1. Einstieg in ein 8051-IC

In diesem Versuch wird ein 8051-IC, der EFM8BB1 Prozessor von Silicon Labs, vorgestellt:

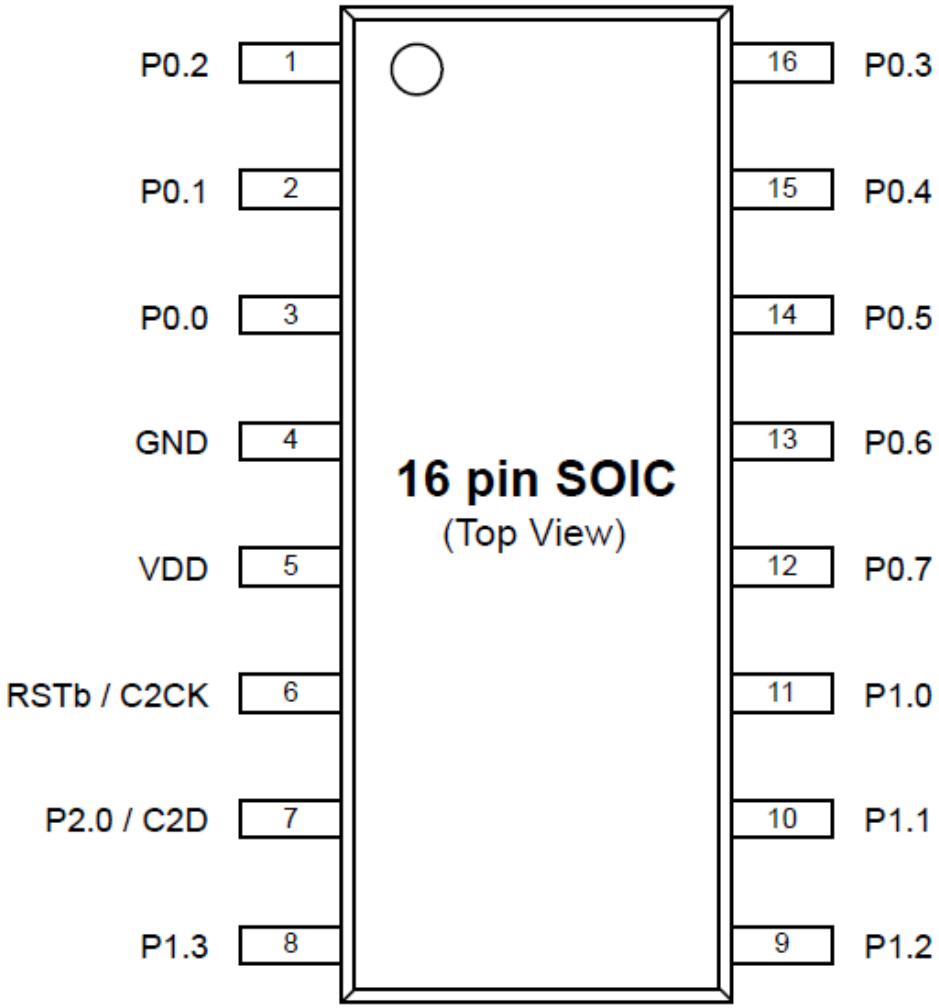
- 8KB Flash, 0.5KB RAM
- 25 MHz interner Oszillator (CLK)
- UART, SPI, I2C
- 2 Analog Comparator / ADC

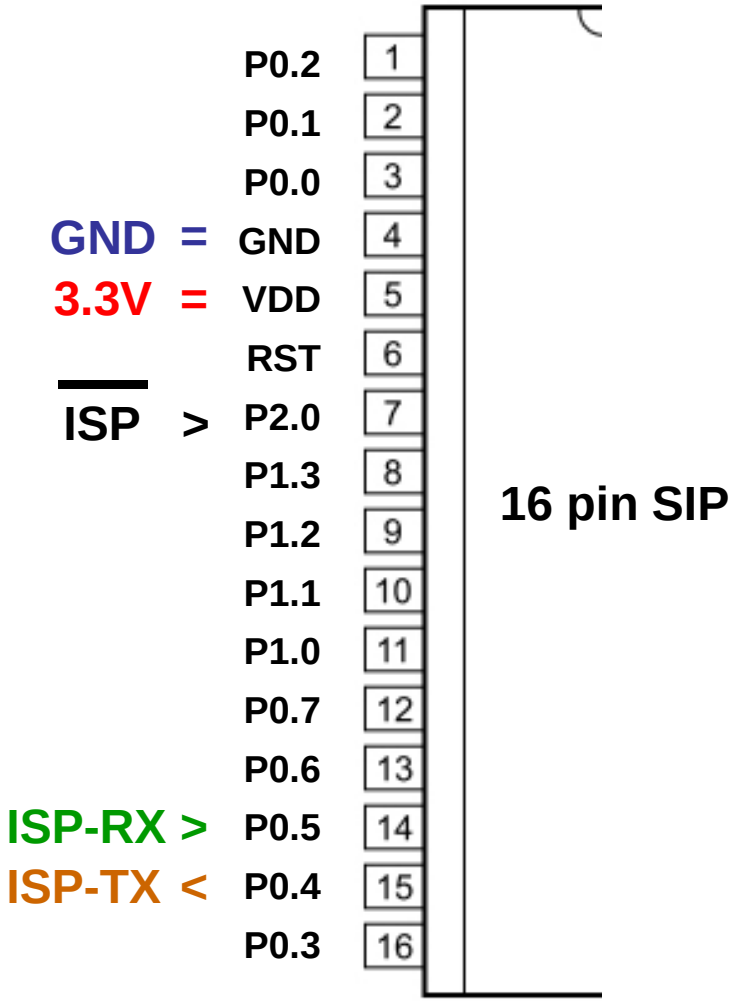
a) Vergleichen Sie die Eigenschaften von EFM8BB1 und 8051

b) Verschaffen Sie sich einen Überblick über den EFM8BB1

Hinweis: Der EFM8BB1 ist nicht als DIP verfügbar, daher wird ein SOIC auf einem SIP-Adapter eingesetzt.

Pinbelegung:

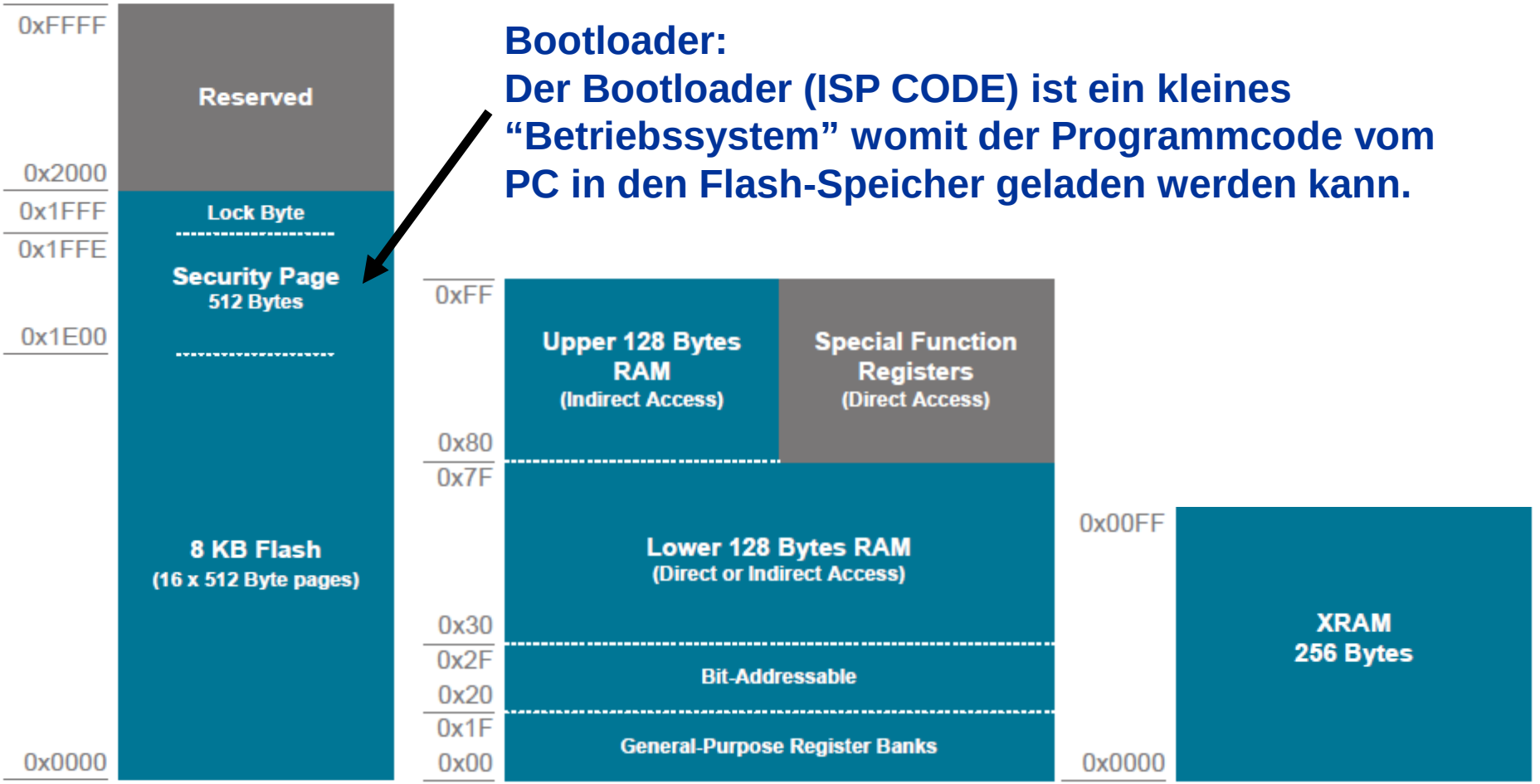




Prof. Dr. Berger © 2018

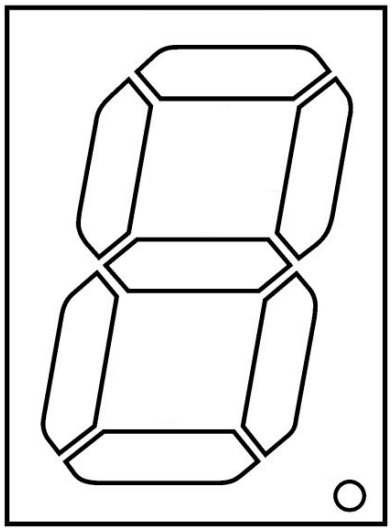
Hochschule Ravensburg-Weingarten

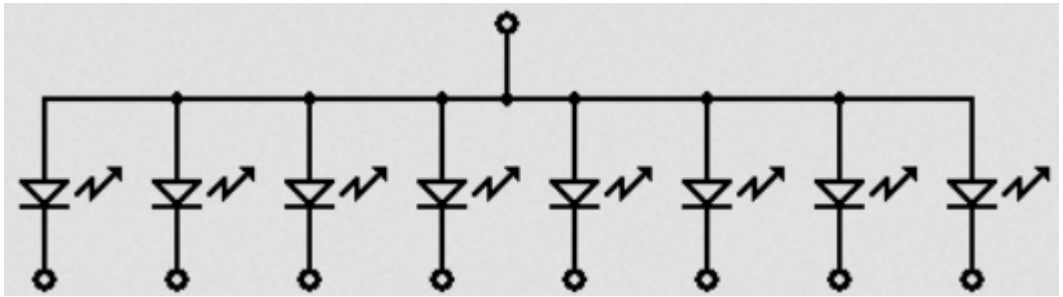
Speicherorganisation des EFM8BB1



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

c) Aufbau einer Schaltung mit 7-Segment Anzeige auf dem Steckbrett:





Anzeige mit gemeinsamer Anode

Die meisten Mikrocontroller können Strom stabiler aufnehmen als abgeben daher ist die gemeinsame Anode üblich. Der EFM8BB1 kann je Pin bis zu 5 mA treiben und 12.5 mA aufnehmen.

Der Anschluss der 8 LEDs der 7-Segment Anzeige erfolgt üblicherweise an einem 8-Bit Port z.B. P0

Beim EFM8BB1 sind aber P0.4 und P0.5 durch den Bootloader belegt. Daher erfolgt der Anschluss als 2x 4-Bit (Nibbles) an P0 und P1

In diesem Versuch wird eine blaue Anzeige mit gemeinsamer Anode direkt ohne Vorwiderstände an den EFM8BB1 angeschlossen, was zu einem Maximalstrom von 11 mA je Pin führt. Das ist möglich, da der EFM8BB1 wie bereits dargelegt bis zu 12.5 mA je Pin aufnehmen kann.

Warnung:
Keine roten, grünen oder gelben Anzeigen ohne Vorwiderstände verwenden, da diese einen Maximalstrom im Bereich 20 mA je Pin erreichen. Der EFM8BB1 hält zwar kurzfristig auch bis zu 100 mA je Pin aus bzw. bis zu 400 mA insgesamt, der USB-seriell-Wandler aber nicht.

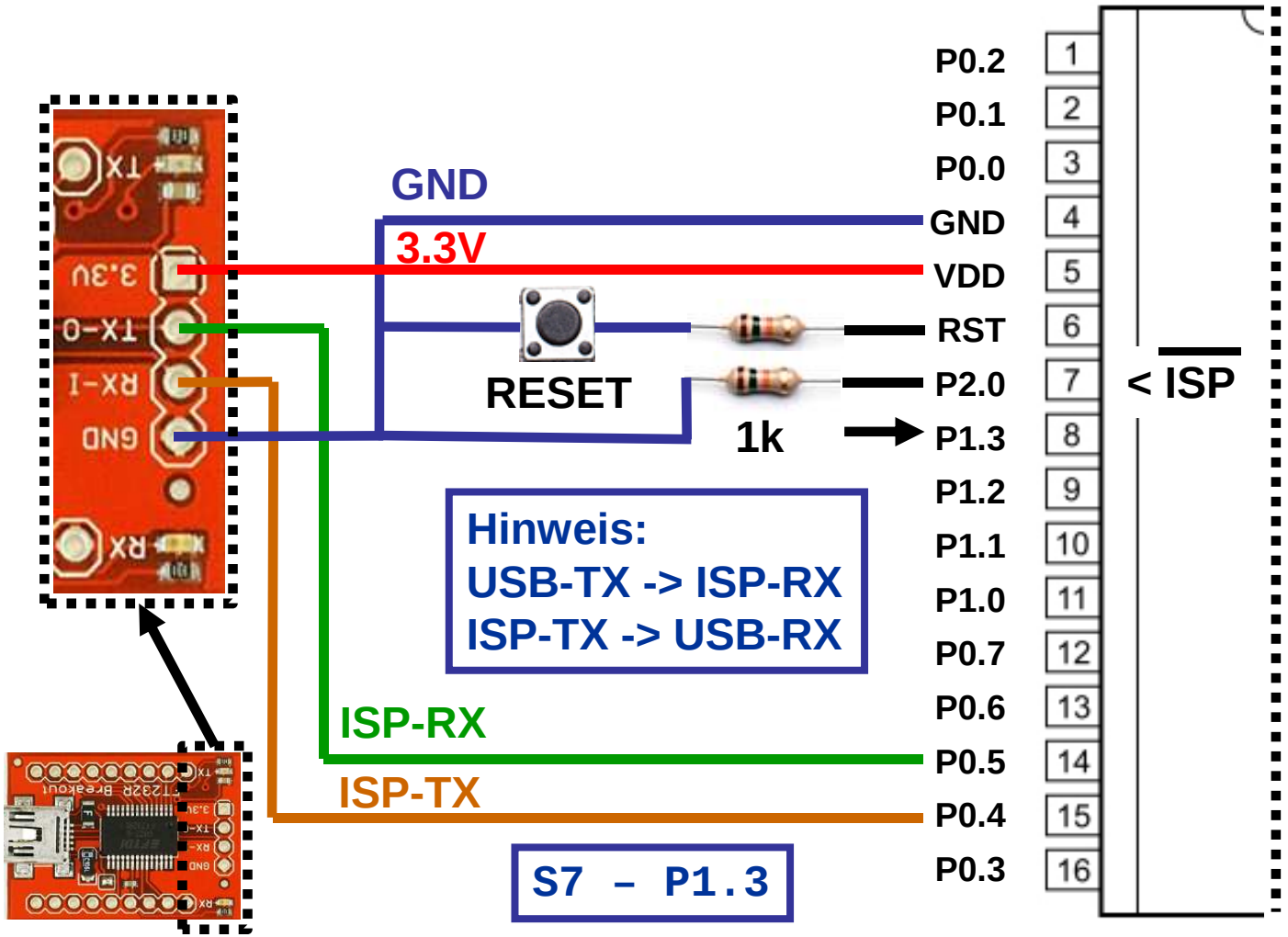
Achtung:
Bei anderen Mikrocontrollern, insbesondere ARM, müssen auch bei blauen Anzeigen Vorwiderstände verwendet werden.

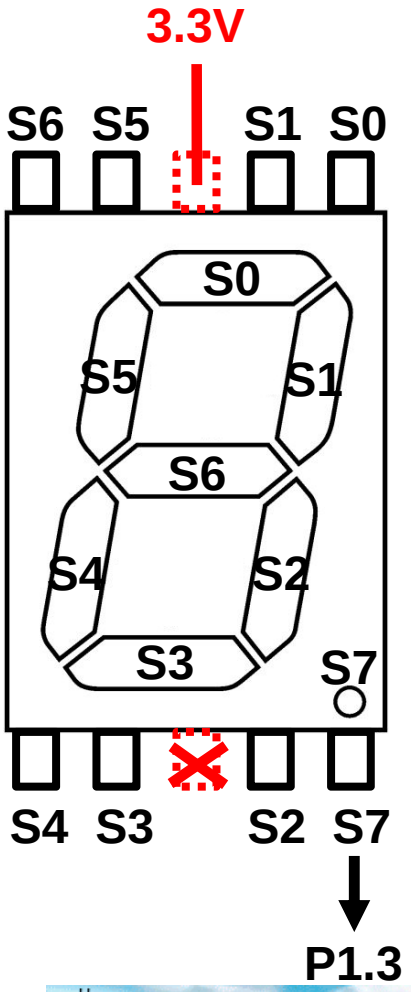
d) **Inbetriebnahme:**
Zunächst wird nur der Anzeigen-Punkt angeschlossen und getestet.

Achtung:
- Vertauschen Sie am EFM8BB1 keinesfalls 3.3V und GND
- Lassen Sie die Schaltung vor Inbetriebnahme von einem Assistenten überprüfen

Schaltplan in Farbe drucken !

USB-seriell-Wandler Typ 1 (FTDI):



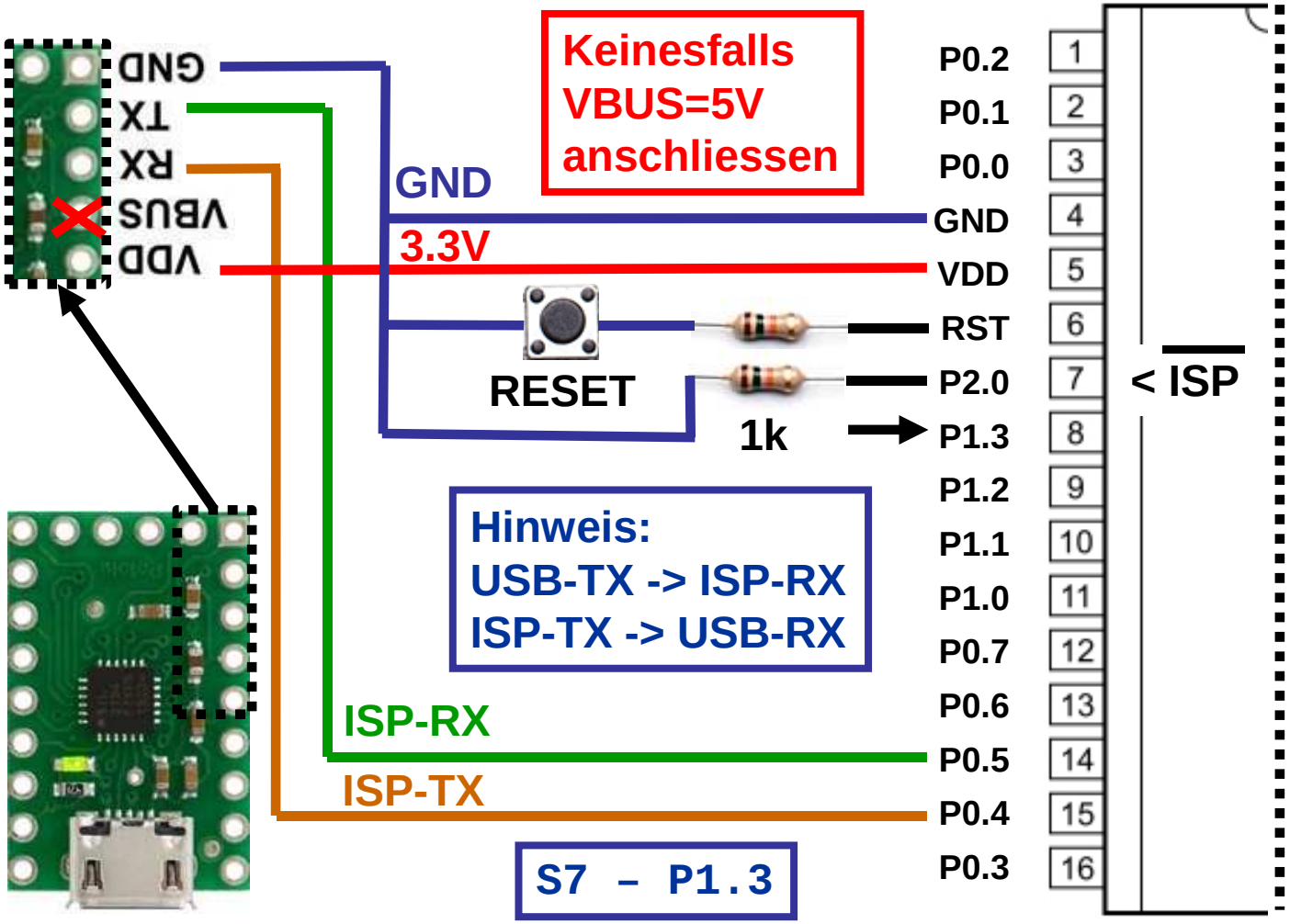


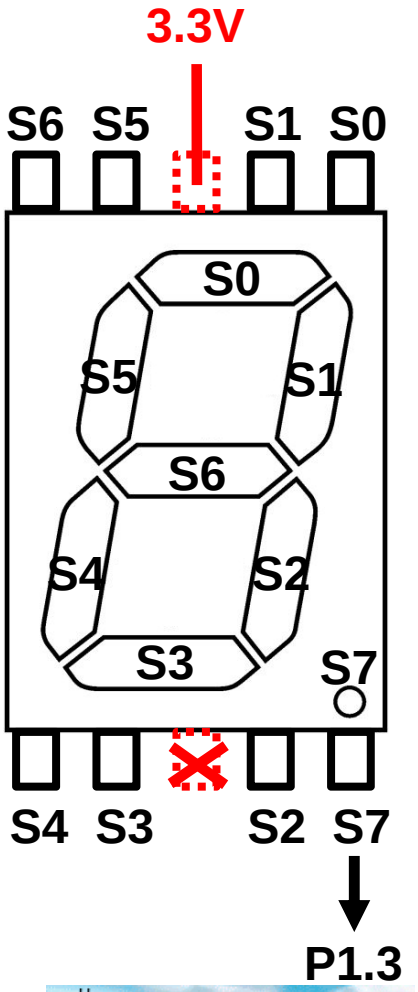
Prof. Dr. Berger © 2018

Hochschule Ravensburg-Weingarten

Schaltplan in Farbe drucken !

USB-seriell-Wandler Typ 2 (Silicon Labs):





Prof. Dr. Berger © 2018

Hochschule Ravensburg-Weingarten

e) Öffnen Sie das Projekt 1a_SEG_ASM und betrachten Sie die Assembler-Datei LED.asm. Erzeugen Sie die HEX-Datei.

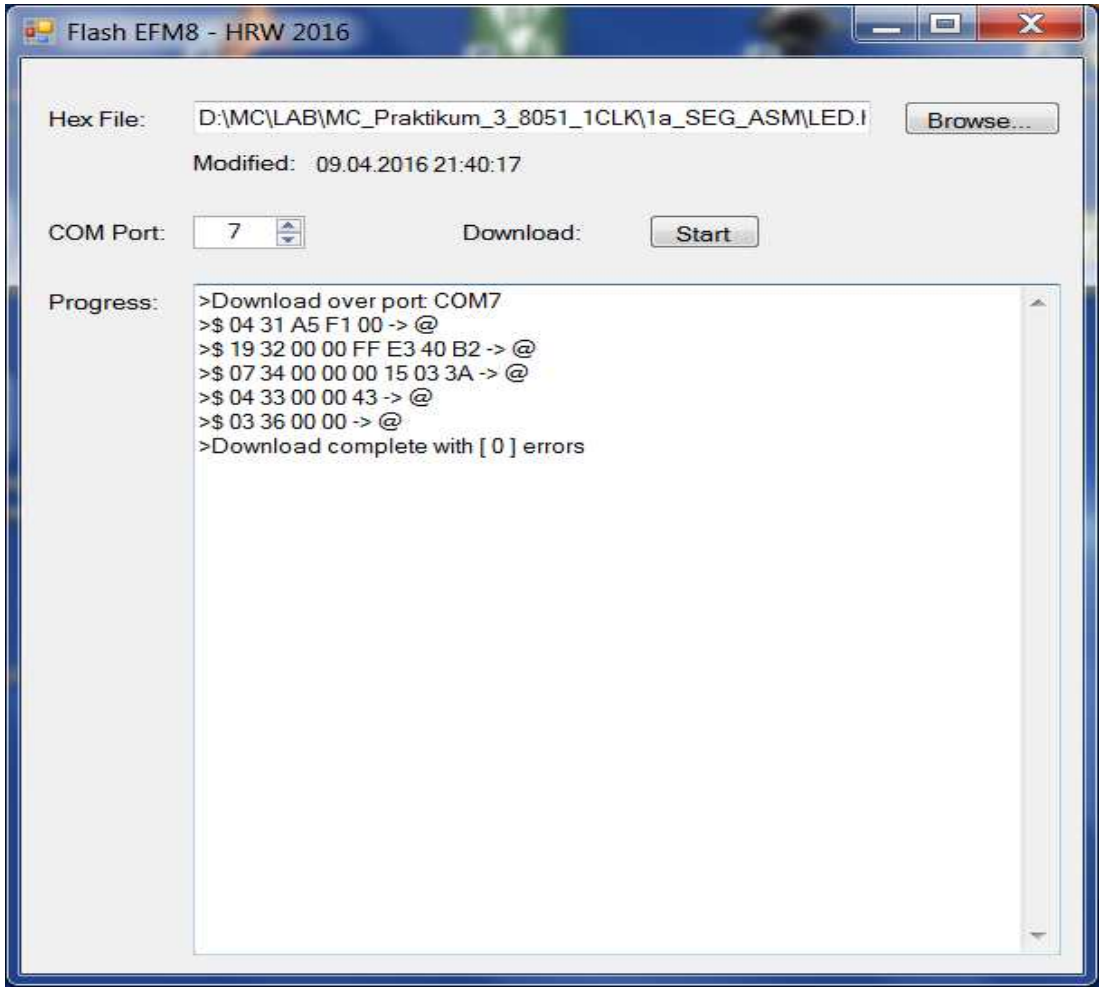
```
XBR2 EQU 0xE3  ; power register

S7   EQU P1.3  ; segment dot

ORG 0
main:
     ORL       XBR2, #01000000B    ; port power
loop:
     CPL       S7                  ; toggle segment dot

     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
```

Flash-Programmierung

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

f) Öffnen Sie FlashEFM8 mit den abgebildeten Einstellungen.

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



- Schließen Sie den USB-seriell Wandler mit einem USB-Kabel an den PC an und drücken Sie den RESET-Taster
- Nun ist der Bootloader aktiv: Laden Sie die HEX-Datei

Das Programm wird ausgeführt. Auf der 7-Segment Anzeige auf dem Steckbrett sollte nun der Anzeigen-Punkt blinken.

g) Fortsetzung der Inbetriebnahme:
Nun sollen schrittweise die weiteren Segmente von S0 bis S6 angeschlossen werden.
Schließen Sie dazu zunächst S0 – P0.0 an und ergänzen Sie das Programm wie folgt:

```
S0   EQU   P0.0
…
CPL  S0
```

Nach Anschluss aller Segmente:
Musterlösung 1b_SEG_ASM

```
S0 – P0.0
S1 – P0.1
S2 – P0.2
S3 – P0.3
S4 – P1.0
S5 – P1.1
S6 – P1.2
S7 – P1.3
```

2. Zifferndarstellung über Pins

a) Öffnen Sie das Projekt 2a_SEG_C und betrachten Sie die C-Datei LED.c:

```
#include "EFM8BB1.h"

// long delay
void delay(unsigned int cnt)
{
     while (--cnt);
     while (--cnt);
     while (--cnt);
     while (--cnt);
}

// segments
sbit S0 = P0^0;
sbit S1 = P0^1;
sbit S2 = P0^2;
sbit S3 = P0^3;
sbit S4 = P1^0;
sbit S5 = P1^1;
sbit S6 = P1^2;
// segment dot
sbit S7 = P1^3;
```

```
void null(void)
{
     S0 = 0;
     S1 = 0;
     S2 = 0;
     S3 = 0;
     S4 = 0;
     S5 = 0;
     S6 = 1;
}

void one(void)
{
     S0 = 1;
     S1 = 0;
     S2 = 0;
     S3 = 1;
     S4 = 1;
     S5 = 1;
     S6 = 1;
}

void main(void)
{
     XBR2 |= 0x40;       // port power
```

```
        while (1)
        {
             null();
             delay(0xFF);
             one();
             delay(0xFF);
             S7 = 0;         // blink dot
             delay(0xFF);
             S7 = 1;
             delay(0xFF);
        }
    }
```

b) Erklären Sie die Funktion des Programms.

c) Ergänzen Sie das Programm so, dass die Ziffern 0 bis 3 dargestellt werden.

Musterlösung: 2b_SEG_C

3. Zifferndarstellung über Ports

a) Öffnen Sie das Projekt 3a_SEG_PORT_C und betrachten Sie die C-Datei LED.c:

```
#include "EFM8BB1.h"

// long delay
void delay(unsigned int cnt)
{
     while (--cnt);
     while (--cnt);
     while (--cnt);
     while (--cnt);
}

// segment numbers
// 0 : %1100|0000
// 1 : %1111|1001
code unsigned char seg[] = {0xC0, 0xF9};

// segment dot
sbit DOT = P1^3;

void seg_write(unsigned char val)
{
```

```
    unsigned char high, low;

    P0 |= 0x0F;    // clear low nibble
    P1 |= 0x0F;    // clear high nibble

    low = 0x0F & seg[val];             // get low nibble from array
    high = (0xF0 & seg[val]) >> 4;     // get high nibble from array

    P0 &= (0xF0 | low);                // write low nibble
    P1 &= (0xF0 | high);               // write high nibble
}

void main(void)
{
    XBR2 |= 0x40;       // port power
    while (1)
    {
        unsigned char idx;
        for (idx=0; idx<=1; idx++)
        {
            seg_write(idx);
            delay(0xFF);
        }
        DOT = 0;                 // blink dot
        delay(0xFF);
        DOT = 1;
        delay(0xFF);
  }
}
```

b) Die 8 LEDs der 7-Segment Anzeige sind als 2x 4-Bit (Nibbles) angeschlossen. Erklären Sie wie aus dem 8-Bit Code eines Zeichens schrittweise die beiden Nibbles erzeugt werden, am Beispiel der Ziffer 0:

```
code unsigned char seg[] = {0xC0, …

…

P0 |= 0x0F;                          // clear low nibble
P1 |= 0x0F;                          // clear high nibble

low = 0x0F & seg[val];               // get low nibble from array
high = (0xF0 & seg[val]) >> 4;       // get high nibble from array

P0 &= (0xF0 | low);                  // write low nibble
P1 &= (0xF0 | high);                 // write high nibble
```

c) Ergänzen Sie das Programm so, dass die Ziffern 0 bis 3 dargestellt werden.

Musterlösung: 3b_SEG_PORT_C

4. Analog-Digital-Wandler

Der ADC kann in 8-Bit, 10-Bit, 12-Bit Auflösung betrieben werden. Für den Betrieb eines Poti ist die 8-Bit Auflösung ausreichend.

Der ADC wird über diese Register konfiguriert:

12.4.1 ADC0CN0: ADC0 Control 0

Bit	7	6	5	4	3	2	1	0
Name	ADEN	ADBMEN	ADINT	ADBUSY	ADWINT	ADCM		
Access	RW	RW	RW	RW	RW	RW		
Reset	0	0	0	0	0	0x0		
SFR Address: 0xE8 (bit-addressable)								

enable done start

12.4.3 ADC0CF: ADC0 Configuration

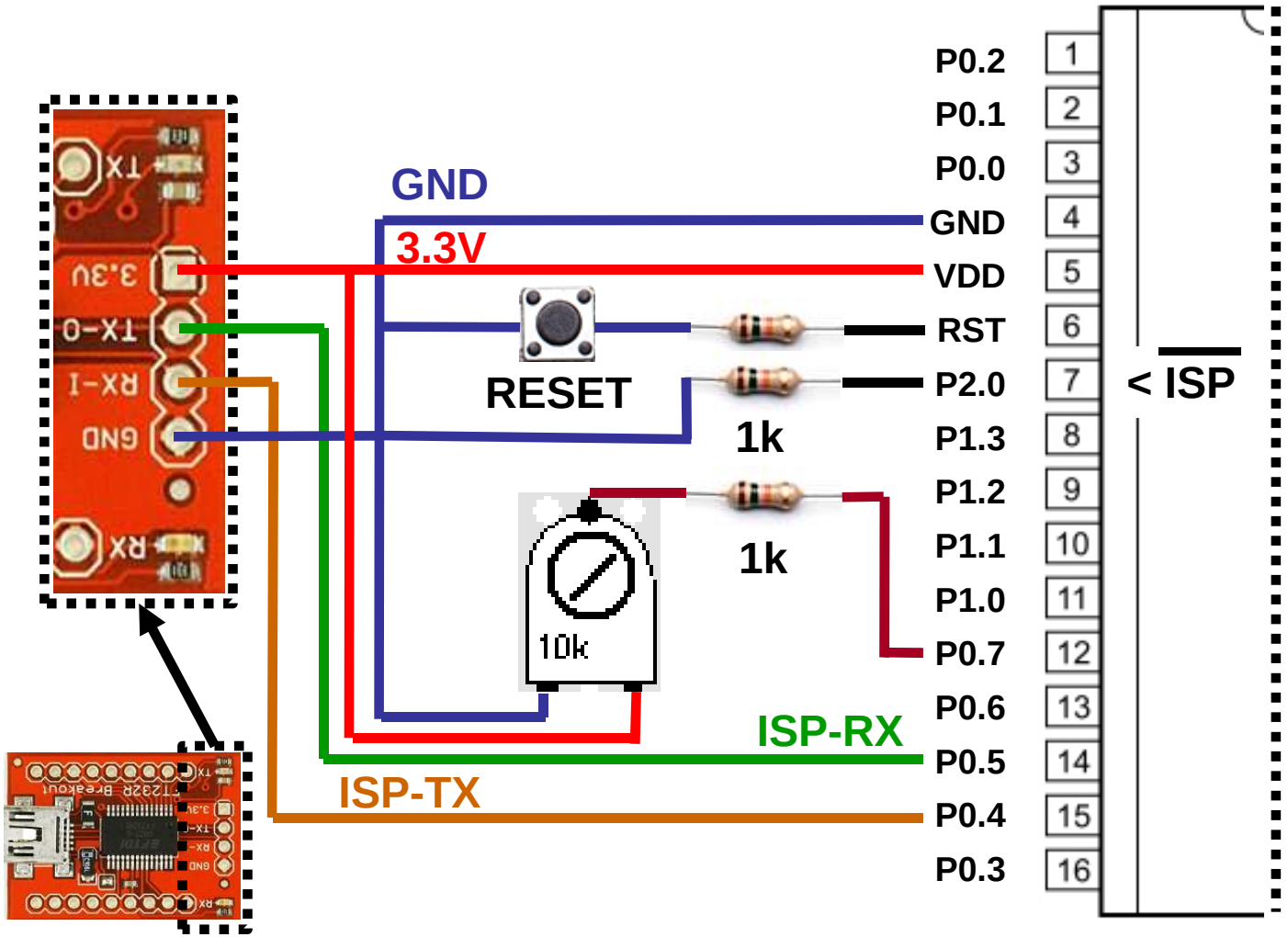
Bit	7	6	5	4	3	2	1	0
Name	ADSC					AD8BE	ADTM	ADGN
Access	RW					RW	RW	RW
Reset	0x1F					0	0	0
SFR Address: 0xBC								

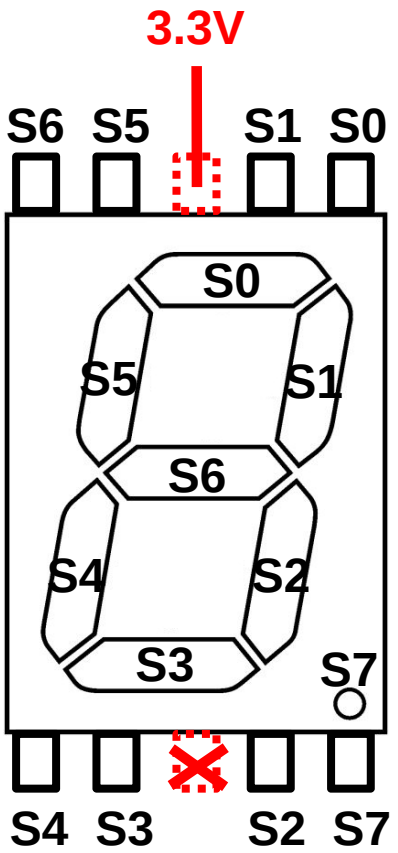
SYSCLK/(n+1) | bit mode | delay | gain

a) Inbetriebnahme:

Das Poti wird wie dargestellt an P0.7 angeschlossen

Schaltplan in Farbe drucken !





Prof. Dr. Berger © 2018

Hochschule Ravensburg-Weingarten

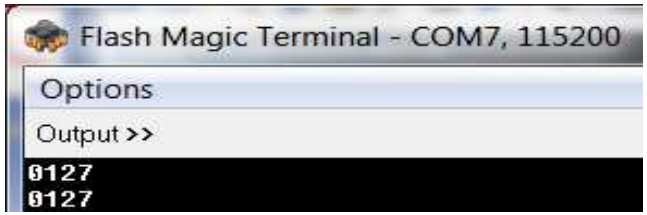
b) Öffnen Sie das Projekt 4a_TRIM_C und betrachten Sie die C-Datei LED.c:

```
#include "EFM8BB1.h"
// long delay
void delay (unsigned char val) {
     volatile short idx, cnt;

     for (idx=0; idx<=val; idx++)
          for (cnt=0; cnt<=0x3FFF; cnt++);
}
// segment dot
sbit S7 = P1^3;
// serial send
void send(unsigned char ch) {
     // add parity to char
     // unsigned char par = ((unsigned char) P) << 7;
     // SBUF = ch | par;

     // no partity
     SBUF = ch;

     while (!TI);   // wait until transmitted
     TI=0;
}
void init(void) {
     // EFM8 specific serial configuration
```

```
    // P0.4 - UART0 TX - PUSH_PULL
    // P0.5 - UART0 RX - OPEN_DRAIN
    P0MDOUT |= (1 << (4));
    // EFM8 specific ADC configuration
    // P0.7 - ADC0
    P0MDIN &= ~(1 << (7));   // set P0.7 to analog input
    P0SKIP |= (1 << (7));    // disconnect P0.7 from digital
    ADC0MX = 7;              // connect P0.7 to ADC0 via MUX
    // ADC0 conf. - %00001|1|0|1 - SYSCLK/(1+1)|8-bit mode|no delay|gain=1
    ADC0CF = 0x0D;
    REF0CN = 0x08;           // voltage reference is VDD=3.3V
    ADC0CN0 |= (1 << (7));   // enable ADC0

    // SYSCLK = HFOSC0 = 24.5 MHz // SYSCLK = SYSCLK/1
    CLKSEL = 0;
    // timer 1 uses SYSCLK
    CKCON0 |= (1 << (3));

    XBR0 |= (1 << (0));      // cross-bar enable UART0 pins
    XBR2 |= (1 << (6));      // cross-bar enable all pins
    // 8051 serial configuration // UART mode %01 = 8-bit data
    SCON = 0x40;
    // timer 1 mode 2
    TMOD = 0x20;
    // Baud rate
    TH1 = 0x96;    // 256-150=106 - 24.5 MHz / 106 = 231132 / 2 = 115200 Baud
    TL1 = 0x96;
    // timer 1 start
    TR1 = 1;
}
```

```
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;

          // ADC0 conversion
          ADC0CN0 |= (1 << (4));                  // start conversion
          while ((ADC0CN0 & 0x10) == 0);          // wait for done
          ADC0CN0 &= ~(1 << (5));                 // reset
          // CAUTION: 8-bit result is bits 9:2
          counter = ((ADC0H << 8) | ADC0L);
          counter >>= 2;      // 8-bit result

          // 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);

          // blink dot
          S7 ^= 1;
          delay(25);
     }
}
```

c) Öffnen Sie in FlashMagic ein Terminal mit den geeigneten Einstellungen: Tools->Terminal / 115200 Baud



d) Mit dem vorliegenden Programm wird das Ergebnis der 8-Bit AD-Konvertierung von 0 bis 255 seriell ausgegeben. Ändern Sie das Programm so ab, dass stattdessen die gemessene Spannung von 0.00 bis 3.30V ausgegeben wird.

Musterlösung: 4b_TRIM_VOLT_C

5. Voltmeter mit 7-Segment Anzeige

Die am Poti anliegende Spannung soll nun mittels der 7-Segment Anzeige ausgegeben werden.

Ergänzen Sie das Programm 4b_TRIM_VOLT_C aus der vorherigen Aufgabe so, dass die ganzzahlige Spannung von 0 bis 3V auf der 7-Segment Anzeige ausgegeben wird. Verwenden Sie hierzu Ihr Ergebnis aus 2b_SEG_C

Musterlösung: 5_TRIM_SEG_C

Prof. Dr. Berger © 2018