funktion (wait(50000) entspricht 0,5 Sekunde; beachten Sie den zulässigen Zahlenbereich des Parameters). Testen Sie das Programm.

Programm:
Das Programm ist wieder an unsere alten Programme angelehnt:

```
1  aufgabe313init(){
2    P4SEL &= ~(0x07);
3    P4DIR |= 0x07;
4    P4OUT |= 0x07;
5  }
6
7  aufgabe313(){
8    P4OUT |= (0x07);
9    wait(50000);  wait(50000);
10   P4OUT |= (0x07);
11   wait(50000);  wait(50000);
12   P4OUT &= ~(1<<0);
13   wait(50000);  wait(50000);
14   P4OUT |= (0x07);
15   wait(50000);  wait(50000);
```

```
16    P4OUT &= ~(1<<1);
17    wait(50000);  wait(50000);
18    P4OUT |= (0x07);
19    wait(50000);  wait(50000);
20    P4OUT &= ~(1<<2);
21    wait(50000);  wait(50000);
22    P4OUT |= (0x07);
23    wait(50000);wait(50000);
24    P4OUT &= ~(0x07);
25    wait(50000); wait(50000);
26  }
```

Dieses Programm haben wir in die Endlosschleife der Main eingebaut.

Beobachtung:
Alle zwei Sekunden geht eine LED an. Zwischen 2 LEDs ist je eine Sekunde keine an. Die LEDs werden alle drei ordentlich durchgegangen und am Ende gehen alle an. Danach fängt das ganze von vorne an.

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.

Beobachtung:
Wir haben unser *init313()* um die Aktivierung des Watchogs erweitert ($WDTCTL = WDTPW$). Damit wird HOLD auf 0 gesetzt. Wir brauchen nur das Passwort, damit die Änderung angenommen wird.

Wir beobachten, dass die LEDs alle anbleiben, auch wenn wir sie direckt vorher ausschalten. Wir schließen daraus, dass der Zähler des Watchdogs in der init() hochgezählt wurde und wenn wir den Watchdog anschalten, wird er sofort ein RESET feuern.

A 3.1.5

Jetzt wollen wir den Watchdog vor der Endlosschleife mit folgenden Einstellungen aktivieren:

- Watchdog auf Basis des ACLK-Takt mit dem Vorteiler 8 (dazu folgender C-Code: *BCSCTL1|=DIVA0 + DIVA1;*)
- Zählerstand 32768 für RESET

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.

Programm:

```
1   aufgabe314init(){
2     P4SEL &= ~(0x07);
3     P4DIR |= 0x07;
4     P4OUT |= 0x07;
5     BSCTL1 |= DIVA0 + DIVA1;
6     WDTCTL = WDTPW + WDTCNTCL + WDTSSEL;
7   }
```

Beobachtung: Wir haben den Taktdivisor auf 8 eingestellt und haben erwartet, dass wir nun etwas 8s warten müssen, bis ein RESET ausgelöst wird. Nach der Wartezeit, die wir zwischen den einzelnen LEDs eingestellt haben, stimmt die Zeit von 8 Sekunden auch. Nachdem der Watchdog angesprungen ist, wurde der Divisor mit in die Systemzeit übernommen. Das führt dazu, dass die LEDs nur kurz anspringen und sofort wieder Resetet werden, da die Wait Anweisung nun auf 8 Sekunden erweitert wurde.

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.

Beobachtung:
Nun läuft das Programm genaus so durch, wie schon bei Aufgabe *A.3.1.2.* Bei Aufgabe A 3.1.5 haben wir schon bestimmt, dass die Zeit ausreicht, dass die Schleife einmal durchlaufen kann. Daher soltle der Watchdog für diese einfache Anwendung immer ausreichend schnell zurück gesetzt werden.

Aufgabe 3.2: Verteilung

A 3.2.1

Stellen Sie sich vor nach dem 3. Schritt aus A3.1.3 und der erfolgten Änderung aus A3.1.6 wird das Programm gezwungen längere Zeit eine andere Aufgabe abzuarbeiten. Wir simulieren das durch folgende Codezeile:

- while(P1IN & 0x01)P4OUT & = 0x01; wait(30000); P4OUT |= 0x01; wait(30000);;

Solange der Taster an P1.0 gedrückt ist, wird die while Schleife abgearbeitet. 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.

Programm: Wir haben die Codezeile in das letzte Programm eingefügt.
Beobachtung:
Das Programm lässt die LEDs wie im alten Programm richtig leuchten. Kommen wir bei gedrückter Taste an die Schleife, so blinkt die rote LED. Drücken wir die Taste zu lange, springt der Watchdog an und das System wird reseted.

A 3.2.2

Jetzt ändern Sie bitte die folgende Codezeile so, dass während der Taster gedrückt ist, kein RESET durch den Watchdog ausgelöst wird.

- while(P1IN & 0x01)P4OUT & = 0x01; wait(30000); P4OUT |= 0x01; wait(30000);;

Erläutern Sie die Änderung. Bei größeren Softwareprojekten mit unterschiedlichen Laufzeiten in Abhängigkeit von Programmverzweigungen, Funktionsaufrufen und anderen Sachverhalten wird der Umgang mit dem Watchdog komplizierter.

Programmierung: Wir fügen in die while Schleife ein WatchdogReset ein (*WDTCTL = WDTPW + WDTCNTCL + WDTSSEL;*).
Beobachtung:
Nun können wir das Programm so lange in der Schleife lassen, wie wir wollen. Drücken wir den Taster zu lange, wird das Modul nicht reseted und wenn wir es loslassen, dann macht er mit dem normalen Programm weiter.

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.

Lösung:
Mit dem NMI Bit kann man dem Watchdog sagen, dass er nicht Reseten soll, sondern eine Benutzerspezifizierte ISR azuspringen. Der Pointer auf diese Routine kann in das WDTNMIES gespeichert

werden. Sind wir in der ISR können wir auf dem Stack ermitteln, von welcher Stelle im Code wir aufgerufen wurden, da der Pointer auf die Rücksprungeaddresse über den Callparametern liegt. Danach müssten wir diese Addresse persitent speichern, entweder indem wir es auf ein externes Modul schreiben oder auf den internen Flashspeicher laden. Nachdem wir alles gespeichert haben, würden wir das RESET Bit des Moduls per Hand setzen und das System damit neu starten.

Aufgabe 3.3: Software-Reset

A 3.3.1

Bei Auftreten der Codezeile

- MCU_RESET;

im Quellcode soll ein RESET des Controllers herbeigeführt werden.

Lösung: Wir müssten dafür ein Makro schreiben das den Systemreset ausführt. Dazu könnten wir zum Beispiel den Watchdog anschalten und sofort anspringen lassen. Glücklicherweise gibt es dieses Makro schon in System.h schon und muss nur noch aufgerufen werden.

A 3.3.2

Vervollständigen Sie dafür folgende Coldezeile

- # define MCU_RESET (...);

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 Neutstart in geeigneter Weise signalisieren. Nutzen SIe dazu den Watchdog.

Lösung:
Wir benutzen das Programm von Aufgabe A 3.2.3 und fügen an das Ende der Schleife vom Tastendruck MCU_RESET ein.
Beobachtung:
Es funktioniert.