1. Einstieg in ein ARM-IC

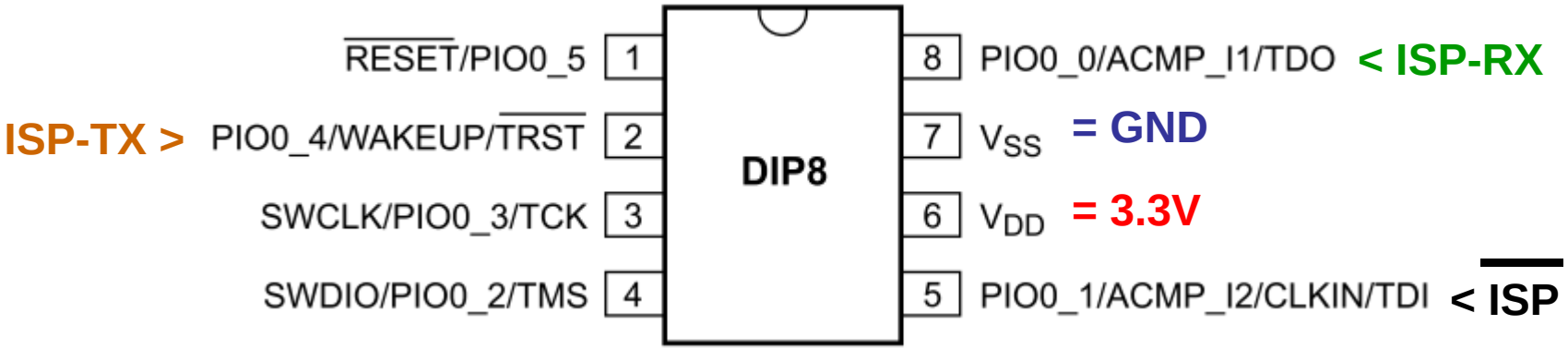
In diesem Versuch wird ein ARM-IC, der Cortex-M0 Prozessor LPC810 von NXP, vorgestellt:

- 4KB Flash, 1KB RAM
- 12 MHz interner Oszillator (CLK)
- 2 USART, SPI, I2C
- 2 Analog Comparator

a) Vergleichen Sie die Eigenschaften von LPC810 und Standard ARM

b) Verschaffen Sie sich einen Überblick über den LPC810

Pinbelegung:



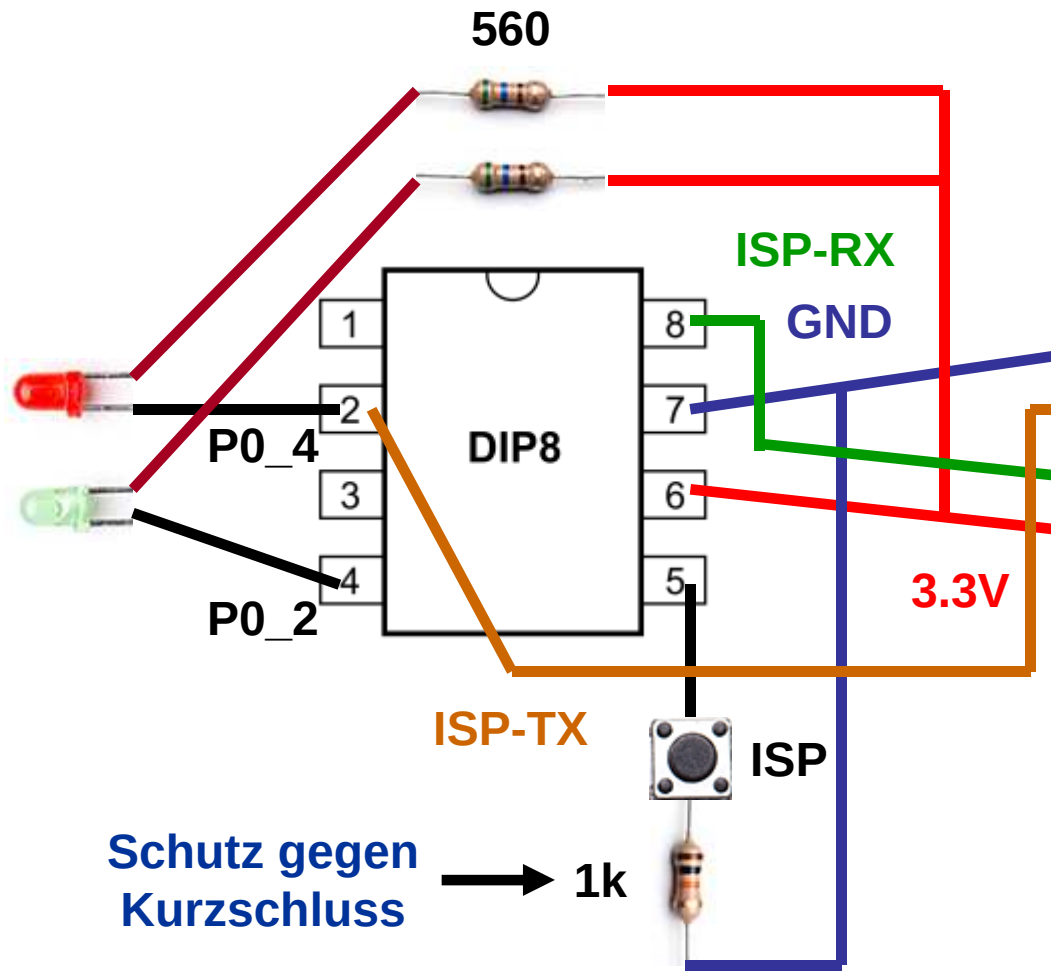
c) Aufbau einer Schaltung mit zwei LEDs auf dem Steckbrett:

Achtung:

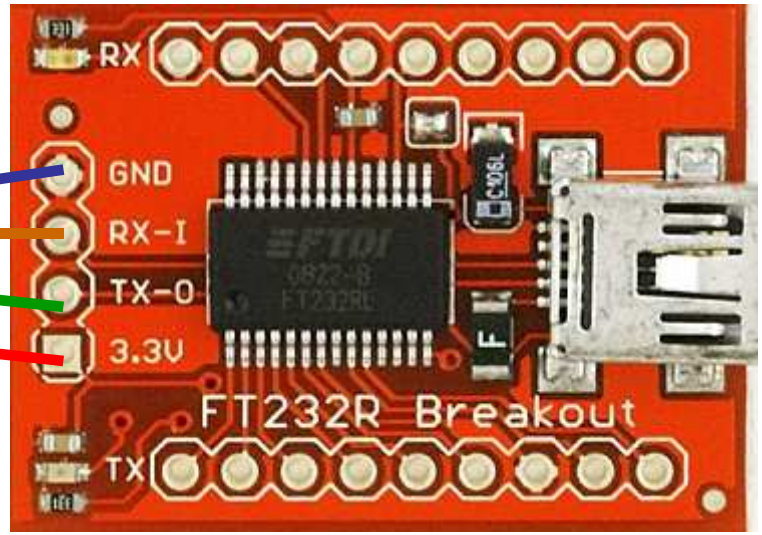
- **Vertauschen Sie am LPC810 keinesfalls 3.3V und GND**
- **Lassen Sie die Schaltung vor Inbetriebnahme von einem Assistenten überprüfen**

Schaltplan in Farbe drucken !

USB-seriell-Wandler Typ 1 (FTDI):



Hinweis:
Auf die Polung der LEDs achten

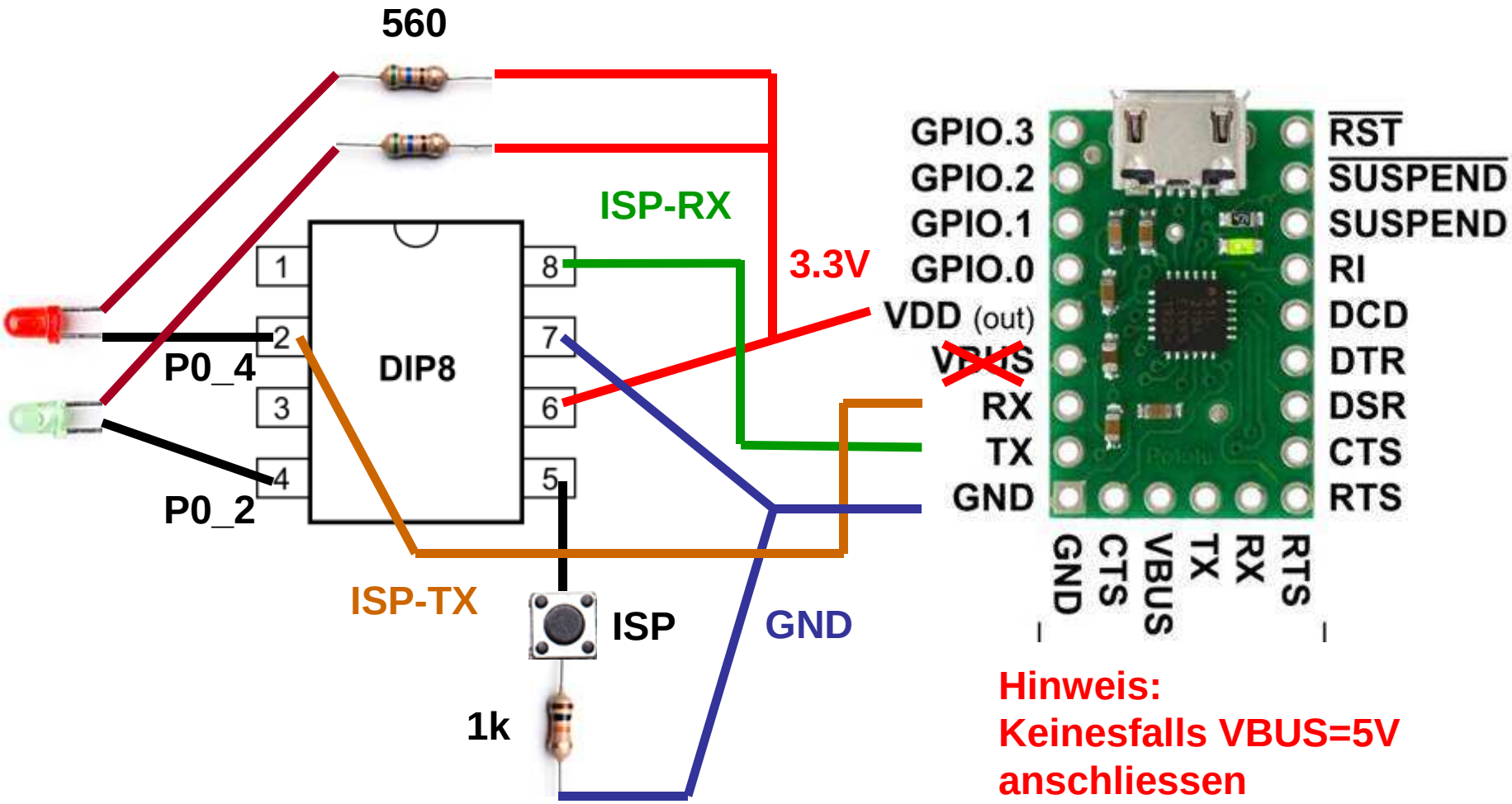


Hinweis:
USB-TX -> ISP-RX
ISP-TX -> USB-RX

Hochschule Ravensburg-Weingarten

Schaltplan in Farbe drucken !

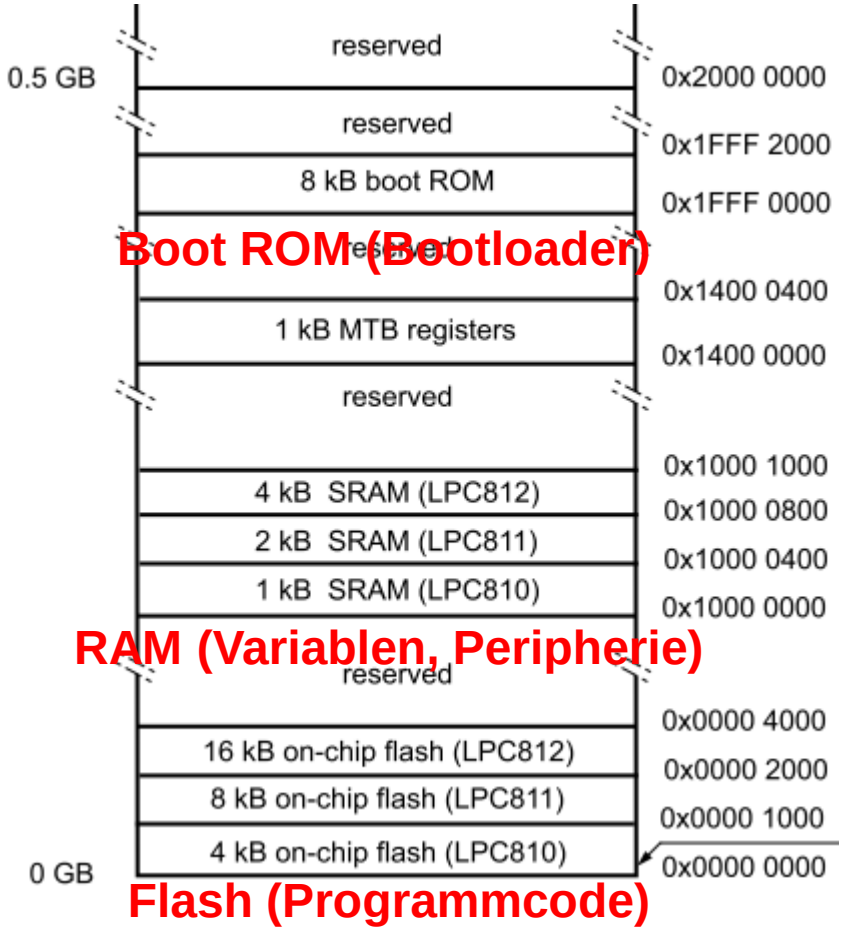
USB-seriell-Wandler Typ 2 (Silicon Labs):

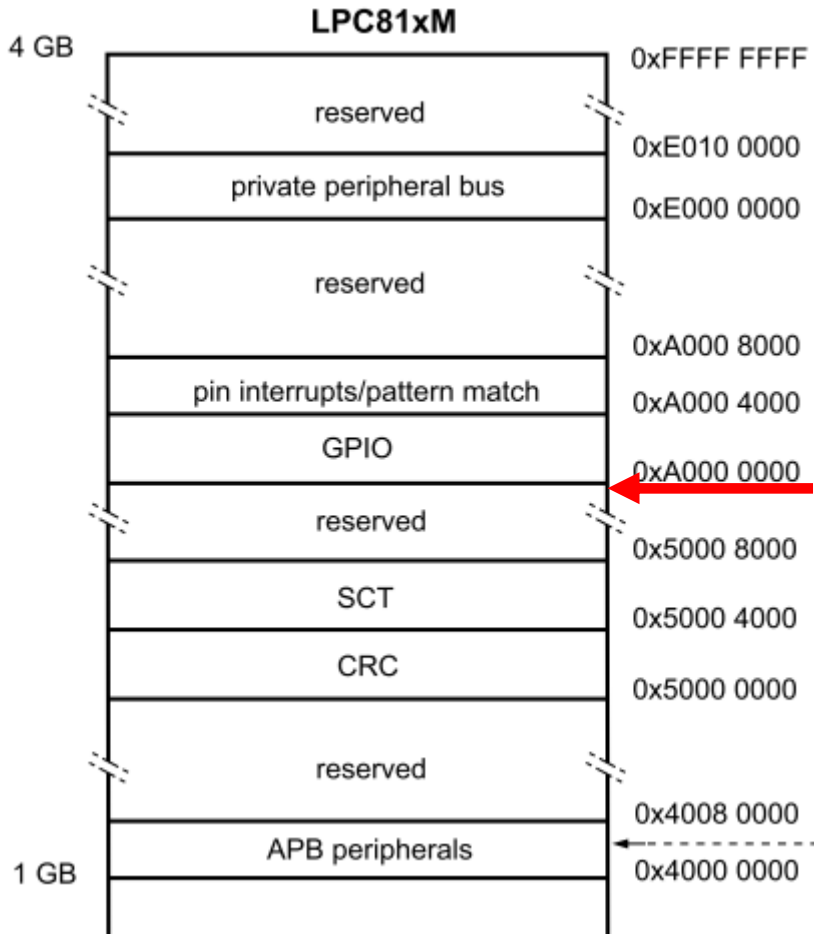


Hinweis:
Keinesfalls VBUS=5V anschliessen

2. Speicherorganisation des LPC810

a) Memory Map





b) Linker: LPC810M021.icf

In der Datei: LPC810M021.icf werden die Speicherbereich-Informationen (Segmentierung) für den Linker hinterlegt. Vergleichen Sie diese Informationen mit der Memory Map.

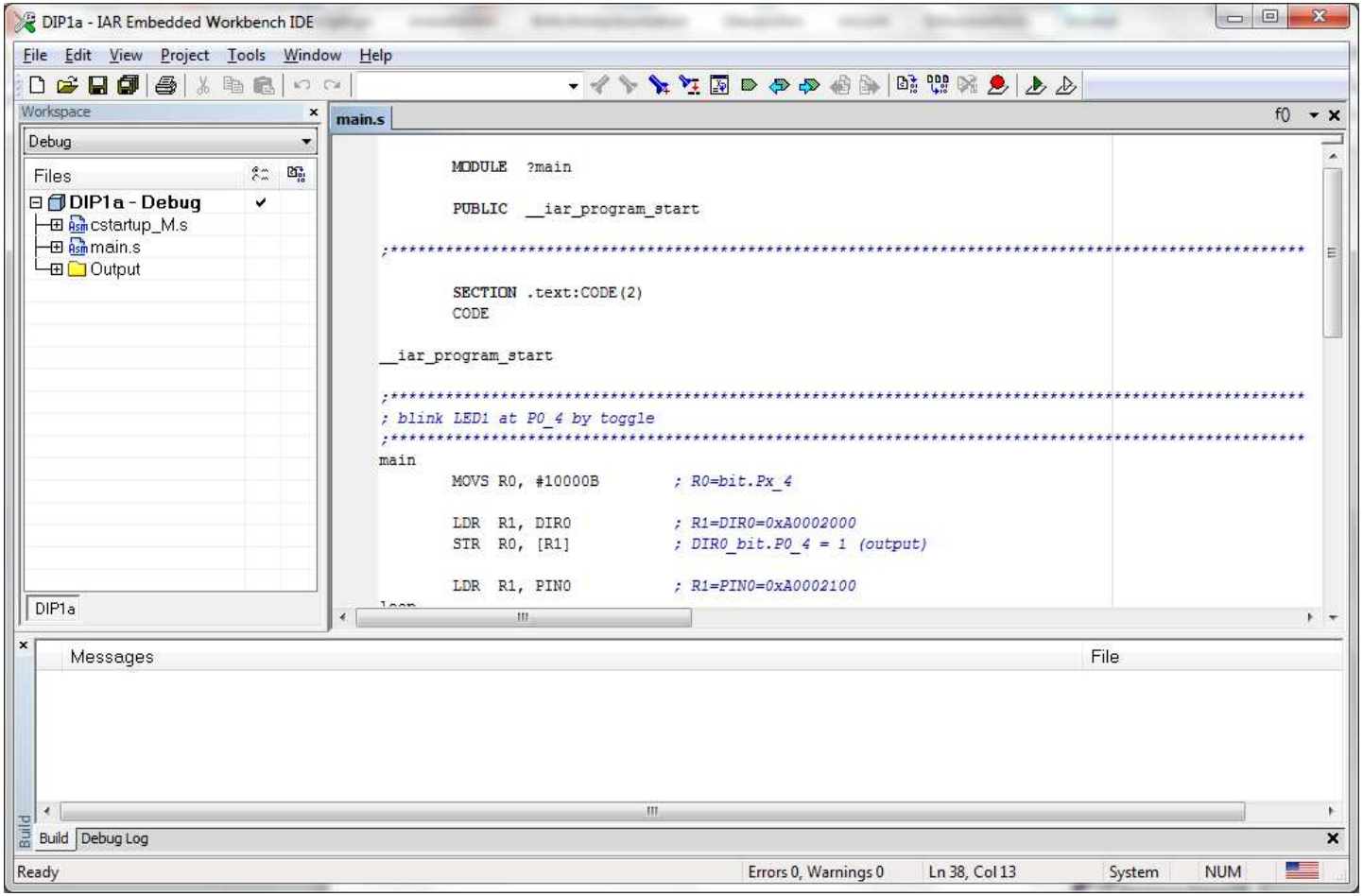
c) Bootloader

Der Bootloader im Boot ROM ist ein kleines “Betriebssystem”, womit der Programmcode in Flash oder RAM geladen werden kann. Bei Reset wird der Programmcode dann dort ausgeführt.

UART-Bootloader: Programmcode über serielle Schnittstelle

3. Einstieg in ARM Assembler

IAR-Entwicklungssystem:



a) Öffnen Sie das Projekt 1a_ASM_TOGGLE/DIP1a.eww und betrachten Sie die Assembler-Datei main.s:

Hinweis: Der M0 Prozessor hat gegenüber dem M3 die Einschränkung dass arithmetisch-logische Befehle mit Immediate Value immer die Flags updaten, es gibt dann z.B. keinen Unterschied zwischen MOV und MOVS

```
        MODULE  ?main

        PUBLIC  __iar_program_start

        SECTION .text:CODE(2)
        CODE

__iar_program_start

main
        MOVS R0, #10000B        ; R0=bit.Px_4

        LDR  R1, DIR0           ; R1=DIR0=0xA0002000
        STR  R0, [R1]           ; DIR0_bit.P0_4 = 1 (output)

        LDR  R1, PIN0           ; R1=PIN0=0xA0002100
loop
        LDR  R2, [R1]           ; read PIN0
        EORS R2, R2, R0         ; PIN0_bit.P0_4 ^= 1 (LED1 toggle)
        STR  R2, [R1]           ; write PIN0

        BL   delay              ; LR=PC (branch)

        B    loop               ; loop
```

Fortsetzung main.s:

```
delay
        PUSH {R0}
        MOVS R0, #1
        LSLS R0, R0, #20        ; 2^20=1048576
count
        SUBS R0, R0, #1
        BNE  count              ; <>0
        POP  {R0}
        MOV  PC, LR             ; PC=LR (return)

        SECTION .text:CODE(2)
        DATA

DIR0
        DC32 0xA0002000
PIN0
        DC32 0xA0002100

        END
```

Return mit MOV anstatt MOVS !

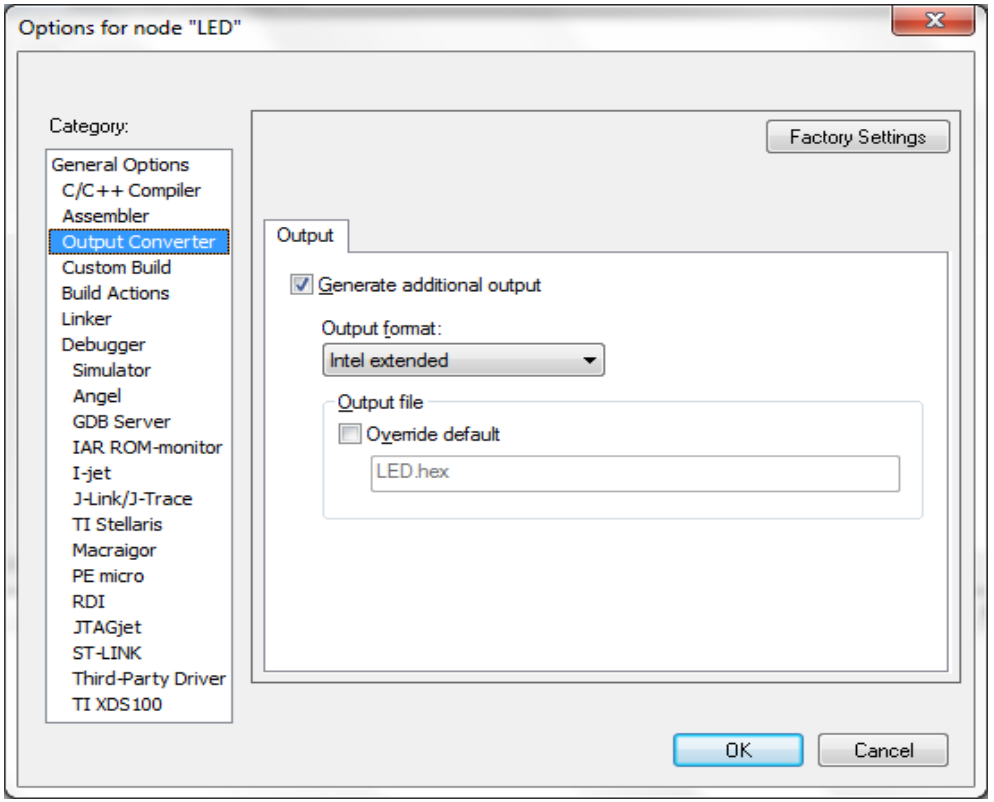
Hinweis:
In der Assembler-Datei main.s wird die Clock CLK nicht initialisiert. Somit läuft der Mikrocontroller mit dem internen RC-Oszillator mit 12 MHz (ein höherer Takt bis zu 30 MHz ist möglich)

b) Erstellen Sie die HEX-Datei:

Output Converter -> HEX

c) Kompilieren Sie das Projekt:

Project->Rebuild All



d) Erklären Sie die Funktion der Assembler-Datei main.s:

Wie wird P0_4 für die Ansteuerung der LED genutzt?
Hinweis: DIR0 und PIN0 Register

Table 69. Register overview: GPIO port (base address 0xA000 0000)

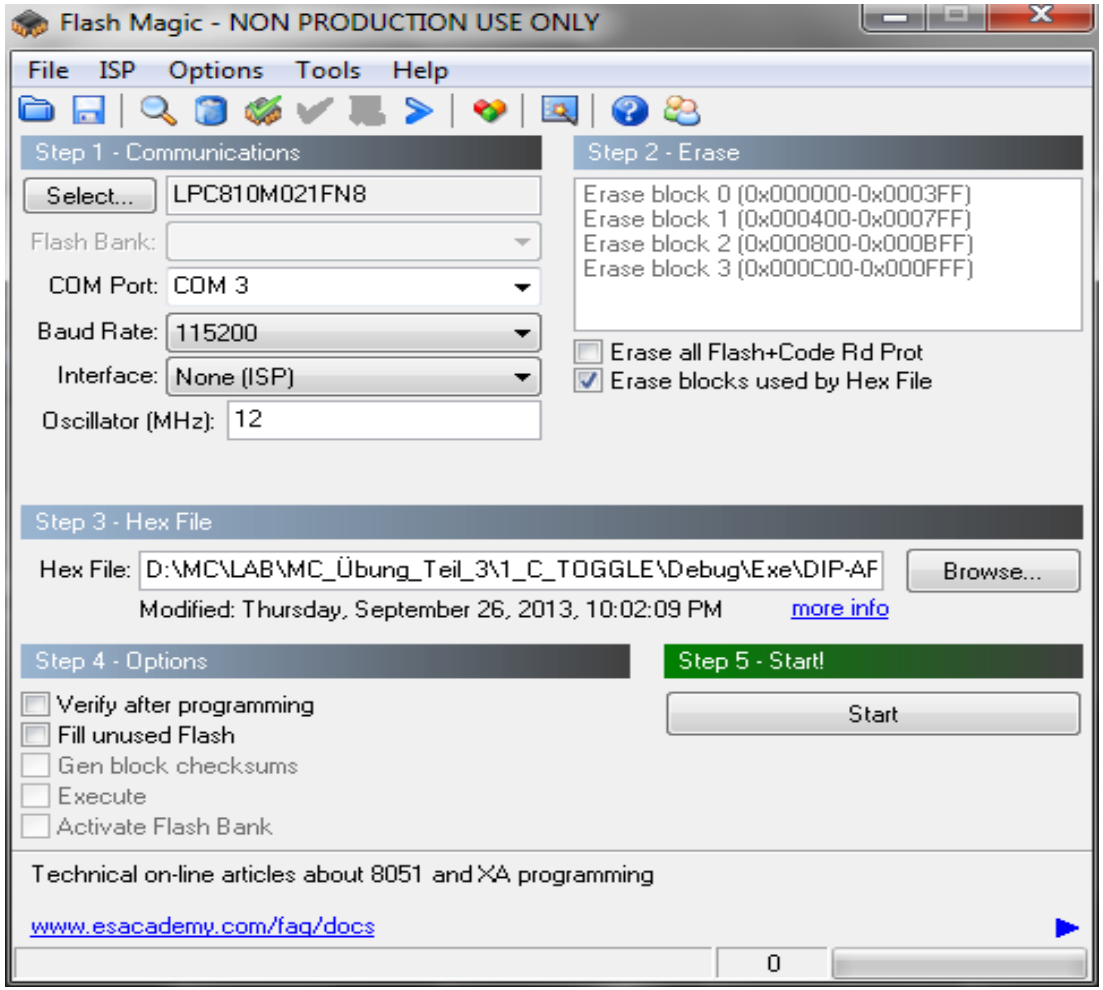
Name	Access	Address offset	Description	Reset value	Width	Reference
B0 to B17	R/W	0x0000 to 0x0012	Byte pin registers port 0; pins PIO0_0 to PIO0_17	ext	byte (8 bit)	Table 70
W0 to W17	R/W	0x1000 to 0x1048	Word pin registers port 0	ext	word (32 bit)	Table 71
DIR0	R/W	0x2000	Direction registers port 0	0	word (32 bit)	Table 72
MASK0	R/W	0x2080	Mask register port 0	0	word (32 bit)	Table 73
PIN0	R/W	0x2100	Port pin register port 0	ext	word (32 bit)	Table 74
MPIN0	R/W	0x2180	Masked port register port 0	ext	word (32 bit)	Table 75
SET0	R/W	0x2200	Write: Set register for port 0 Read: output bits for port 0	0	word (32 bit)	Table 76
CLR0	WO	0x2280	Clear port 0	NA	word (32 bit)	Table 77
NOT0	WO	0x2300	Toggle port 0	NA	word (32 bit)	Table 78

4. Einstieg in die Flash-Programmierung

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

a) Öffnen Sie Flash Magic mit den abgebildeten Einstellungen.

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



b) Halten Sie den ISP-Taster gedrückt und schließen Sie den USB-seriell Wandler mit einem USB-Kabel an den PC an

c) Lesen Sie die Device Signature des Prozessors aus.
-> ISP

d) Laden Sie die HEX-Datei:
-> Projektordner: /Debug/Exe
und stecken Sie das USB-Kabel am PC kurz ab und wieder an
(oder lösen Sie einen RESET aus)

Auf dem Steckbrett sollte nun eine LED blinken.

e) In welche Speicherblöcke wird der Code abgelegt?
-> ISP

f) Wieviel Prozent des Flash werden durch den Programmcode belegt?
-> ISP

g) Vertiefung: Öffnen Sie das Projekt 1b_ASM_BIT/DIP1b.eww und erstellen und laden Sie die HEX-Datei:

```
        …

; blink LED1 at P0_4 by bit-addressable memory (atomic access)

        …

        LDR  R1, PIN0BASE       ; R1=PIN0BASE=0xA0000000

        MOVS R2, #0
        MOVS R3, #1
loop
        STRB R2, [R1, #4]       ; clear P0_4
        BL   delay              ; LR=PC (branch)
        STRB R3, [R1, #4]       ; set P0_4
        BL   delay              ; LR=PC (branch)

        B    loop               ; loop

        …
```

Hinweis: Der LPC810 verfügt wie der 8051 über Bit-adressierbaren Speicher (anders als Standard ARM)

Erklären Sie die Funktion der Assembler-Datei main.s

Was ist „atomarer Zugriff“ ?

h) Vertiefung: Öffnen Sie das Projekt 1c_ASM_SFR/DIP1c.eww und erstellen und laden Sie die HEX-Datei:

```
        …
; blink LED1 at P0_4 by SFR (atomic access)
        …
        MOVS R0, #10000B        ; R0=bit.Px_4
        …
        LDR  R1, NOT0           ; R1=NOT0=0xA0002300
loop
        STR  R0, [R1]           ; NOT0_bit.P0_4 = 1 (LED1 toggle)
        BL   delay              ; LR=PC (branch)
        B    loop               ; loop
        …
```

Erklären Sie die Funktion der Assembler-Datei main.s

Was ist bei den Cortex-M ein SFR (Special Function Register) ?

5. Einstieg in ARM C

a) Öffnen Sie das Projekt 2a_C_TOGGLE/DIP2a.eww und betrachten Sie die C-Datei main.c:

```
#include "lpc810.h"

void delay(volatile unsigned long cycles)
{
  while (cycles) {cycles--;}
}

void init(void)
{
  // LED on P0_4

  // P0_4 digital is default
  // no setting required

  // P0_4 set to output
  DIR0_bit.P0_4 = 1;

  // P0_4 set to high (LED off)
  PIN0_bit.P0_4 = 1;
}
```

Hinweis: Der Zugriff auf Register-Bits erfolgt über in „lpc810.h“ definierte structs

Fortsetzung main.c:

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

  while (1)
  {
    // P0_4 toggle
    PIN0_bit.P0_4 ^= 1;

    delay(500000);
  }
}
```

b) Erstellen Sie die HEX-Datei, kompilieren Sie das Projekt und laden Sie die HEX-Datei -> Projektordner: /Debug/Exe

Auf dem Steckbrett sollte nun eine LED blinken.

c) Erklären Sie die Funktion der C-Datei main.c
Wie wird P0_4 für die Ansteuerung der LED genutzt?
Hinweis: DIR0 und PIN0 Register

d) Öffnen Sie das Projekt 2b_C_TOGGLE/DIP2b.eww und betrachten Sie die Änderungen in der C-Datei main.c:

```
#include "lpc810.h"

void delay(volatile unsigned long cycles)
{
  while (cycles) {cycles--;}
}

void init(void)
{
  // LED on P0_4

  // P0_4 digital is default
  // no setting required

  // P0_4 set to output
  DIR0_bit.P0_4 = 1;

  // P0_4 set to high (LED off)
  PIN0_bit.P0_4 = 1;
```

Fortsetzung main.c:

```
 // LED on P0_2

  // P0_2 digital is NOT default
  PINENABLE0_bit.SWDIO_DISABLE = 1;

  // P0_2 set to output
  DIR0_bit.P0_2 = 1;

  // P0_2 set to low (LED on)
  PIN0_bit.P0_2 = 0;
}

void main(void)
{
  init();

  while (1)
  {
    // P0_4 toggle and P0_2 toggle
    PIN0_bit.P0_4 ^= 1;
    PIN0_bit.P0_2 ^= 1;

    delay(500000);
  }
}
```

e) Erstellen Sie die HEX-Datei, kompilieren Sie das Projekt und laden Sie die HEX-Datei -> Projektordner: /Debug/Exe

Auf dem Steckbrett sollten nun beide LEDs abwechselnd blinken.

f) Erklären Sie die Funktion der C-Datei main.c:

Wie wird die Funktion P0_2 freigeschaltet?
Hinweis: PINENABLE0 Register

Wie wird das abwechselnde Blinken der LEDs erreicht?

Table 106. Pin enable register 0 (PINENABLE0, address 0x4000 C1C0) bit description

Bit	Symbol	Value	Description	Reset value
0	ACMP_I1_EN		Enables fixed-pin function. Writing a 1 deselects the function and any movable function can be assigned to this pin. By default the fixed--pin function is deselected and GPIO is assigned to this pin.	1
		0	Enable ACMP_I1. This function is enabled on pin PIO0_0.	
		1	Disable ACMP_I1. GPIO function PIO0_0 (default) or any other movable function can be assigned to pin PIO0_0.	
1	ACMP_I2_EN		Enables fixed-pin function. Writing a 1 deselects the function and any movable function can be assigned to this pin. By default the fixed-pin function is deselected and GPIO is assigned to this pin. Functions CLKIN and ACMP_I2 are connected to the same pin PIO0_1. To use ACMP_I2, disable the CLKIN function in bit 7 of this register and enable ACMP_I2.	1
		0	Enable ACMP_I2. This function is enabled on pin PIO0_1.	
		1	Disable ACMP_I2. GPIO function PIO0_1 (default) or any other movable function can be assigned to pin PIO0_1.	
2	SWCLK_EN		Enables fixed-pin function. Writing a 1 deselects the function and any movable function can be assigned to this pin. This function is selected by default.	0
		0	Enable SWCLK. This function is enabled on pin PIO0_3.	
		1	Disable SWCLK. GPIO function PIO0_3 is selected on this pin. Any other movable function can be assigned to pin PIO0_3.	
3	SWDIO_EN		Enables fixed-pin function. Writing a 1 deselects the function and any movable function can be assigned to this pin. This function is selected by default.	0
		0	Enable SWDIO. This function is enabled on pin PIO0_2.	
		1	Disable SWDIO. GPIO function PIO0_2 is selected on this pin. Any other movable function can be assigned to pin PIO0_2.	

g) Vertiefung: Öffnen Sie das Projekt 2c_LED_C_BIT/DIP2c.eww und erstellen und laden Sie die HEX-Datei:

```
…
#define LED     P0_4

void init(void)
{
  …
  // P0_4 set to high (LED off)
  LED = 1;
}

void main(void)
{
  init();

  while (1)
  {
    // P0_4 toggle
    LED ^= 1;
    …
```

Hinweis: Der LPC810 verfügt wie der 8051 über Bit-adressierbaren Speicher (anders als Standard ARM)

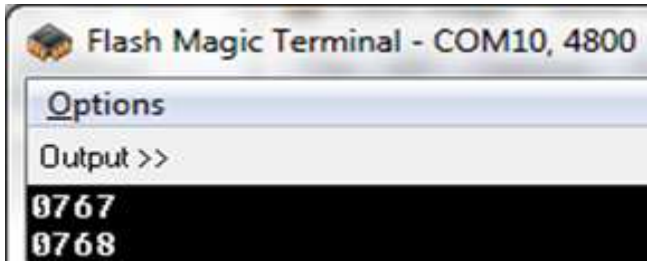
Erklären Sie die Funktion der C-Datei main.c

Wie funktioniert der „atomare Zugriff“ ? Siehe „lpc810.h“ !

6. Serielle Kommunikation

a) Öffnen Sie das Projekt 3_C_USART/DIP3.eww und betrachten Sie die C-Datei main.c:

```
#define CLK             12000000
#define BAUD            4800
…
void init(void) {
  …
  // P0_4 as U0_TXD and P0_0 as U0_RXD
  // (same as ISP bootloader)
  PINASSIGN0_bit.U0_TXD = 4;
  PINASSIGN0_bit.U0_RXD = 0;
  // UART0 init
  uart0Init(BAUD, CLK);
}
```



b) Wie werden Pins für TXD und RXD freigeschaltet?

c) Erstellen Sie die HEX-Datei, kompilieren Sie das Projekt und laden Sie die HEX-Datei -> Projektordner: /Debug/Exe

d) Öffnen Sie in FlashMagic ein Terminal mit den geeigneten Einstellungen: Tools->Terminal / Im Terminal sollte nun der Counter laufen.

e) Betrachten Sie nun die C-Datei uart.c:

```
void uart0Init(unsigned long baudRate, unsigned long clk) {
  …
  const unsigned long div = 1, frgdiv = 0xFF;
  …
  sclk = clk/div;
  UARTCFG_bit.DATALEN = 1;
  UARTCFG_bit.PARITYSEL = 0;
  UARTCFG_bit.STOPLEN = 0;

  brgval = sclk / 16 / baudRate - 1;
  UARTBRG_bit.BRGVAL = brgval;          // baud rate generator value
  UARTFRGDIV_bit.DIV = frgdiv;          // baud rate generator divider

  UARTFRGMULT_bit.MULT = (((sclk / 16) * (frgdiv + 1)) /
    (baudRate * (brgval + 1))) - (frgdiv + 1);
  …
  // Start USART0
  UARTCFG_bit.ENABLE = 1;
}

void uart0SendChar(char buffer) {
  // Wait until ready to send
  while (!(UARTSTAT_bit.TXRDY));

  UARTTXDAT_bit.TXDAT = buffer;
}
```

f) Wie erfolgt die Berechnung der Parameter aus der Baudrate?

Hinweis:

sclk = 12000000/1 = 12000000
brgval = 12000000 / 16 / 4800 – 1 = 155
frgdiv = 0xFF = 255

UARTFRGMULT =
(((12000000 / 16) * (255 + 1)) / (4800 * (155 + 1))) - (255 + 1) =
(750000*256)/(4800*156) - 256 =
0

Baudrate 4800 wird ohne Fraktionieren erreicht: UARTFRGMULT + 1

g) Wie funktioniert das serielle Senden (Funktion: `uart0SendChar`)?

h) Ändern Sie das Programm so ab dass eine Baudrate von 115200 erreicht wird und testen Sie mit dem Flash Magic Terminal.

Table 174. USART Configuration register (CFG, address 0x4006 4000 (USART0), 0x4006 8000 (USART1), 0x4006 C000 (USART2)) bit description

Bit	Symbol	Value	Description	Reset Value
0	ENABLE		USART Enable.	0
		0	Disabled. The USART is disabled and the internal state machine and counters are reset. While Enable = 0, all USART interrupts are disabled. When Enable is set again, CFG and most other control bits remain unchanged. For instance, when re-enabled, the USART will immediately generate a TxRdy interrupt (if enabled in the INTENSET register) because the transmitter has been reset and is therefore available.	
		1	Enabled. The USART is enabled for operation.	
1	-		Reserved. Read value is undefined, only zero should be written.	NA
3:2	DATALEN		Selects the data size for the USART.	00
		0x0	7 bit Data length.	
		0x1	8 bit Data length.	
		0x2	9 bit data length. The 9th bit is commonly used for addressing in multidrop mode. See the ADDRDET bit in the CTL register.	
		0x3	Reserved.	
5:4	PARITYSEL		Selects what type of parity is used by the USART.	00
		0x0	No parity.	
		0x1	Reserved.	
		0x2	Even parity. Adds a bit to each character such that the number of 1s in a transmitted character is even, and the number of 1s in a received character is expected to be even.	
		0x3	Odd parity. Adds a bit to each character such that the number of 1s in a transmitted character is odd, and the number of 1s in a received character is expected to be odd.	

Table 182. USART Baud Rate Generator register (BRG, address 0x4006 4020 (USART0), 0x4006 8020 (USART1), 0x4006 C020 (USART2)) bit description

Bit	Symbol	Description	Reset Value
15:0	BRGVAL	This value is used to divide the USART input clock to determine the baud rate, based on the input clock from the FRG. 0 = The FRG clock is used directly by the USART function. 1 = The FRG clock is divided by 2 before use by the USART function. 2 = The FRG clock is divided by 3 before use by the USART function. ...	0

Table 181. USART Transmitter Data Register (TXDAT, address 0x4006 401C (USART0), 0x4006 801C (USART1), 0x4006 C01C (USART2)) bit description

Bit	Symbol	Description	Reset Value
8:0	TXDAT	Writing to the USART Transmit Data Register causes the data to be transmitted as soon as the transmit shift register is available and any conditions for transmitting data are met: CTS low (if CTSEN bit = 1), TXDIS bit = 0.	0

7. Timer

Das Multi-Rate-Timer (MRT) Modul bietet 4 unabhängige Timer:

a) Öffnen Sie das Projekt 4a_C_TIMER_POLLING/DIP4a.eww und betrachten Sie die C-Datei main.c:

```
#define CLK     12000000                // CLK 12 MHz

#define DELAY   (CLK/1000)*500          // MRT 0.5s

void init(void)
{
  // MRT
  SYSAHBCLKCTRL_bit.MRT = 1;            // MRT clock enable
  PRESETCTRL_bit.MRT_RESET_LOW = 0;     // MRT reset
  PRESETCTRL_bit.MRT_RESET_LOW = 1;

  // MRT channel 0
  MRTCTRL0_bit.MODE = 1;                // MRT single mode
  MRTINTVAL0_bit.IVALUE = DELAY;        // MRT start

  // LED on P0_4
  DIR0_bit.P0_4 = 1;                    // P0_4 set to output

  PIN0_bit.P0_4 = 1;                    // P0_4 set to high (LED off)
}
```

Schreiben des Anfangswerts startet Timer

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

  while (1)
  {
    // MRT channel 0
    if (MRTSTAT0_bit.RUN == 0)          // MRT has stopped ?
    {
      PIN0_bit.P0_4 ^= 1;               // P0_4 toggle
      MRTINTVAL0_bit.IVALUE = DELAY;    // MRT re-start
    }
  }
}
```

b) Erklären Sie die Funktion der C-Datei main.c
Was ist „single mode“ ?

c) Ändern Sie das Programm so ab, dass die LED schneller oder langsamer blinkt.

d) Ändern Sie das Programm so ab, dass zum Polling nicht geprüft wird, ob der Timer gestoppt hat, sondern ob der Timer abgelaufen ist (d.h. den Wert 0 erreicht hat). Hinweis: `MRTTIMER0_bit.VALUE`

Table 152. Control register (CTRL[0:3], address 0x4000 4008 (CTRL0) to 0x4000 4038 (CTRL3)) bit description

Bit	Symbol	Value	Description	Reset value
0	INTEN		Enable the TIMERn interrupt.	0
		0	Disable.	
		1	Enable.	
2:1	MODE		Selects timer mode.	0
		0x0	Repeat interrupt mode.	
		0x1	One-shot interrupt mode.	

Table 153. Status register (STAT[0:3], address 0x4000 400C (STAT0) to 0x4000 403C (STAT3)) bit description

Bit	Symbol	Value	Description	Reset value
0	INTFLAG		Monitors the interrupt flag.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMERn has reached the end of the time interval. If the INTEN bit in the CONTROLn is also set to 1, the interrupt for timer channel n and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
1	RUN		Indicates the state of TIMERn. This bit is read-only.	0
		0	Idle state. TIMERn is stopped.	
		1	Running. TIMERn is running.	

e) Öffnen Sie das Projekt 4b_C_TIMER_IRQ/DIP4b.eww und betrachten Sie die C-Datei main.c:

```
#define CLK     12000000                // CLK 12 MHz

#define DELAY   (CLK/1000)*500          // MRT 0.5s

void MRT_IRQHandler(void)
{
  if (MRTSTAT0_bit.INTFLAG)             // MRT channel 0
  {
    PIN0_bit.P0_4 ^= 1;                 // P0_4 toggle
    MRTSTAT0_bit.INTFLAG = 1;           // CAUTION: =1 clears interrupt flag
  }
}

void init(void)
{
  // MRT
  SYSAHBCLKCTRL_bit.MRT = 1;            // MRT clock enable
  PRESETCTRL_bit.MRT_RESET_LOW = 0;     // MRT reset
  PRESETCTRL_bit.MRT_RESET_LOW = 1;

  // MRT channel 0
  MRTCTRL0_bit.MODE = 0;                // MRT repeated mode
  MRTCTRL0_bit.INTEN = 1;               // MRT irq enable
```

in „cstartup_M.s“ definiert

```
  NVICISER0_bit.ISE_MRT = 1;            // NVIC MRT irq enable

  MRTINTVAL0_bit.IVALUE = DELAY;        // MRT start

  // LED on P0_4
  DIR0_bit.P0_4 = 1;                    // P0_4 set to output

  PIN0_bit.P0_4 = 1;                    // P0_4 set to high (LED off)
}

void main(void)
{
  init();

  while (1);
}
```

warten auf IRQ

f) Erklären Sie die Funktion der C-Datei main.c
Was ist „repeated mode“ ?
Wie ist die Funktion des Interrupt-Controllers NVIC ?

g) Ändern Sie das Programm so ab, dass die LED schneller oder langsamer blinkt.

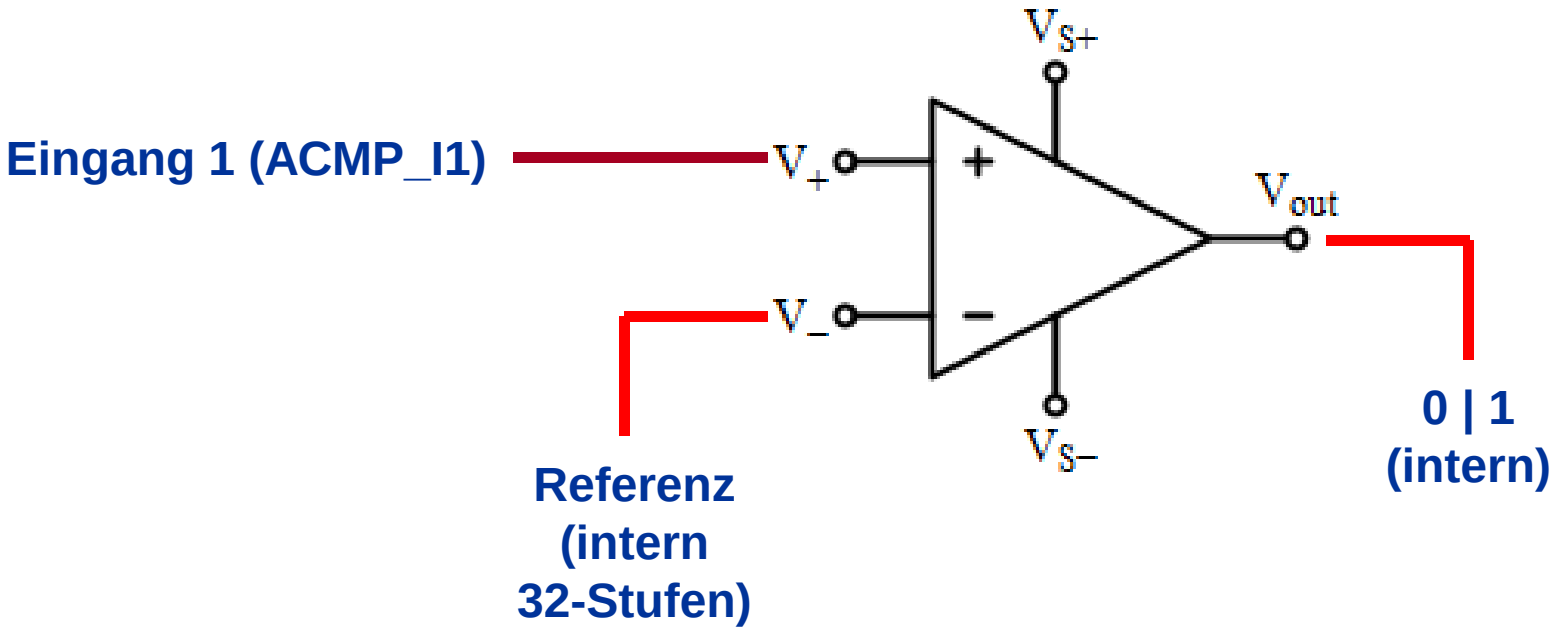
Table 5. Interrupt Set Enable Register 0 register (ISER0, address 0xE000 E100) bit description

Bit	Symbol	Description	Reset value
0	ISE_SPI0	Interrupt enable.	0
1	ISE_SPI1	Interrupt enable.	0
2	-	Reserved.	-
3	ISE_UART0	Interrupt enable.	0
4	ISE_UART1	Interrupt enable.	0
5	ISE_UART2	Interrupt enable.	0
6	-	Reserved.	-
7	-	Reserved.	-
8	ISE_I2C	Interrupt enable.	0
9	ISE_SCT	Interrupt enable.	0
10	ISE_MRT	Interrupt enable.	0
11	ISE_CMP	Interrupt enable.	0

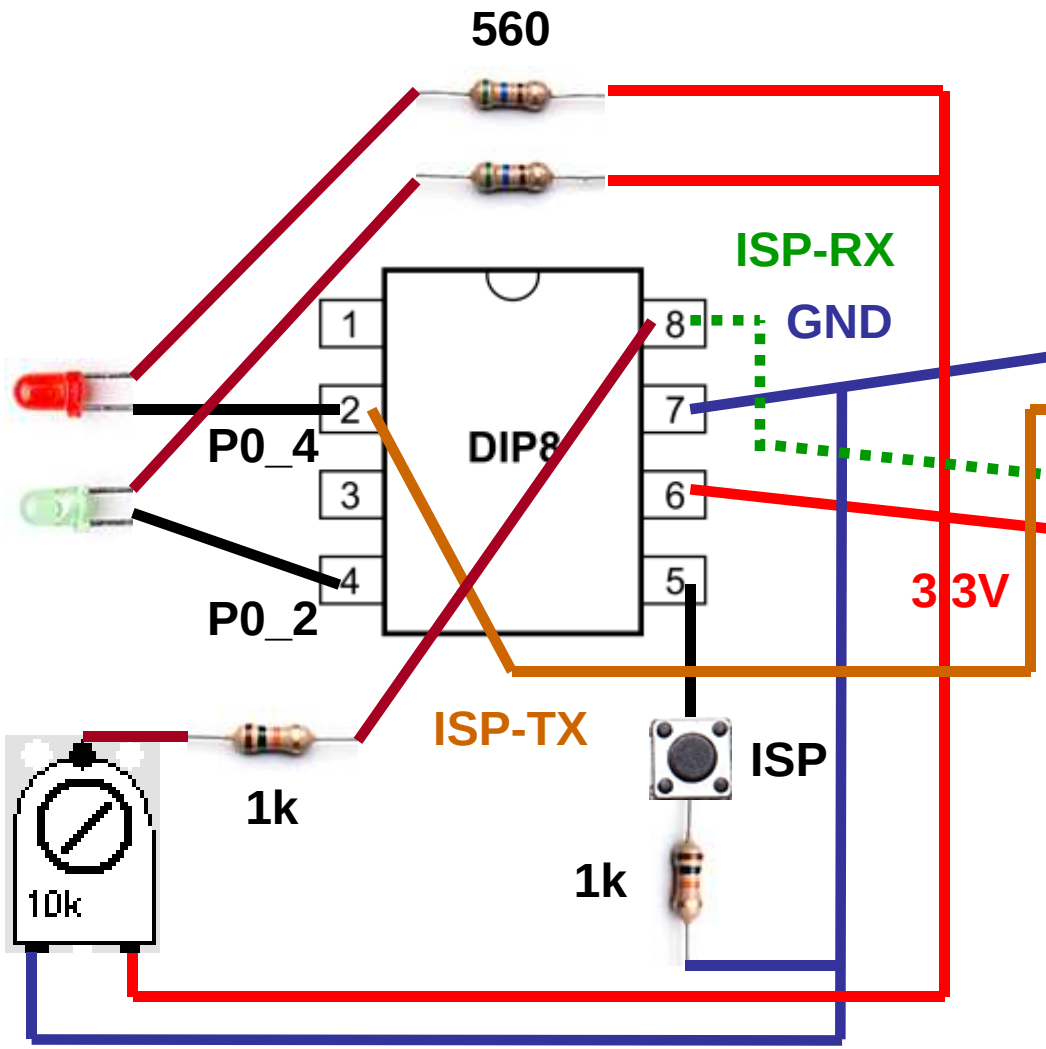
8. Analog-Komparator

Der Komparator vergleicht zwei analoge Spannungen im Bereich 0V – 3.3V

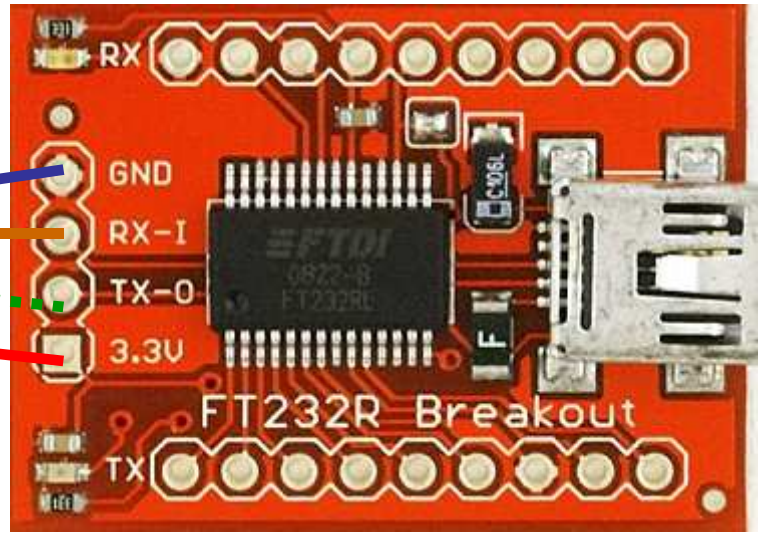
Der Komparator wird nun wie folgt beschaltet:



Schaltplan in Farbe drucken !



Hinweis:
Eingang 1 (ACMP_I1)
liegt auf P0.0 (Pin 8)



Hinweis:
ISP-RX stört als digitaler Eingang den Analog-Komparator und ist nach dem Flashen abzuziehen.

Hochschule Ravensburg-Weingarten

a) Öffnen Sie das Projekt 5_C_COMP/DIP5.eww und betrachten Sie die C-Datei main.c:

```
#define VOLTMAX 31      // 31=VDD=3.3V

void main(void)
{
  COMP_init(VOLTMAX/2);         // trigger at 3.3V/2=1.65V !

  while (1)
  {
    // COMP <> threshold ?
    if (COMPCTRL_bit.COMPSTAT)
      PIN0_bit.P0_4 = 0;        // LED on
    else
      PIN0_bit.P0_4 = 1;        // LED off
  }
}
```

b) Was geschieht beim Überschreiten des Schwellwerts?

c) Erstellen Sie die HEX-Datei, kompilieren Sie das Projekt und laden Sie die HEX-Datei -> Projektordner: /Debug/Exe

Der LPC810 verfügt über keinen Analog-Digital-Konverter (ADC). Allerdings kann der Analog-Komparator mittels der internen 32-Stufen-Referenz als 5-bit ADC genutzt werden.

d) Öffnen Sie das Projekt 6_C_ADC/DIP6.eww und betrachten Sie die C-Datei main.c:

```
#define VOLTMAX 31      // 31=VDD=3.3V
#define DELAY   750000

void main(void)
{
  COMP_init(VOLTMAX);           // 31=VDD=3.3V

  while (1)
  {
    unsigned char n;
    unsigned int d = 1;

    // check voltage from 3.3V to 0V
    for (n=VOLTMAX; n>0; n--)
    {
      // COMP voltage ladder value change
      COMPLAD_bit.LADSEL = n;
```

Fortsetzung main.c:

```
          // COMP sees current voltage ?
          if (COMPCTRL_bit.COMPSTAT)
          {
            d = n;
            break;
          }
        }

        // LED toggle
        PIN0_bit.P0_4 ^= 1;

        // 3.3V fast and 0V slow
        delay((unsigned int) (DELAY/d));
      }
    }
```

e) Wie wird die anliegende analoge Spannung bestimmt und angezeigt?
Hinweis: `DELAY/d`

f) Erstellen Sie die HEX-Datei, kompilieren Sie das Projekt und laden Sie die HEX-Datei -> Projektordner: /Debug/Exe

Table 219. Comparator control register (CTRL, address 0x4002 4000) bit description

Bit	Symbol	Value	Description	Reset value
10:8	COMP_VP_SEL		Selects positive voltage input	0
		0x0	Voltage ladder output	
		0x1	ACMP_I1	
		0x2	ACMP_I2	
		0x3	Reserved	
		0x4	Reserved	
		0x5	Reserved	
		0x6	Internal reference voltage (bandgap)	
		0x7	Reserved	
21	COMPSTAT		Comparator status. This bit reflects the state of the comparator output.	0

Table 220. Voltage ladder register (LAD, address 0x4002 4004) bit description

Bit	Symbol	Value	Description	Reset value
0	LADEN		Voltage ladder enable	0
5:1	LADSEL		Voltage ladder value. The reference voltage Vref depends on the LADREF bit below. 00000 = V_{SS} 00001 = 1 × Vref/31 00010 = 2 × Vref/31 ... 11111 = Vref	0

Anhang • Verwendung von LPC812 SIP16 anstelle von LPC810 DIP8