Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 6

Teilnehmer:

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

Gruppe: Freitag, Arbeitsplatz: HWP 1

A 6.1 Zeitbasis - der Sekunden Interrupt

A 6.1.1 Es soll mit Hilfe eines Timer-Interrupts jede Sekunde der Zustand der LED (P4.0) getoggelt werden.

Für diese Aufgabe ist beinhaltet die Funktion aufgabe611 keine Anweisung, sodass der Inhalt der while(1)-Schleife in der main leer effektiv leer ist. Die Funktion init611, die alle notwendigen Register korrekt setzt, wird vor der main-Funktion aufgerufen und sieht folgendermaßen aus:

```
void init611() {
  // Alle LEDs aus
  LED1OFF;LED2OFF;LED3OFF;LED0FF;

    // LED P4.0 vorbereiten
    P4DIR |= (0x01);
    P4SEL &= ~(0x01);
    P4OUT |= 0x01;

    // Timer setzen:
    // Taktanzahl fuer eine Sekunde, da 7,3728MHz
    TBCCR0 = (unsigned int)7372800;
    // Taktquelle ACLK ist Flag TBSSEL_1
    TBCTL &= ~(TBSSEL_3 + MC_3);//reset
    // Count-Mode ist MC_1
    TBCTL |= (TBSSEL_1 + MC_1);//set
    // Interrupt freigeben
    TBCCTL0 |= CCIE;
}

void aufgabe611() {}
```

Das Register TBCCR0 enthält die Anzahl der Takte, bis der Timer-Interrupt ausgelöst werden soll. Da das Modul bei einer Taktfrequenz von 7,3728 MHz läuft, werden 7372800 Takte pro Sekunde ausgelöst. Dies ist also der Wert für das Register TBCCR0.
In dem Register TBCTL werden durch die beiden im Code genannten Bitmasken die Taktquelle ACLK und der Count-Mode MC_1 ausgewählt. Dies entspricht dem Modus count up; hier wird von Null an bis zu dem in TBCCR0-Register eingestellten Vergleichswert hochgezählt.
Schlussendlich wird noch in dem Control-Register zu TBCCR0 der Interrupt eingeschaltet.

Sind diese Einstellungen umgesetzt, ist die eigentliche Funktionalität sehr simpel umzusetzen: In jedem Timer-Interrupt, soll die LED einfach getoggelt werden, was durch folgenden Code realisiert wird:

```
#pragma vector = TIMERB0_VECTOR
__interrupt void TIMER (void) {
  P4OUT ^= 0x01;//Toggle LED
}
```

A 6.1.2 RECHERCHE LPM

A 6.1.3 Wir verändern die Lösung von A 6.1.1 so, dass durch Tastendruck das Zeitintervall des Interrupts halbiert vzw. verdoppelt werden kann.

Um uns in der Aufgabe zu merken, bei welchem Divisor wir gerade sind, legen wir eine Variable divider613 an, die eine Zahl zwischen 0 und 3 (jeweils inklusive) ist. Dabei ist die Zahl 0 als Divisor 1 (voller Takt) und die Zahl 3 als Divisor 8

zu interpretieren. Dies wurde so gewählt, da so die Bitmuster der Zahl mit dem geforderten Bitmuster des ID-Eintrags (Input Divider) in TBCTL übereinstimmt. Wollen wir also den neuen Divisor setzen, so müssen wir die Zahl nur noch an die richtige Stelle verschieben und in das TBCTL-Register übernehmn. Dies erledigt das Makro DIV_BIT_FLAG; hier wird die Divisor-Zahl sechs Stellen nach Links verschoben, sodass die Bits sieben bis sechs von TBCTL korrekt getroffen werden.

```
unsigned int divider613 = 0x03u;
#define DIV_BIT_FLAG (divider613<<6)
```

Die Initialbelegung der Register erfolg analog zu A 6.1.1. Der einzige Unterschied ist, dass initial ebenfalls der Divisor gesetzt wird, nämlich auf 8. Die Funktion aufgabe613 bleibt wieder leer.

```
void init613() {
  // Alle LEDs aus
  LED1OFF;LED2OFF;LED3OFF;LED0FF;

    // LED P4.0 vorbereiten
    P4DIR |= (0x01);
    P4SEL &= ~(0x01);
    P4OUT |= 0x01;

    // Taste 1: Input, IO-Funktion, Interrupt, Edge
    P1DIR &= ~(0x03);
    P1SEL &= ~(0x03);
    P1IE |= (0x03);
    P1IES &= ~(0x03);

    // Timer setzen:
    // Taktanzahl fuer eine Sekunde, da 7,3728MHz
    TBCCR0 = (unsigned int)7372800;
    // Taktquelle ACLK ist Flag TBSSEL_1
    TBCTL &= ~(TBSSEL_3 + MC_3  + ID_3);//reset
    // Count-Mode ist MC_1
    TBCTL |= (TBSSEL_1 + MC_1 + DIV_BIT_FLAG );//set
    // Interrupt freigeben
    TBCCTL0 |= CCIE;
}

void aufgabe613() {}
```

Um nun den Divisor mittels Taster steuern zu können, fügen wir einen Taster-Intterrupt ein. Dieser Überprüft bei einem Tastendrück, welche Taste gedrückt wurde und ob man durch das Ausführen der Verdopplung/Halbierung in dem zulässigen Bereich bleiben würde. Ist dies der Fall, wird die Variable divider613 entsprechend angepasst und am Ende gesetzt.

```
#pragma vector = PORT1_VECTOR
__interrupt void PORT1 (void) {
  CLEAR(P1IFG, 0xFF);
  // Taster gedrueckt
  if((P1IN & 0x01)) {
    // Taster P1.0 gedrueckt, verdoppeln
    if (divider613 < 3) {
      divider613++;
    }
  }
  if((P1IN & 0x02)) {
    // Taster P1.0 gedrueckt, halbieren
    if (divider613 > 0) {
      divider613--;
    }
  }
  // Divisor setzen
  TBCTL &= ~(ID_3);//reset
  TBCTL |= DIV_BIT_FLAG;
  wait(5000);
}
```

Der Timer-Interrupt entspricht exakt dem aus A 6.1.1.

A 6.2 Zeitmessung

A 6.2.1 Messung blagbla

```
#include "stdio.h"


unsigned int delta = 0u;
unsigned int sec;
void init621() {
  LED1OFF;LED2OFF;LED3OFF;LED0FF;

    // LEDS vorbereiten
    P4DIR |= (0x01);
    P4SEL &= ~(0x01);
    P4OUT |= 0x01;

    // Taste 1 input, IO, Interrupt, edge
  P1DIR &= ~(0x03);
    P1SEL &= ~(0x03);
    P1IE |= (0x03);
    P1IES &= ~(0x03);

    // Timer setzen
    //TBCCR0 = (unsigned int) 73728u; // GEHT NICHT IST ABER DOCH
        DAS GLEICHE???? (fast zumindest)
    TBCCR0 = (unsigned int) (7372800u & 0xFFFFu)/100u; //
        Taktanzahl fuer eine Sekunde, da 7,3728MHz
    // Taktquelle ACLK ist Flag TBSSEL_1, Countmethod ist MC_1
    TBCTL &= ~(TBSSEL_3 + MC_3 + ID_3);//reset
    TBCTL |= (TBSSEL_1 + MC_0 + ID_0 );//set
    TBCCTL0 |= CCIE ; // Interrupt freigeben
}

#pragma vector = PORT1_VECTOR
__interrupt void PORT1 (void) {
    char zeitausgabe[10];


  // Pin gedrueckt
  if((P1IN & 0x01)) {
    // Taster P1.0 gedrueckt, P4.1 an
    LED2ON;
    delta = 0u;
    TBCTL &= ~(MC_3);//reset
      TBCTL |= (MC_1);//set
  }
  if((P1IN & 0x02)) {
    // Taster P1.0 gedrueckt, P4.1 aus
    LED2OFF;
    TBCTL &= ~(MC_3);//reset
      TBCTL |= (MC_0);//set

      sec = (delta / 100u) % 100u;

      //zeitausgabe[9] = '\0';
      sprintf(zeitausgabe,"%02u.%02u␣sek", sec,(delta%100));

  }


  wait(5000);
  CLEAR(P1IFG, 0xFF);
}

#pragma vector = TIMERB0_VECTOR
__interrupt void TIMER (void) {
  LEDTOGGLE;
  ++delta;
}
```

```
void aufgabe621() {
  //skip
}
```

A 6.3 Zeitschaltwerk

A 6.4 LED als Indikator

A 6.5 PWM - Pulsweitenmodulation