Inhaltsverzeichnis

1 Praktikum

mikrocontroller.net für weitere Infos (Schaltpläne lesen)

1.1 Umgang mit der Software(KEIL)

erst übersetzen(F7), dann runterladen(F8)

1.2 Aufgabe 2

- in der PDF von chip1768 den Schaltplan aufschlagen und in die GPIOs sehen. In GPIO(1) finden sich die LEDs. Dort sind die Ports mit $P[x]$ dargestellt, wobei x das Bit für den jeweiligen Port darstellt.
- Über #DEFINE die LEDx mit der zugehörigen Bitkonstanten verkoppeln
- mit FIODIR die Ports auf Eingang setzen
  ($LPC_GPIO1 -> FIODIR| = 1 << LEDx$)
- mit FIOSET Strom auf den Port geben (=1)
- mit FIOCLR Strom auf den Port abstellen (=1)
- FIOPIN gibt den Status der LED an, somit kann auch direkt auf die LED zugegriffen werden

1.3 Aufgabe 4

- Achtung, LEDs 4 und 5 gehen nur auf LOW an!

1.4 code

```
#include "LPC17xx.h"

#define LED2 20
#define LED3 21
#define SPEAKER 5
#define LED4 8
#define LED5 9

void delay(int ms)
{
        int y = 0;
        int z = 0;
        while (z < ms)
        {
```

```
                z++;
                y=0;
                while (y<20000)
                {
                        y++;
                }
        }
}

void testBedLedBlink(int led)
{
        delay(200);
        LPC_GPIO0->FIOCLR |= 1<<led;
        delay(200);
        LPC_GPIO0->FIOSET |= 1<<led;
}

int main()
{

        //Initialisierungen
        LPC_GPIO1->FIODIR |= 1<<LED2;
        LPC_GPIO1->FIODIR |= 1<<LED3;
        LPC_GPIO0->FIODIR |= 1<<LED4;
        LPC_GPIO0->FIODIR |= 1<<LED5;
        LPC_GPIO2->FIODIR |= 1<<SPEAKER;

        while(1)
        {
                /*         Aufgabe 1
                LPC_GPIO0->FIOSET |= 1<<LED4;
                delay(100);
                LPC_GPIO0->FIOCLR |= 1<<LED4;
                delay(100);
                /**/
                /*        Aufgabe 2
                LPC_GPIO1->FIOCLR |= 1<<LED3;
                LPC_GPIO1->FIOSET |= 1<<LED2;
                delay(4000);
                LPC_GPIO1->FIOCLR |= 1<<LED2;
                LPC_GPIO1->FIOSET |= 1<<LED3;
                delay(4000);
                /**/
                /*         Aufgabe 3
```

```
                LPC_GPIO2->FIOSET |= 1<<SPEAKER;
                delay(1);
                LPC_GPIO2->FIOCLR |= 1<<SPEAKER;
                delay(1);
                /**/
                /*        Aufgabe 4
                testBedLedBlink(LED4);
                testBedLedBlink(LED5);
                /**/
        }
}
```

2 Praktikum

2.1 TASTENDRÜCKEN

Button abrufen

- $if(LPC\_GPIO0->FIOPIN\&(1<<BT1)==0)$
- BT1 = auf die Binärstelle des Button 1 gestellt
- Wenn FIOPIN an der Stelle BT1 "gedrücktïst, dann ist dort NNull"gesetzt
- Wenn FIOPIN an der Stelle BT1 nicht gedrücktïst, dann ist dort Ëins"gesetzt
- Bei der Und-Verknüpfung sollte an der Stelle von BT1 der Wert herauskommen der signalisiert ob der Button gedrückt ist

2.2 CODE

```
#include "LPC17xx.h"
#include <stdbool.h>
//Definitionszuweisungen------------------
#define LED1 18   //P1.18 LPC Modul high-aktiv
#define LED2 20   //P1.20 LPC Modul high-aktiv
#define LED3 21   //P1.21 LPC Modul high-aktiv
#define LED4 23   //P1.23 LPC Modul high-aktiv
#define BT1 6     //Button 1 an P0.6
#define BT2 16    //Button 2 an P0.16
#define SPK 5     //Speaker P2.5
#define LED4_BD 9   //LED4 mBed low-aktiv
#define LED5_BD 8   //LED5 mBed low-aktiv
//Funktionsdeklarationen------------------
```

```
int warte();
int outLED();   //Ausgabe Zählvariable auf LED Zeile LPC Modul

// Variablenzuweisungen-------------------
// globale variable
int izaehl=0;   //Zählvariable
//***Mainprogramm*******************************
int main()
{
 //Initialisierungen ---------------------
 LPC_GPIO1->FIODIR|=(1<<LED1);      //LED1 1-4 LPC Modul auf Portausgang
 LPC_GPIO1->FIODIR|=(1<<LED2);
 LPC_GPIO1->FIODIR|=(1<<LED3);
 LPC_GPIO1->FIODIR|=(1<<LED4);
 LPC_GPIO2->FIODIR|= 1<<SPK;

 bool toggle = 0;

 while(1)
 {
  /*
  if((LPC_GPIO0->FIOPIN & (1<<BT1)) == 0)
  {
   warte(50);
   if(toggle)
    toggle = 0;
   else
    toggle = 1;
   while((LPC_GPIO0->FIOPIN & (1<<BT1)) == 0);
  }
  if(toggle)
  {
   // Aufgabe 2
   LPC_GPIO1->FIOSET |= 1<<LED1;
   warte(500);
   LPC_GPIO1->FIOCLR |= 1<<LED1;
   warte(500);

   // Aufgabe 3
   LPC_GPIO2->FIOSET |= 1<<SPK;
   warte(1);
   LPC_GPIO2->FIOCLR |= 1<<SPK;
   warte(1);
  }
```

```
  */
  if((LPC_GPIO0->FIOPIN & (1<<BT1)) == 0)
   LPC_GPIO1->FIOSET |= 1<<LED1;
  if((LPC_GPIO0->FIOPIN & (1<<BT2)) == 0)
   LPC_GPIO1->FIOSET |= 1<<LED4;
  warte(500);
  LPC_GPIO1->FIOCLR |= 1<<LED1;
  LPC_GPIO1->FIOCLR |= 1<<LED4;
  warte(500);
 }
}

//***Funktionen*********************************
//Zeitschleife in Millisekunden
//------------------------------
int warte(int izahl)   //izahl in Millisekunden
{
 int iz=0,iy=0;
 while (iz<izahl)
      {
      iz++;
      iy=0;
      while (iy<20000)
            {
         iy++;
            }
      } // ende while
}
```

3 Praktikum

3.1 TIMER EINSTELLEN

- Timer zählt kontinuierlich hoch (max. 2^{32})
- Es wird ein Match-Register"gesetzt, dass festsetzt bis wie weit der Timer zählt
- Wenn der Timer mit dem Match-Register matched, wird ein Ereignis ausgelöst (z.B. ein Interrupt, Timer zurücksetzen Anhalten) -> Match-Control Register (MRxI/MRxR/MRxS x - Matchregister Nummer #)
- Der Timer fängt daraufhin wieder von vorne an zu zählen
- die Hälfte der Frequenz gibt an, wie lange der Timer braucht um einmal bis zum Maximum hochzulaufen (Speed)

- Formeln (siehe PR_DOKU Seite 19)
- PCLK gibt für jeden Takt der CPU einen Impuls
    1. PCLK modifiziert werden (jede 2./4./8.)
- Jeder Timer hat eigene 4 Matchregister
- Es gibt 4 Timer
- PCLKSELx gibt den Teiler für die PCLK an

00	/1.	)
01	/2.	
10	/4. (Standard)	
11	/8.	

3.2 ABLAUF

1. Timer initialisieren
    - Timer 0 einschalten
        - $LPC\_SC->PCONP| = 1 << 1$
    - Clock-Teiler einstellen
        - $LPC\_SC->PCLKSEL0| = 3 << 2$
    - Matchregister einstellen
        - $cputakt = 100mhz$
        - $100mhz/8 = 12,5$
        - $MR = 12,5Mhz/2 * 1hz$
        - $MR = 6250000$
        - $LPC\_TIM0->MR0 = 6250000$
    - Einstellen was das Matchregister tun soll
        - item Interrupt soll ausgelöst werden
        - $LPC\_TIM0->MCR| = 3 << 0$
2. Interrupt Service Routine Schreiben
    - z.B. void $TIMER0\_IRQHandler()$
    - name ist vorgegeben und muss zwingend verwendet werden
    - $system\_LPC17xx.s$ definiert die Namen der ISR-Funktionen (External Interrupts)
        - enthält die IVT (Interrupt Vektor Table)
        - Timer 0 meldet IMMER an Port 17
        - Ist über $system\_LPC17xx.s$ nachvollziehbar

- !!!Interrupt Flags zurücksetzen!!!
  - sonst bleibt der Interrupt bestehen und man würde nie die ISR verlassen
  - $LPC\_TIM0->IR| = 1 << 0$
    - Setzen um zu resetten
    - Interrupt Register um Bit-Positionen zu finden
- Nutzfunktion implementieren

3. Interrupt Service Routine Priorität setzen
   - $NVIC\_SetPriority(TIMER0\_IRQn, 15)$
4. Interrupt Service Routine aktivieren (sonst würde nie die Funktion abgefeuert werden)
   - $NVIC\_EnableIRQ(TIMER0\_IRQn)$
5. Timer aktivieren
   - $LPC\_TIM0->TCR| = 1 << 0$

3.3 CODE