In diesem Versuch wird ein 8051-Entwicklungsboard mit dem EFM8BB3 Prozessor von Silicon Labs mit diverser Peripherie vorgestellt.

Eigenschaften des EFM8BB3:

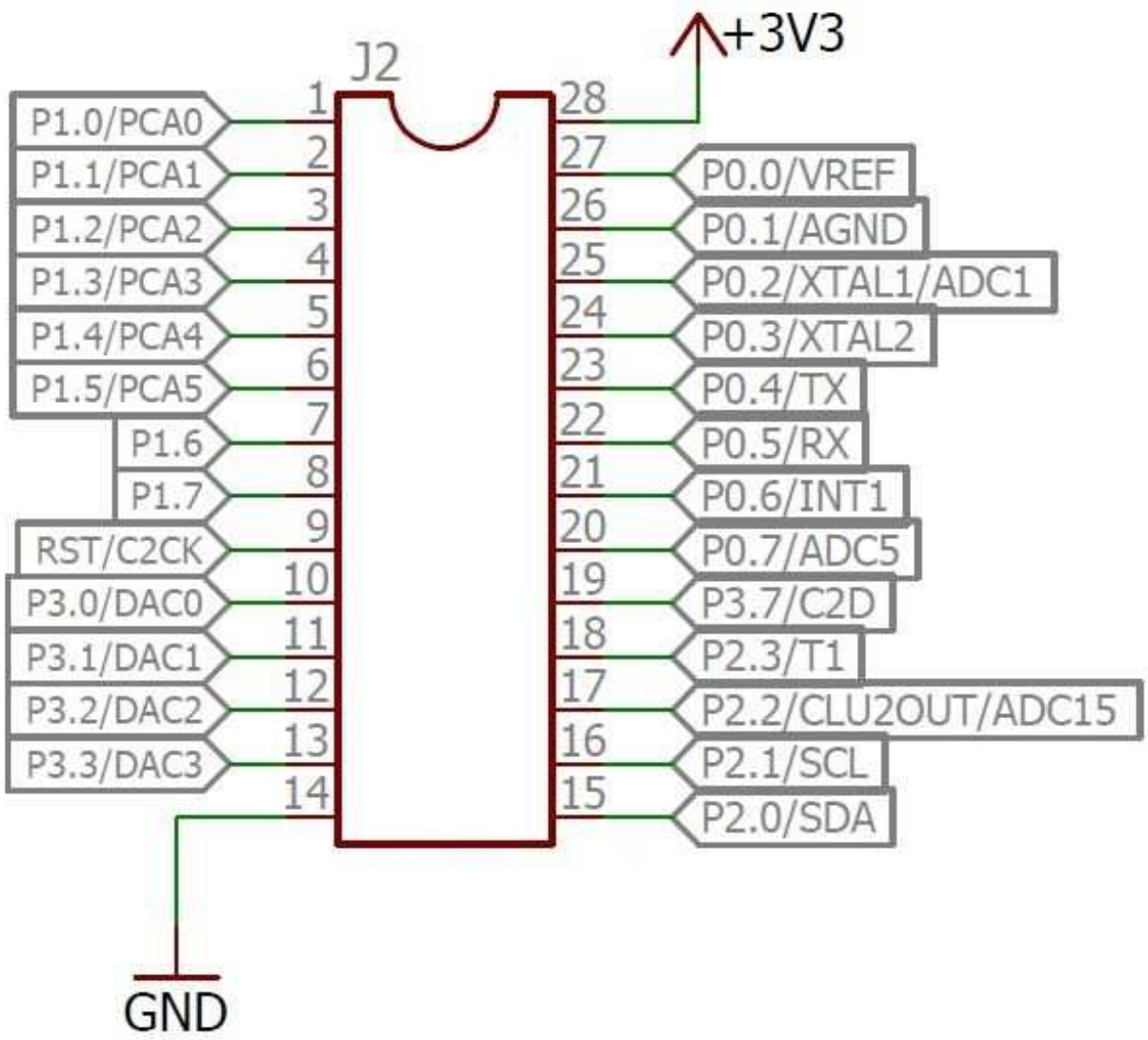
- 16KB Flash, 2.25KB RAM
- 50 MHz interner Oszillator (CLK)
- 2 UART, SPI, I2C
- 2 Analog Comparator / ADC / DAC

1. Einstieg in die 8051 Programmierung

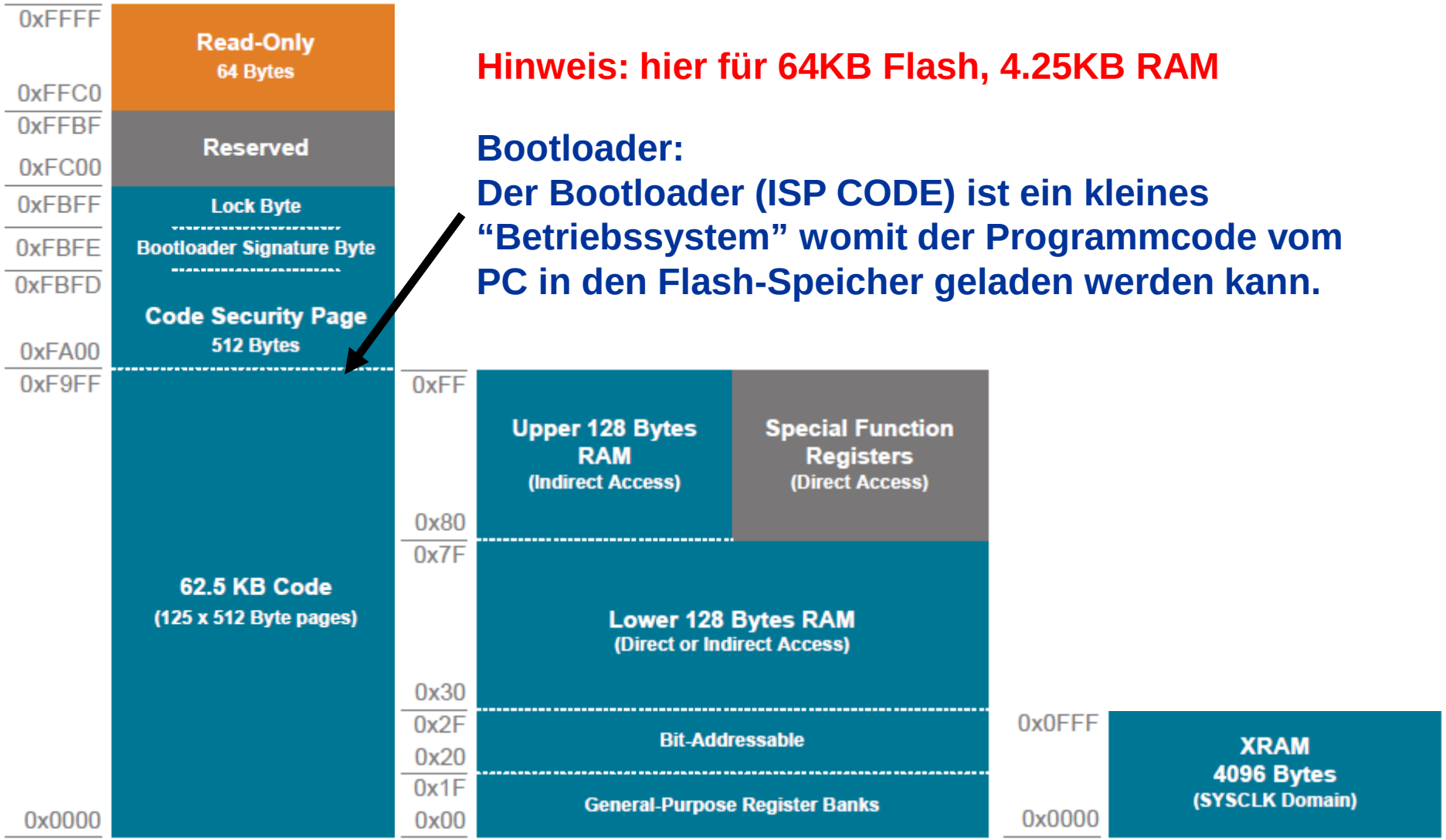
a) Vergleichen Sie die Eigenschaften von EFM8BB3 und EFM8BB1

b) Verschaffen Sie sich einen Überblick über den EFM8BB3

8051 Entwicklungsboard • Pinbelegung



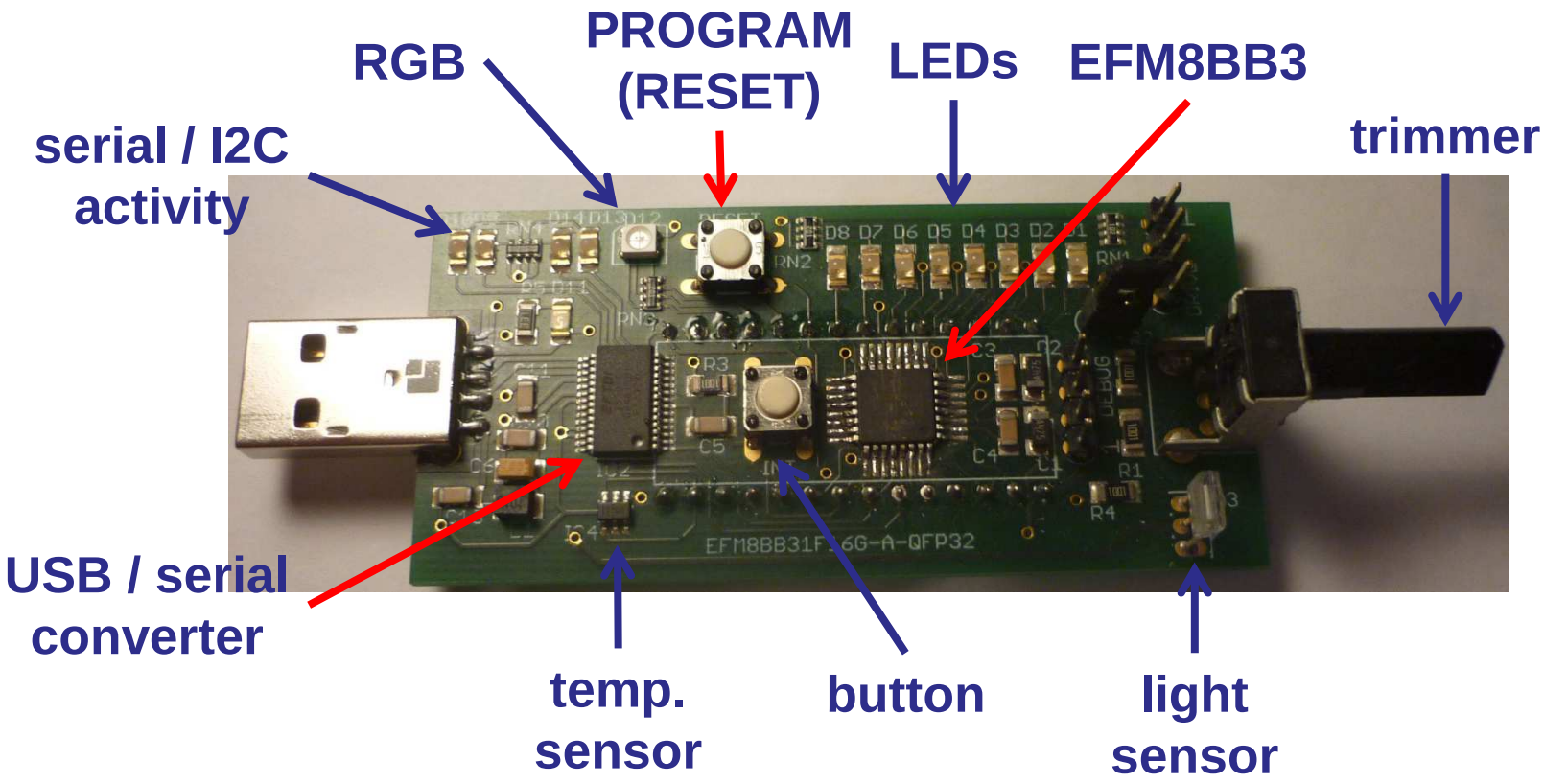
Speicherorganisation des EFM8BB3



Hinweis: hier für 64KB Flash, 4.25KB RAM

Bootloader:
Der Bootloader (ISP CODE) ist ein kleines "Betriebssystem" womit der Programmcode vom PC in den Flash-Speicher geladen werden kann.

c) Verschaffen Sie sich einen Überblick über das Entwicklungsboard:



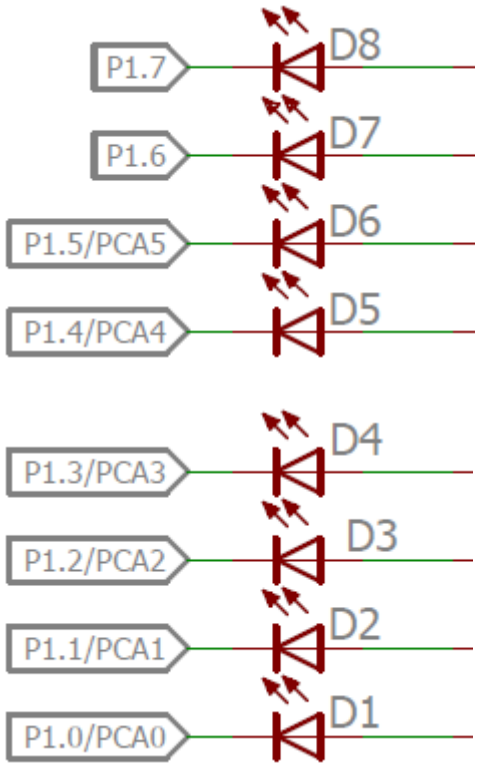
Problem:

Blinkende LED

Lösung:

Die LEDs sind am Port 1 über Pull-Up-Widerstände angeschlossen.

Hinweis: Anders als beim 8051 verfügen beim EFM8BB3 alle Ports über programmierbare Pull-Widerstände so dass auf externe Pull-Widerstände auch verzichtet werden könnte.



d) Öffnen Sie das Projekt 1a_LED_ASM\LED.uvproj und betrachten Sie die Assembler-Datei LED.asm. Erzeugen Sie die HEX-Datei.

```
ORG 0

main:
     ORL       XBR2, #01000000B          ; port power
loop:
     CPL       P1.0                      ; toggle LED

     MOV       R0, #0FFH
     CALL      delay
     JMP       loop

delay:
     MOV       R1, #10
loop1:
     MOV       R2, #0FFH
loop2:
     DJNZ      R2, loop2
     DJNZ      R1, loop1
     DJNZ      R0, delay
     RET
```

Blinkdauer

Pin Switch Matrix / Cross-Bar

11.4.3 XBR2: Port I/O Crossbar 2

enable

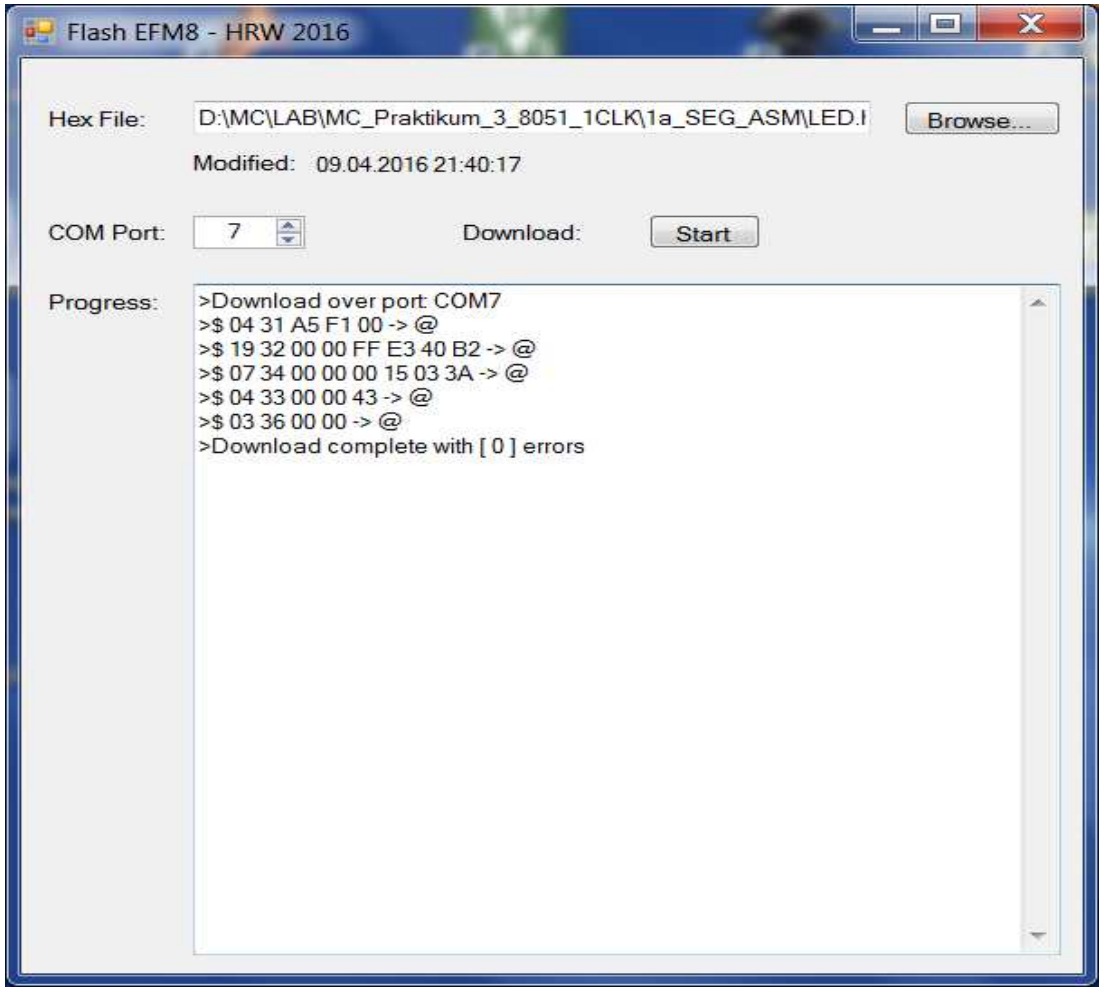
Bit	7	6
Name	WEAKPUD	XBARE
Access	RW	RW
Reset	0	0

Flash-Programmierung

Zum Betrieb des UART-Bootloaders bzw. für den Download der HEX-Datei wird das Tool FlashEFM8 genutzt.

e) Öffnen Sie FlashEFM8 mit den abgebildeten Einstellungen.

Hinweis:
Durch die Verwendung des USB-seriell Wandlers kann sich eine andere COM Port Nummer ergeben.



- Drücken Sie auf dem 8051 Entwicklungsboard den RESET-Taster
- Nun ist der Bootloader aktiv: Laden Sie die HEX-Datei

Jetzt sollte eine LED blinken.

Hinweis: Für den Start des Bootloaders muss auf dem 8051 Entwicklungsboard der Jumper gesteckt sein.

f) Öffnen Sie das Projekt 1c_LED_C\LED.uvproj und betrachten Sie die C-Datei LED.c. Erzeugen und Flashen Sie die HEX-Datei.

```
// default CLK = 24.5 MHz / 8 = 3 MHz
void delay(unsigned int cnt)
{
  while (--cnt);
     while (--cnt);
     while (--cnt);
     while (--cnt);
}

sbit LED = P1^0;

void main(void)
{
     XBR2 |= (1 << (6)); // port power
     while (1)
  {
          LED ^= 1;      // toggle LED
          delay(200);
  }
}
```

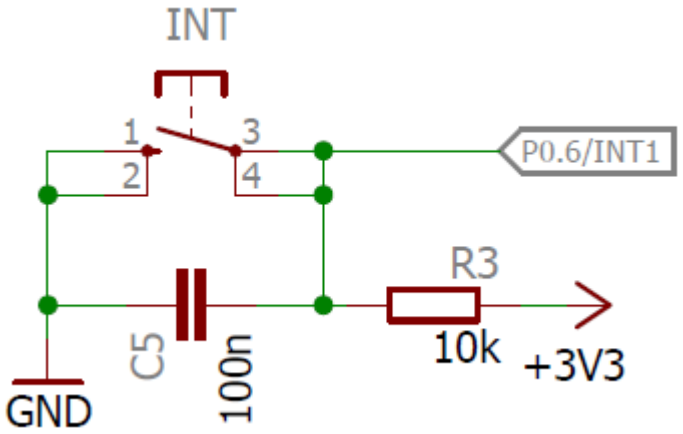
Default System Clock CLK = 3 MHz

Blinkdauer

g) Wie ist das C-Programm abzuändern wenn zwei LEDs abwechselnd blinken sollen?

2. Externe Interrupts

a) Öffnen Sie das Projekt 2a_BUT_INT_ASM\LED.uvproj und betrachten Sie die Assembler-Datei LED.asm:




```
ORG 0
     JMP       main

; BUT on INT1=P0.6
ORG 013H
     CPL       P1.0 ; toggle LED
     RETI

main:
     MOV       XBR2, #01000000B    ; port power
     MOV       WDTCN, #0DEH        ; disable watchdog passcode
     MOV       WDTCN, #0ADH
     ; EFM8 specific irq configuration
     MOV       IT01CF, #01100000B  ; IRQ assign 6:4 INT1=P0.6 2:0 INT0 not assigned
     ; 8051 standard irq configuration
     MOV       TCON, #00000100B    ; enable INT1 edge
     MOV       IE, #10000100B      ; enable INT1 IRQ and enable global
     CLR       P1.0 ; LED on
loop:
     JMP       loop
```

b) Erklären Sie die Funktion des Programms anhand des Schaltplans und der Pinbelegung des EFM8BB3

c) Öffnen Sie das Projekt 2b_BUT_INT_C\LED.uvproj und betrachten Sie die C-Datei LED.c:

Watchdog:
Sicherheitsschaltung die einen Programm-Absturz feststellen kann. Muss bei Interrupt-Steuerung abgeschaltet werden.

```
sbit LED = P1^0;

// BUT on INT1=P0.6
void extint1(void) interrupt 2 using 3 {
     LED ^= 1;
}

void main (void) {
     XBR2 |= (1 << (6));                    // port power
     WDTCN = 0xDE; WDTCN = 0xAD;            // disable watchdog passcode

     // EFM8 specific irq configuration
     // IRQ assign 6:4 INT1=P0.6 2:0 INT0 not assigned
     IT01CF |= (6 << (4));

     // 8051 standard irq configuration
     TCON |= (1 <<(2));       // enable INT1 edge
     IE |= (1 << (2));        // enable INT1 IRQ
     IE |= (1 << (7));        // enable global

     LED = 0;
     while (1);
}
```

d) Welches Problem tritt auf wenn Sie die INT1 Edge Detection abschalten?

3. Speicherzugriff

Konstantes Array in Flash (ROM)

a) Öffnen Sie das Projekt 3a_ROM_ASM\LED.uvproj und betrachten Sie die Assembler-Datei LED.asm:

```
; Flash/ROM
ORG 0
     JMP       main
; BUT on INT1=P0.6
ORG 013H
     INC       R0
     INC       DPTR
     MOV       A, #0
     MOVC      A, @A+DPTR              ; n*n from table
     CPL       A
     MOV       P1, A                   ; output n*n to LEDs
     CJNE      R0, #9, return          ; n == 9 (max)
     MOV       DPTR, #table            ; table address
     MOV       R0, #-1
return:
     RETI
; Flash/ROM from address 0.5K
ORG 200H
main:
     MOV       XBR2, #01000000B        ; port power
```

```
        MOV       WDTCN, #0DEH             ; disable watchdog passcode
        MOV       WDTCN, #0ADH
        ; EFM8 specific irq configuration
        MOV       IT01CF, #01100000B       ; IRQ assign 6:4 INT1=P0.6 2:0 INT0 not
    assigned
        ; 8051 standard irq configuration
        MOV       TCON, #00000100B         ; enable INT1 edge
        MOV       IE, #10000100B           ; enable INT1 IRQ and enable global

        MOV       DPTR, #table   ; table address
        MOV       R0, #0
    loop:
        JMP       loop
    ; Flash/ROM from address 1.0K
    ORG 400H
    table:
        DB        0,1,4,9,16,25,36,49,64,81
```

b) Erklären Sie die Funktion des Programms.

c) Öffnen Sie das Projekt 3b_ROM_C\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
// Flash/ROM
code unsigned char table[] = {0,1,4,9,16,25,36,49,64,81};

signed char n = 0;
```

```
// BUT on INT1=P0.6
void extint1(void) interrupt 2 using 3 {
     P1 = ~table[++n];        // n*n from table and output to LEDs

     if (n==9)                // n == 9 (max)
          n = -1;
}
void main(void) {
     XBR2 |= (1 << (6));              // port power
     WDTCN = 0xDE; WDTCN = 0xAD;      // disable watchdog passcode
     // EFM8 specific irq configuration
     // IRQ assign 6:4 INT1=P0.6 2:0 INT0 not assigned
     IT01CF |= (6 << (4));
     // 8051 standard irq configuration
     TCON |= (1 <<(2));       // enable INT1 edge
     IE |= (1 << (2));        // enable INT1 IRQ
     IE |= (1 << (7));        // enable global

     while (1);
}
```

Variables Array in RAM

d) Öffnen Sie das Projekt 3c_RAM_C\LED.uvproj und betrachten Sie die C-Datei LED.c

e) Erklären Sie die Ausgaben des Compilers für die Programme aus c) und d)

```
Program Size: data=18.0 xdata=0 code=235
Program Size: data=18.0 xdata=10 code=237
```

4. RGB LED

a) Öffnen Sie das Projekt 4a_RGB_ASM\LED.uvproj und betrachten Sie die Assembler-Datei LED.asm:

```
LED_R EQU P3.1 ; LED_R on P3.1
LED_G EQU P3.0 ; LED_G on P3.0
LED_B EQU P3.2 ; LED_B on P3.2

ORG 0
main:
     ORL       XBR2, #01000000B        ; port power

     // LEDs off
     SETB      LED_R
     SETB      LED_G
     SETB      LED_B
loop:
     ; run LEDs
     CLR       LED_R
     MOV       R0, #0FFH
     CALL      delay

     SETB      LED_R
     CLR       LED_G
     MOV       R0, #0FFH
     CALL      delay
```

RGB LED Pins

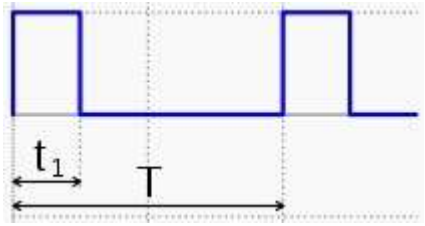
```
        SETB    LED_G
        CLR     LED_B
        MOV     R0, #0FFH
        CALL    delay

        SETB    LED_B
        JMP     loop

delay:
        MOV     R1, #10
loop1:
        MOV     R2, #0FFH
loop2:
        DJNZ    R2, loop2
        DJNZ    R1, loop1
        DJNZ    R0, delay
        RET
```

b) Erklären Sie die Funktion des Programms anhand des Schaltplans.

c) Erstellen Sie nun ein funktionsgleiches C Programm.
Hinweis: Musterlösung „4b_RGB_C“

d) Um mit der RGB LED verschiedene Farben und Helligkeiten anzeigen zu können wird PWM (Pulsweitenmodulation) genutzt.

Für eine Helligkeit von 25% wird beispielsweise ein PWM-Signal im Verhältnis t1:T von 1:4 erzeugt. Um ein sichtbares Flimmern zu vermeiden muss die Periodendauer T entsprechend kurz sein.



Öffnen Sie das Projekt 4c_RGB_PWM\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
#include <intrins.h>

#define LED_R       1          // LED_R on P3.1
#define LED_G       0          // LED_G on P3.0
#define LED_B       2          // LED_B on P3.2

bit toggle = 0;

// short delay
void delay(void) {
```

```
        _nop_();
        _nop_();
        _nop_();
        _nop_();
}

// BUT on INT1=P0.6
void extint1(void) interrupt 2 using 3 {
        toggle = ~toggle;
}

void init(void) {
        XBR2 |= (1 << (6));      // port power

        // EFM8 specific irq configuration
        // IRQ assign 6:4 INT1=P0.6 2:0 INT0 not assigned
        IT01CF |= (6 << (4));

        // 8051 standard irq configuration
        TCON |= (1 <<(2));       // enable INT1 edge
        IE |= (1 << (2));        // enable INT1 IRQ
        IE |= (1 << (7));        // enable global
}

void main(void) {
        init();
```

```
        // LEDs on : white
        P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));

        while (1) {
             // PWM 20%
             if (! toggle) {
                  // LEDs on : white
                  P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));
                  delay();
                  // LEDs off
                  P3 |= (1 << (LED_R)); P3 |= (1 << (LED_G)); P3 |= (1 << (LED_B));
                  delay(); delay(); delay(); delay();
             }
             // PWM 80%
             else {
                  // LEDs on : white
                  P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));
                  delay(); delay(); delay(); delay();
                  // LEDs (high)
                  P3 |= (1 << (LED_R)); P3 |= (1 << (LED_G)); P3 |= (1 << (LED_B));
                  delay();
             }
        }
   }
```

e) Erklären Sie die Funktion des Programms und skizzieren Sie das PWM-Signal. Messen Sie t1:T mit dem Oszilloskop.

Hinweis: Sie können auf der Rückseite des Entwicklungsboards z.B.P3.0 messen.

f) Öffnen Sie das Projekt 4d_RGB_TIMER_PWM\LED.uvproj und betrachten Sie die C-Datei LED.c. Messen Sie nun t1:T mit dem Oszilloskop.

```
…
unsigned int counter = 0;
…
void timer0(void) interrupt 1 using 3 {
     counter++;
}
void init(void) {
     …
     // timer 0
     TMOD |= (1 <<(1));       // timer 0 mode 2
     TH0 = -100;              // 100us

     // 8051 standard irq configuration
     TCON |= (1 <<(2));       // enable INT1 edge
     IE |= (1 << (1));        // enable timer 0 IRQ
     IE |= (1 << (2));        // enable INT1 IRQ
     IE |= (1 << (7));        // enable global
     TR0 = 1;                 // timer 0 start
}
void main(void) {
     init();

     // LEDs on : white
     P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));
```

```
while (1) {
     // PWM 20%
     if (! toggle) {
          if (counter < 10)        // 10*100us=1ms
          {
               // LEDs on : white
               P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));
          }
          if (counter >= 10) {
               // LEDs off
               P3 |= (1 << (LED_R)); P3 |= (1 << (LED_G)); P3 |= (1 << (LED_B));
          }
          if (counter >= 50)       // 50*100us=5ms
               counter = 0;        // counter reset
     }
     // PWM 80%
     else {
          if (counter < 40)        // 40*100us=4ms
          {
               // LEDs on : white
               P3 &= ~(1 << (LED_R)); P3 &= ~(1 << (LED_G)); P3 &= ~(1 << (LED_B));
          }
          if (counter >= 40) {
               // LEDs off
               P3 |= (1 << (LED_R)); P3 |= (1 << (LED_G)); P3 |= (1 << (LED_B));
          }
          if (counter >= 50)       // 50*100us=5ms
               counter = 0;        // counter reset
     }
}
```

5. UART

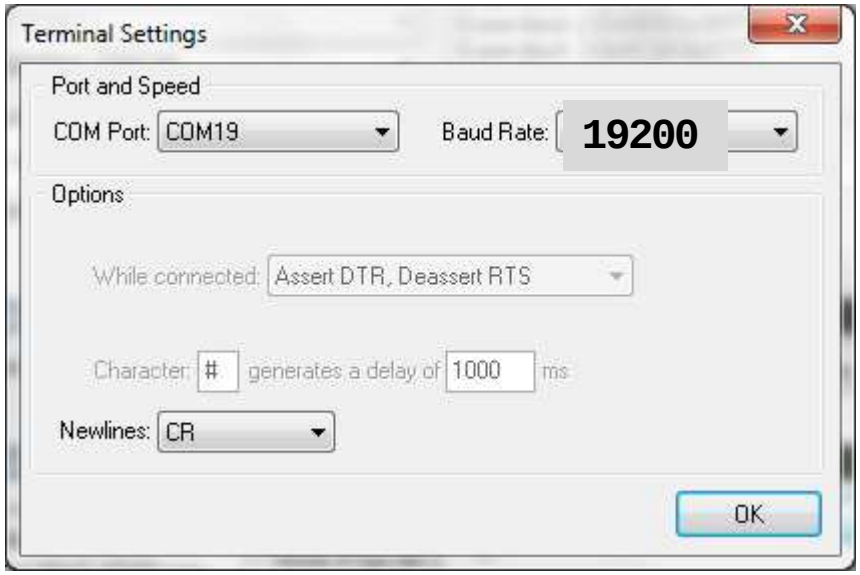
a) Öffnen Sie das Projekt 5a_UART\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
// serial send
void send(unsigned char ch) {
     SBUF = ch;
     while (!TI);   // wait until transmitted
     TI=0;
}
void init(void) {
     // SYSCLK = HFOSC0 = 24.5 MHz
     // SYSCLK = SYSCLK / 1
     CLKSEL = 0;
     // Timer 0/1 prescale SYSCLK / 4
     CKCON0 = 1;

     // UART mode %01 = 8-bit data
     SCON = 0x40;
     // Timer 1 mode 2
     TMOD |= 0x20;
     // Baud rate with prescale SYSCLK / 4
     TH1 = -160;    // 24.5 MHz / 4 / 160 = 38281 / 2 = 19141 (19200 Baud 0.3% error)
     TL1 = -160;
     // Timer 1 start
     TR1 = 1;
```

```
    XBR0 |= (1 << (0));                 // cross-bar enable UART0 pins
    XBR2 |= (1 << (6));                 // cross-bar enable all pins
    WDTCN = 0xDE; WDTCN = 0xAD;         // disable watchdog passcode
}

sbit LED = P1^0;
void main(void) {
    unsigned short counter = 0;
    char ch, buffer[] = "-----";       // 4 decimal digits + CR terminal new line
    buffer[4] = 0x0D;                   // CR terminal new line
    init();
    while (1) {
        unsigned char idx = 0;
        // toggle LED
        LED ^= 1;
        // counter 4 decimal digits
        if (counter++ > 9999)
            counter = 0;
        // convert to ASCII
        buffer[0] = 0x30 + counter/1000;
        buffer[1] = 0x30 + (counter%1000)/100;
        buffer[2] = 0x30 + (counter%100)/10;
        buffer[3] = 0x30 + counter%10;

        while ((ch = buffer[idx++]) != 0)
            send(ch);
        delay(200);
    }
}
```

b) Erklären Sie die Funktion des Programms.

c) Nutzen Sie das Flash Magic Terminal mit folgenden Einstellungen zur Anzeige der UART-Ausgabe:
-> Tools



Hinweis: Durch die Verwendung des USB-seriell Wandlers kann sich eine andere COM Port Nummer ergeben.

UART-Emulation in Software

Der EFM8BB3 verfügt über zwei UART in Hardware: UART0 und UART1. Falls noch mehr UART benötigt werden, ist es möglich, mittels eines weiteren Timer einen zusätzlichen UART in Software-Emulation zu realisieren.

d) Öffnen Sie das Projekt 5b_UART_SOFT\LED.uvproj und betrachten Sie die C-Datei LED.c :

```
…
#define Baud_Rate   4800
// TXD-SOFT on P0.4
sbit TXD_SOFT = P0^4;
#define LOW    TXD_SOFT = 0
#define HIGH   TXD_SOFT = 1
// SBUF_SOFT 10-bit : start-bit/8-bit/stop-bit
#define SBUF_SIZE       9
unsigned char SBUF_SOFT[SBUF_SIZE+1];
signed char SBUF_IDX = -1;

void timer2(void) interrupt 5 using 3 {
     // UART-SOFT transmit start
     if (SBUF_IDX != -1) {
          if (SBUF_SOFT[SBUF_IDX++])
               HIGH;
          else
               LOW;
```

← siehe Skript: UART

← Timer IRQ sendet Bits

```
      // transmit stop
           if (SBUF_IDX==SBUF_SIZE+1)
                SBUF_IDX = -1;
      }
      // Clear Timer 2 High Byte Overflow Flag
      TMR2CN0_TF2H = 0;
}
// TXD-SOFT
void send(char ch) {
  unsigned char idx;

  SBUF_SOFT[0] = 0;  // start-bit : 0
  for (idx=0; idx<8; idx++)
  {
    if (ch & (1<<(idx)))
      SBUF_SOFT[1+idx] = 1;
    else
      SBUF_SOFT[1+idx] = 0;
  }
  SBUF_SOFT[9] = 1;  // stop-bit : 1

  // transmit start
  SBUF_IDX = 0;
  // transmit time <1ms
  delay_short(500);
}
…
```

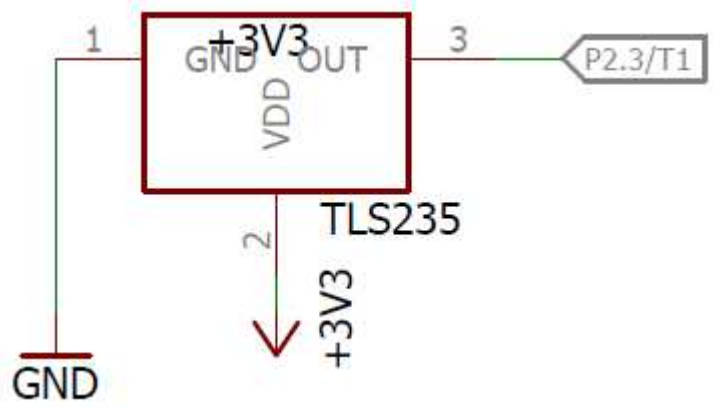
e) Erklären Sie die Funktion des Programms

Hinweis: Projekt 6a_LFU demonstriert manuelles Scanning

6. Licht-Frequenz-Umsetzer mit Timer/Counter

a) Öffnen Sie das Projekt 6b_LFU_COUNTER\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
bit toggle = 0;
unsigned short freq = 0;

void timer0(void) interrupt 1 using 2 {
     // reload timer 0 with 10 ms
     TH0 = -16;
     TL0 = 0x00;

     toggle = ~toggle;

     if (!toggle)
     {
          TR1 = 0;
          // stop count
          freq = (TH1 << 8) + TL1;
          TL1 = TH1 = 0; // clear timer 1
     }
     else
          TR1 = 1;  // next count
}
void init(void) {
     // timer0 mode1 and timer1 mode1 / counter : %0101|0001
     TMOD = 0x51;
```



```
        // skip to move to T1=P2.3
        SFRPAGE = 0x20;
        P0SKIP = 0xFF; // %1111|1111
        P1SKIP = 0xFF; // %1111|1111
        P2SKIP = 0x07; // %0000|0111
        SFRPAGE = 0x00;
        XBR1 |= (1 << (5)); // T1 routed to port pin

        // load timer 0 with 10 ms
        TH0 = -16;
        TL0 = 0x00;
        // TF0 = 0;              // timer 0 reset

        IE |= (1 << (1));        // enable timer 0 IRQ
        IE |= (1 << (7));        // enable global
        TR0 = 1;                 // timer 0 start
        XBR2 |= (1 << (6));      // port power
}
```

Pin Switch Matrix / Cross-Bar: Wie bei einem FPGA müssen Pins vor Nutzung mit der Peripherie verbunden werden.

b) Erklären Sie die Funktion des Programms und skizzieren Sie das LFU-Signal.

Hinweis: Timer 1 wird als Counter zur Flankenzählung genutzt, Timer 0 zum periodischen Update der Frequenzmessung.

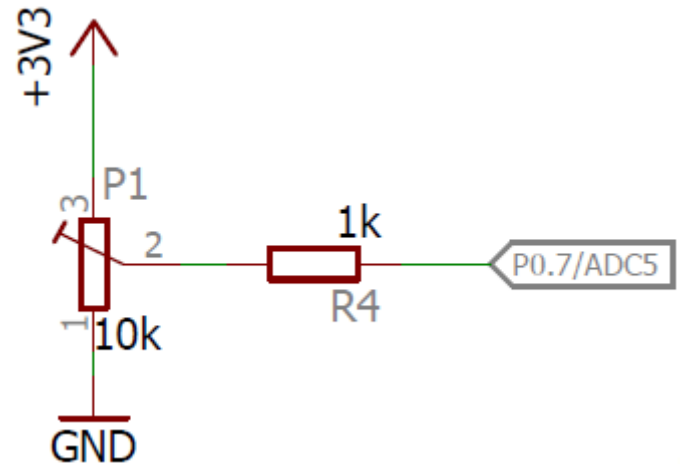
c) Messen Sie das LFU-Signal mit dem Oszilloskop.
Hinweis: Sie können auf der Rückseite des Entwicklungsboards P2.3 messen.

7. ADC: Analog-Digital-Wandler (12-Bit)

a) Öffnen Sie das Projekt 7a_TRIMMER_ADC\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
void init(void) {
     // SYSCLK = HFOSC0 = 24.5 MHz
     // SYSCLK = SYSCLK / 1
     CLKSEL = 0;

     // cross-bar enable all pins
     XBR2 |= (1 << (6));

     ADC0MX = 5;                        // P0.7/ADC5
     ADC0CN1 = (1 << (6));              // ADC0 12-bit
     ADC0CF0 = (0x1 << (3));            // SYSCLK / (1 + 1)
     ADC0CF2 = (1 << (5));              // ADC0 voltage reference is VDD pin
     ADC0CN0 |= (1 << (7));             // ADC0 enable
}
void main(void) {
      …
     // ADC0 conversion
     ADC0CN0 |= (1 << (4));             // start conversion
     while ((ADC0CN0 & 0x10) == 0);     // wait for done
     ADC0CN0 &= ~(1 << (5));            // reset
     counter = (ADC0H << 8) | ADC0L;    // 12-bit
     counter = counter >> 4;            // to 8-bit
     …
}
```

b) Erklären Sie die Funktion des Programms anhand des Schaltplans und der Pinbelegung.



12.4.1 ADC0CN0: ADC0 Control 0

enable

Bit	7	6	5	4	3	2	1	0
Name	ADEN	IPOEN	ADINT	ADBUSY	ADWINT	ADGN		TEMPE
Access	RW	RW	RW	RW	RW	RW		RW
Reset	0	0	0	0	0	0x0		0

12.4.2 ADC0CN1: ADC0 Control 1

Bit	7	6	5	4	3	2	1	0
Name	ADBITS		ADSJST			ADRPT		
Access	RW		RW			RW		
Reset	0x1		0x0			0x0		

12.4.4 ADC0CF0: ADC0 Configuration

Bit	7	6	5	4	3	2	1	0
Name	ADSC					ADCLKSEL	Reserved	
Access	RW					RW	R	
Reset	0x1F					0	0x0	

c) Ändern Sie das Programm so ab, dass statt des vom ADC ermittelten HEX-Wertes die gemessene Spannung im Flash Magic Terminal ausgegeben wird.

Hinweis: Musterlösung „7b_TRIMMER_ADC“

8. DAC: Digital-Analog-Wandler (12-Bit)

a) Öffnen Sie das Projekt 8_DAC\LED.uvproj und betrachten Sie die C-Datei LED.c:

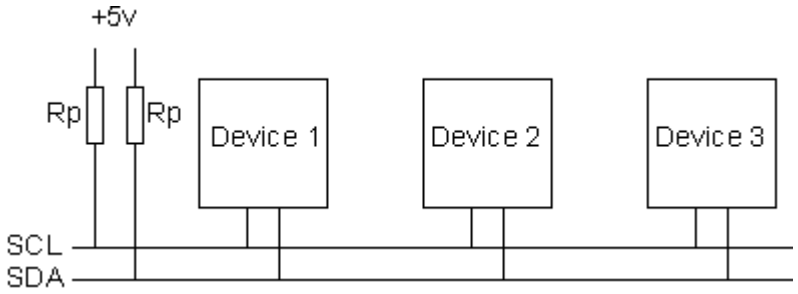
Hinweis: SFR der DACs sind in höherem Speicherbereich.

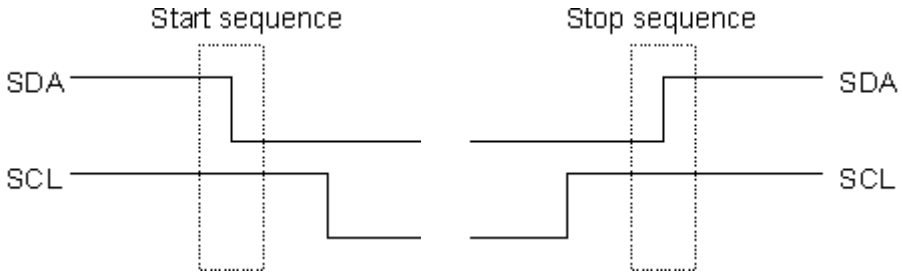
```
void init(void) {
     …
     SFRPAGE = DAC;
     DAC0CF0 |= (1 << 7);     // enable DAC0 on P3.0
     DAC1CF0 |= (1 << 7);     // enable DAC1 on P3.1
     SFRPAGE = DEFAULT;
}
#define DAC0_write(val) {SFRPAGE=DAC; DAC0L=((val)&0x00FF); DAC0H=((val)&0xFF00)>>8; SFRPAGE=DEFAULT;}

#define DAC1_write(val) {SFRPAGE=DAC; DAC1L=((val)&0x00FF); DAC1H=((val)&0xFF00)>>8; SFRPAGE=DEFAULT;}
…
void main(void) {
     …
     while (1) {
          short val;

          LED = 0;
          for (val=0x0FFF; val>0; val--)          // DACs 12-bit down : LEDs on
          {
               DAC1_write(val);         // LED_R
               DAC0_write(val);         // LED_G
               delay();
          }
          …
```

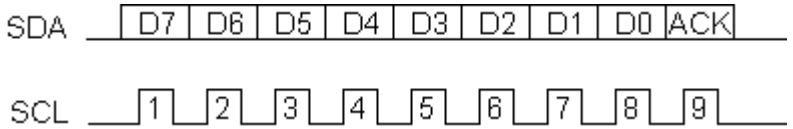
9. Temperatursensor (I2C)

Der I2C-Bus ist ein serieller Bus zur bi-direktionalen synchronen Datenübertragung über nur zwei Leitungen:

SCL synchronous clock
SDA synchronous data





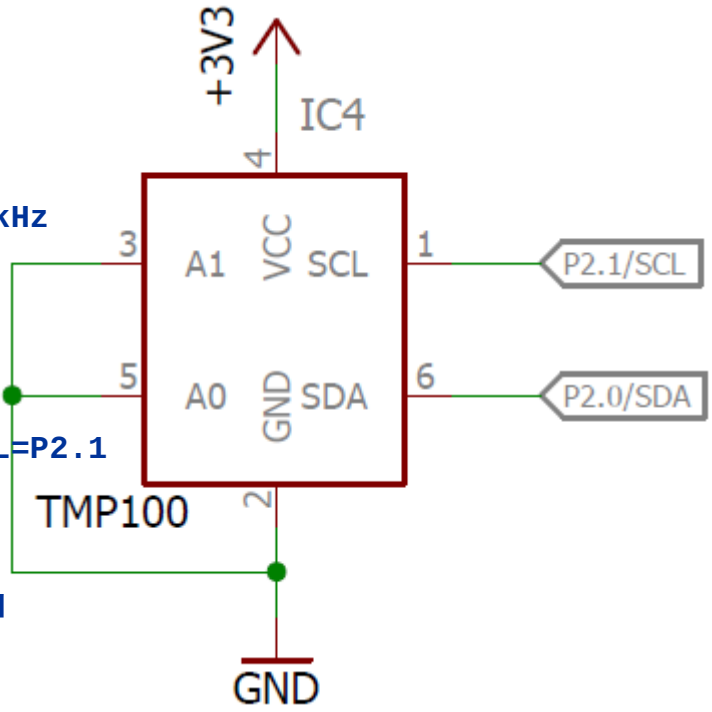


ADD1	ADD0	SLAVE ADDRESS
0	0	1001000
0	Float	1001001
0	1	1001010
1	0	1001100
1	Float	1001101
1	1	1001110
Float	0	1001011
Float	1	1001111

a) Erklären Sie die Funktionsweise des I2C-Bus.

b) Öffnen Sie das Projekt 9_TMP_I2C_HW\LED.uvproj und betrachten Sie die C-Datei LED.c:

```
void init_i2c0(void) { // I2C0 + Timer 0
     I2C0_disconnect();
     // Timer 0 mode 2
     TMOD |= 0x02;
     // Baud rate // 24.5 MHz / 4 / 204 / 2 = 15 kHz
     TH0 = -204;
     TL0 = -204;
     // Timer 0 start
     TR0 = 1;
     I2C0_timeout();
     // I2C0 - skip P1 to move to SDA=P2.0 and SCL=P2.1
     P1SKIP = 0xFF;
     // cross-bar enable I2C0 pins
     XBR0 |= (1 << (2));
     IE |= (1 << (7));        // enable global irq
     I2C0_connect();
}
void init(void) {
     // SYSCLK = HFOSC0 = 24.5 MHz // SYSCLK = SYSCLK / 1
     CLKSEL = 0;
     // Timer 0/1 prescale SYSCLK / 4
     CKCON0 = 1;
     …
}
```



```
#define TMP100_ADR      0x48          // temp. sensor I2C address
…
int main(void) {
     …
     init();
     init_i2c0();
     init_uart0();

     while (1) {
     …
     I2C0_BUF_OUT[0] = 0x01;  // temp. sensor configuration register
     I2C0_BUF_OUT[1] = 0x60;  // %1100000 : TMP_R0=1 and TMP_R1=1 : 12-bit conversion
     I2C0_transfer_start(TMP100_ADR<<1, 2, 0);

     // start conversion
     I2C0_BUF_OUT[0] = 0x00;
     I2C0_transfer_start(TMP100_ADR<<1, 1, 0);

     // temp. sensor I2C address : read (2 bytes) : %1001000.1
     I2C0_transfer_start(TMP100_ADR<<1, 0, 2);

     TempH = I2C0_BUF_IN[0];          // high byte : 1-bit = 1°C
     TempL = I2C0_BUF_IN[1];          // low byte  : 1-bit = 0.0625°C

     Temp = TempH << 8;               // high byte
     Temp = Temp | TempL;             // low byte
     Temp = Temp >> 4;                // 12-bit resolution
     counter = (10*Temp)/16;          // scale : Temp*0.0625*10
     …
```

c) Erklären Sie die Funktion des Programms.