Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

January 26, 2013

3 Watchdog

3.1 Programmierung

3.1.1 Ermitteln Sie (auf Basis des Blockschaltbildes der Watchdog-Einheit) für den Watchdog alle möglichen 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.

WDTISx Watchdog divider:

Belegung	Divider	Zeit
00	32768	1000 ms
01	8192	250 ms
10	512	15,265 ms
11	64	1,9 ms

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 Taktfrequenz wird geteilt und dadurch das Intervall multipliziert.

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**

```
1  #define BIT_SET(a,b) ((a) |= (b))
2  #define BIT_CLR(a,b) ((a) &= ~(b))
3  #define BIT_TOGGLE(a,b) ((a) ^= (b))
4  #define LED_ROT (0x01)   // 0 0 1
5  #define LED_GELB (0x02)  // 0 1 0
6  #define LED_GRUEN (0x04) // 1 0 0
7  #define LED_ALL (LED_ROT | LED_GELB | LED_GRUEN)
8  #define LED_ON(led) (BIT_CLR(P4OUT, led))
9  #define LED_OFF(led) (BIT_SET(P4OUT, led))
10 #define LED_TOGGLE(led) (BIT_TOGGLE(P4OUT, led))
```

```
11 #define TASTE_LINKS (0x1) #define TASTE_RECHTS (0x2)
12
13 #define SLEEP_QUANTUM 10000
14
15 /*
16  * NOPs um n Sekunden zu warten.
17  */
18 #define SLEEP(n) do { \
19 long time = n * 100000; /* wait() sleeps 10*n microseconds */ \
20                   while(time > SLEEP_QUANTUM) { \
21                   wait(SLEEP_QUANTUM); \
22                   time -= SLEEP_QUANTUM; \
23                                          } \
24                   wait(time); \
25                   } while(0)
26
27 // Setze P5.4 auf Input
28 BIT_SET(P5SEL, 1 << 4);
29
30 //===Hier die Endlosschleife quasi das Betriebssystem==================
31 while(1){
32          LED_OFF(LED_ALL);
33          SLEEP(1);
34          LED_ON(LED_ROT);
35          SLEEP(1);
36          LED_OFF(LED_ROT);
37          LED_ON(LED_GELB);
38          SLEEP(1);
39          LED_OFF(LED_GELB);
40          LED_ON(LED_GRUEN);
41          SLEEP(1);
42          LED_OFF(LED_GRUEN);
43          LED_ON(LED_ALL);
44          SLEEP(1);
45     }
```

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:

- es gehen keine LEDs an
- die Power On LED flackert

Erklärung: Der Watchdog führt immer wieder sehr schnell (durch ein sehr kurzes Timerintervall) einen RESET durch, was die Power On LED aus- und wieder anschaltet.

3.1.5 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 Sie das Programm und diskutieren Sie das erkennbare Verhalten.

- **Berechnung der RESET-Zeit:**

  $\frac{32768}{\frac{32768\,Hz}{8}} = \frac{32768 \cdot 8}{32768 \cdot \frac{1}{s}} = \underline{8\,s}$

  Es dauert 8 Sekunden, bis der RESET ausgelöst wird (= 1/8 Hz).

- **Beobachtung:**

  - Es dauert ca. 4 Sekunden bis der RESET ausgelöst wird.

- Ein Schleifendurchlauf dauert etwa 5 Sekunden.
- Beim 2. Durchlauf des Programms löst der Watchdog einen RESET aus. Danach läuft das LED-Programm nicht mehr, aber die Power-LED blinkt weiter, weil der Watchdog vor der Ausführung einen RESET auslöst.

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.

Folgende Codezeile wird am Ende der Schleife hinzugefügt:

```
1 ...
2 // Verwenden des Watchdog Passworts, Löschen des Counters, waehle ACLK
3 WDTCTL = WDTPW + WDTCNTCL + WDTSSEL;
4 ...
```

3.2 Verteilung

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:

```
1 ...
2 while(P1IN&0x01){P4OUT &=~0x01; wait(30000); P4OUT |= 0x01; wait(30000);};
3 ...
```

Die while-Schleife wird solange ausgeführt wie der linke Taster gedrückt ist. In der Schleife wird die rote LED zunächst angeschaltet und alle anderen LEDs ausgeschaltet, dann wird drei Sekunden gewartet. Danach wird die rote LED angeschaltet und wieder drei Sekunden gewartet.

Wenn der Taster länger als sechs Sekunden oder dauerhaft gedrückt wird, blinkt die rote LED im Takt von sechs Sekunden, jedoch wird dies durch den Watchdog RESET nach acht Sekunden unterbrochen.

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){
    WDTCTL = WDTPW + WDTHOLD; // Watchdog aus
    P4OUT &=~0x01;
     wait(30000);
    P4OUT |= 0x01;
     wait(30000);
    WDTCTL = WDTPW + WDTCNTCL + WDTSSEL; //Watchdog neu konfigurieren, starten
};
...
```

Der Watchdog wird so eingestellt, dass er vor der Ausführung der Programmcodes ausgeschaltet und danach wieder angeschaltet wird. Damit wird verhindert, dass ein RESET während der Ausführung des Programms ausgelöst wird.

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:

- NMI-Interrupt
- Stackpointer
- Programcounter
- Software bzw. System Reset
- Information memory

Anstatt der RESET-Funktion wird die NMI-Funktion benutzt, um die Kontrolle behalten zu können. Dabei werden eigene Funktionen für die RSI-Interupts verwendet. Einer ISR stehen PC und SR zur Verfügung. Im Flash-Speicher gibt es einen Bereich (Information Memory 256-Byte), in dem PC und SR Werte gespeichert werden, wenn der Interupt ausgelöst wird. Danach wird ein Reset von uns ausgeführt.

3.3 Software-Reset

3.3.1 Bei Auftreten der Codezeile

```
...
MCU_RESET;
...
```

im Quellcode soll ein RESET des Controllers herbeigeführt werden. Vervollständigen Sie dafür folgende Codezeile

```
...
#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 Neustart des Controllers in geeigneter Weise signalisieren. Nutzen Sie dazu den Watchdog.

```
...
// loest einen Power-Up Clear aus, da kein WDTPW verwendet wird
#define MCU_RESET (WDTCTL = 0)

// verbinde Taster mit P1
BIT_SET(P1SEL, TASTE_LINKS | TASTE_RECHTS);
// schlate alle LEDs aus
LED_OFF(LED_ALL);

int main()
{
    while (1) {
        // wenn eine Taste gedrückt wird
        if (P1IN & (TASTE_LINKS | TASTE_RECHTS)) {
            LED_ON(LED_ALL); // alle LEDs an
            wait(30000);
            MCU_RESET; // ein Reset des Controllers wird durchgeführt
        }
    }
}
...
```