Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

January 24, 2013

1 Watchdog

1.1 Programmierung

1.1.1 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:

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

1.1.2 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?

Da wird die Taktfrequenz geteilt, daraus wird das Intervall multipliziert.

1.1.3 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 #define SLEEP_QUANTUM 10000 #define SLEEP(n) do { \
13 long time = n * 100000; /* wait() sleeps 10*n microseconds */ \
14                     while(time > SLEEP_QUANTUM) { \
15                     wait(SLEEP_QUANTUM); \
16                     time -= SLEEP_QUANTUM; \
17                                             } \
18                     wait(time); \
19                     } while(0)
20 print_value();
21 // P5.4 Input
22 BIT_SET(P5SEL, 1 << 4);
23 //===Hier die Endlosschleife quasi das Betriebssystem==================
24 while(1){          LED_OFF(LED_ALL);          SLEEP(1);
25           LED_ON(LED_ROT);
26           SLEEP(1);
27           LED_OFF(LED_ROT);
28           LED_ON(LED_GELB);
29           SLEEP(1);
30           LED_OFF(LED_GELB);
31           LED_ON(LED_GRUEN);
32           SLEEP(1);
33           LED_OFF(LED_GRUEN);
34           LED_ON(LED_ALL);
35           SLEEP(1);     }
```

1.1.4 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: keine LEDs gehen an, nur power On LED
Erklärung: Watchdog hat Reset sehr schnell durchgeführt und das Timer-Intervall ist zu klein.

1.1.5 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 das Programm und diskutieren Sie das erkennbare Verhalten.

- Berechnung: Es dauert 8 Sekunden, bis Reset auslöst (= 1/8 Hz).
- Beobachtung: Es dauert ca. 4 Sekunden, bis Reset auslöst.
- Ein Durchlauf der Schleife dauert 5 Sekunden.
- Beim 2. Durchlauf des Programms wird ein Reset von Watchdog ausgelöst.
- Danach läuft das LED-Programm nicht mehr, aber die Power-LED blinkt weiter.

1.1.6 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.

Dieser Zeilencode wird am Ende der Schleife hinzugefügt:

```
1 ...
2 WDTCTL = WDTPW + WDTCNTCL + WDTSSEL;
3 ...
```

1.2 Verteilung

1.2.1 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 linke Taster gedrückt ist, blinkt die rote LED wait(30000) lang, dann fängt alles von vorne an.

1.2.2 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 an
};
...
```

Der Watchdog wird so eingestellt, dass er vor der Ausführung der Programmcodes ausgeschaltet und danach wieder angeschaltet wird. Damit wird verhindert, dass kein 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

Lösung:
Anstatt Reset-Funktion wird die NMI-Funktion benutzt um 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), indem PC und SR Werte gespeichert werden, wenn der Interupt ausgelöst wird. Danach wird ein Reset von uns ausgeführt.

1.3 Software-Reset

1.3.1 Bei Auftreten der Codezeile

```
...
int main()
{ #define MCU_RESET (WDTCTL = 0)
    BIT_SET(P1SEL, TASTE_LINKS | TASTE_RECHTS);
    LED_OFF(LED_ALL);
    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
        }
    }
}
...
```