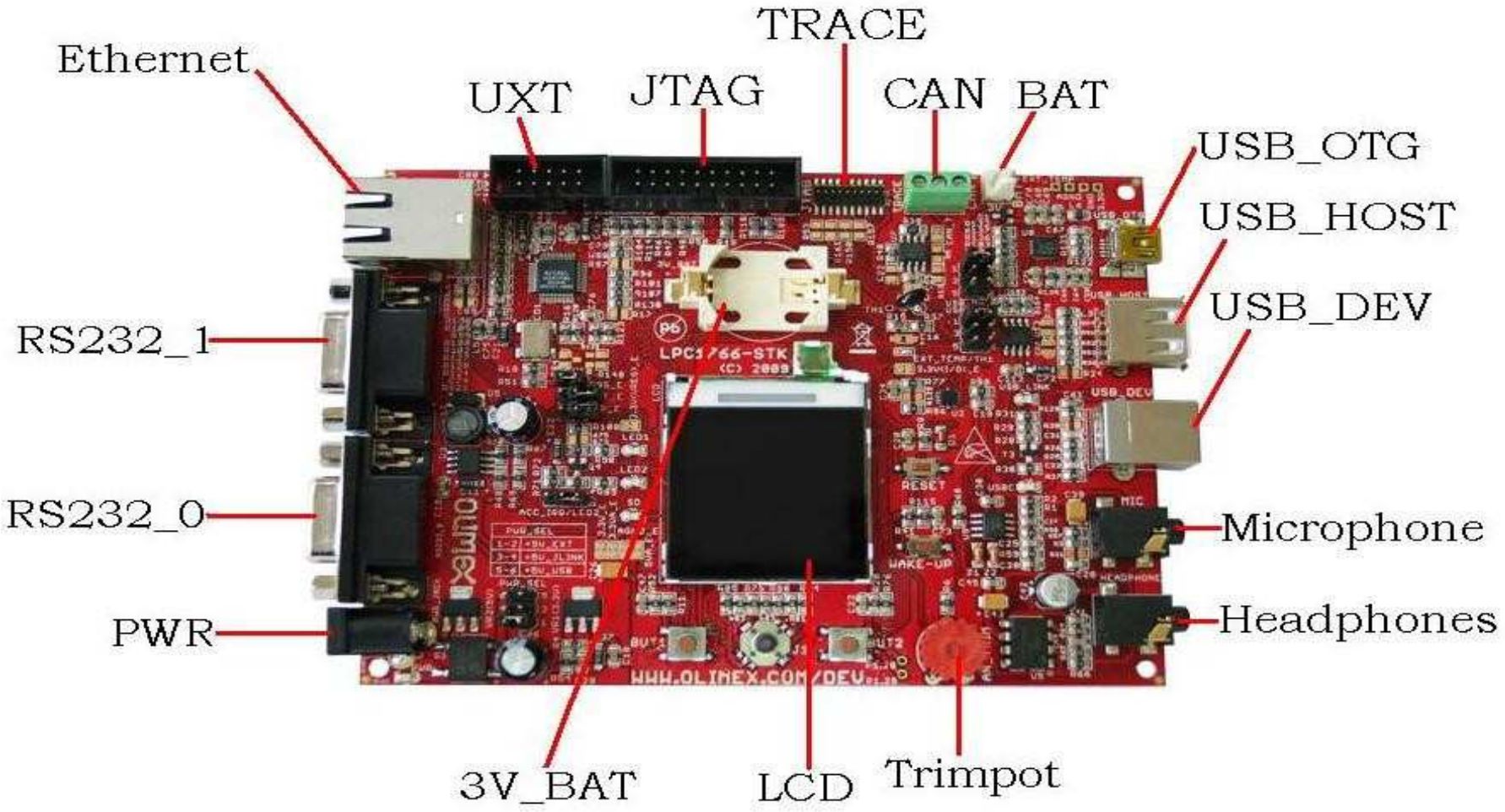
1. Einstieg in ein ARM-Entwicklungsboard

In diesem Versuch wird ein ARM-Entwicklungsboard mit dem Cortex-M3 Prozessor LPC1766 von NXP vorgestellt:

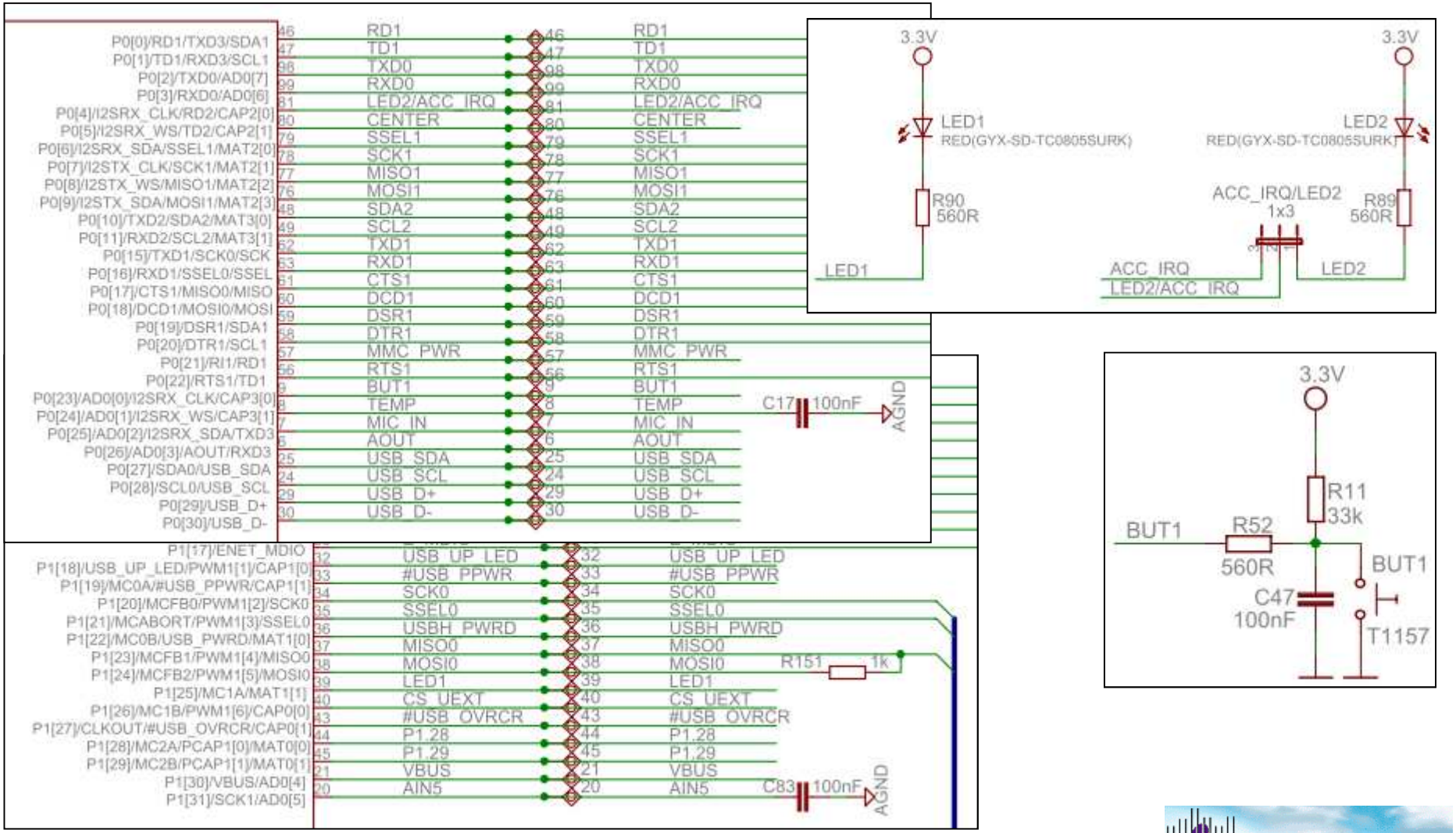
- 100 MHz, 256KB Flash, 64KB RAM
- 4 UART, USB, Ethernet
- CAN, SPI, I2C
- ADC, DAC
- 2 LEDs, 3 Taster, Poti
- LCD Display Nokia 6610 (128x128 12bit Color)
- 3-Axis Digital Accelerometer (11bit)
- Temperatur-Sensor
- Audio I/O
- SD/MMC Card

a) Vergleichen Sie die Eigenschaften von LPC1766 (Cortex-M3) und LPC810 (Cortex-M0)

b) Verschaffen Sie sich einen Überblick über das ARM-Board


TRACE
Ethernet
UXT
JTAG
CAN
BAT
USB_OTG
USB_HOST
USB_DEV
RS232_1
RS232_0
Microphone
PWR
Headphones
3V_BAT
LCD
Trimpot

c) An welche Pins sind die LEDs 1 und 2 und der BUTTON 1 angeschlossen?



2. Speicherorganisation des ARM-Entwicklungsboard

a) Memory Map

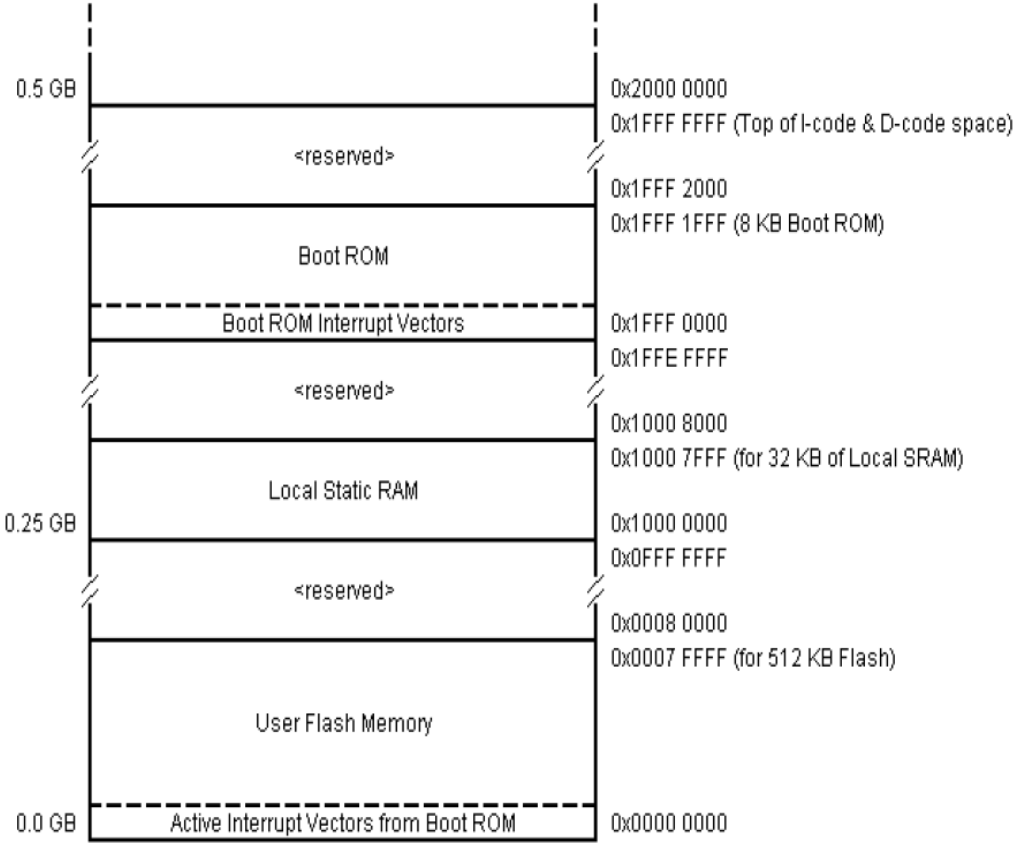
Flash (Programmcode)

RAM (Variablen, Peripherie)

Boot ROM (Bootloader)

Boot ROM Interrupt Vectors:
Hardware-Interruptadressen

Active Interrupt Vectors from BootROM:
nach Reset aktive Interrupt-Vektortabelle



b) Linker: Flash.icf

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

c) Bootloader

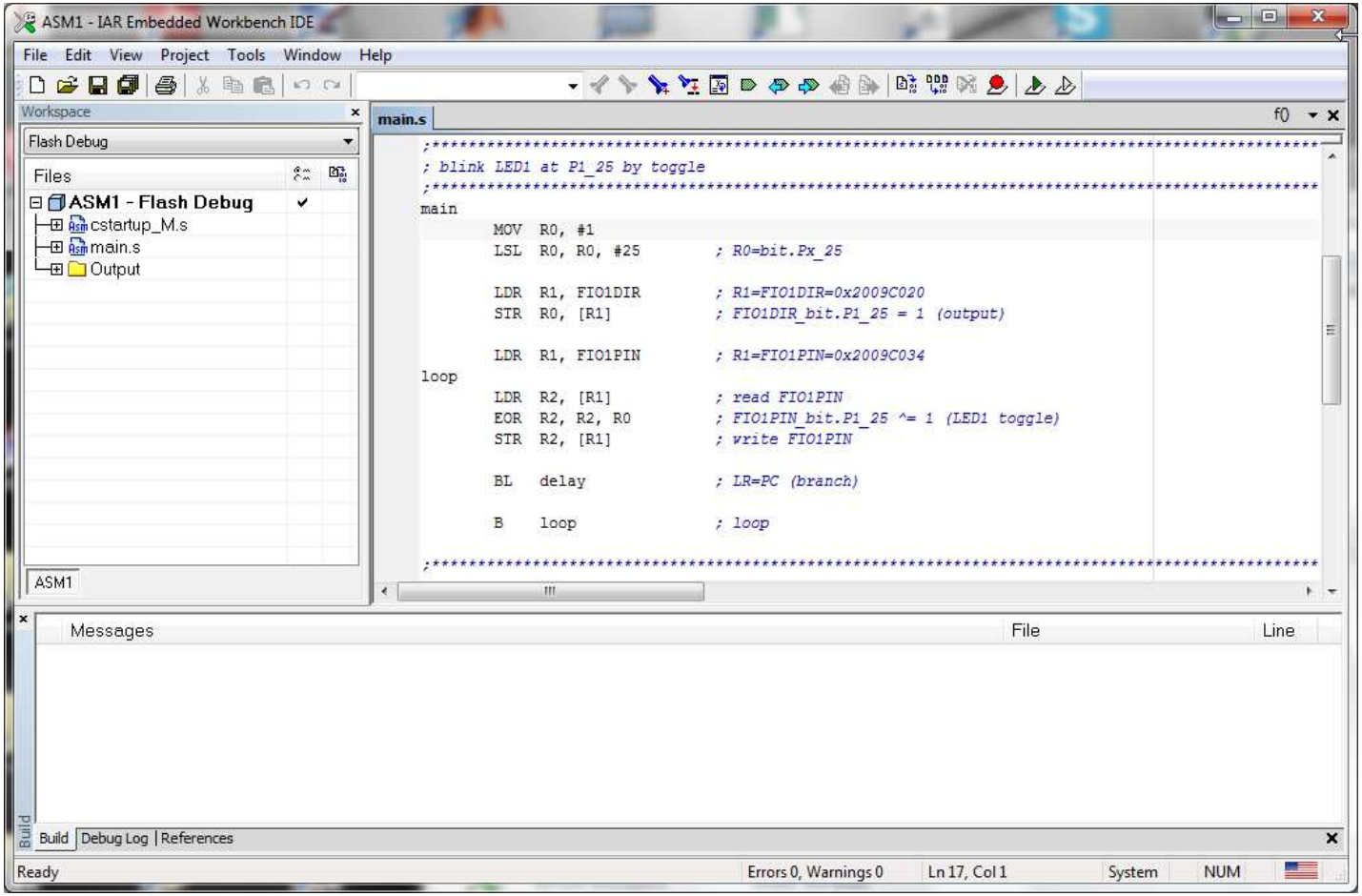
Der Bootloader im Boot ROM ist ein kleines "Betriebssystem", womit der Programmcode in Flash oder RAM geladen werden kann. Bei Reset wird der Programmcode dann dort ausgeführt.

Standard:
UART-Bootloader: Programmcode über serielle Schnittstelle

Optional:
USB-Bootloader
Ethernet-Bootloader
CAN-Bootloader

3. Einstieg in ARM Assembler

IAR-Entwicklungssystem:



a) Öffnen Sie das Projekt 1_ASM_TOGGLE/ASM1.eww und betrachten Sie die Assembler-Datei main.s:

```
        MODULE  ?main
        PUBLIC  __iar_program_start

        SECTION .text:CODE(2)
        CODE

__iar_program_start

main
        MOV  R0, #1
        LSL  R0, R0, #25        ; R0=bit.Px_25

        LDR  R1, FIO1DIR        ; R1=FIO1DIR=0x2009C020
        STR  R0, [R1]           ; FIO1DIR_bit.P1_25 = 1 (output)

        LDR  R1, FIO1PIN        ; R1=FIO1PIN=0x2009C034
loop
        LDR  R2, [R1]           ; read FIO1PIN
        EOR  R2, R2, R0         ; FIO1PIN_bit.P1_25 ^= 1 (LED1 toggle)
        STR  R2, [R1]           ; write FIO1PIN

        BL   delay              ; LR=PC (branch)

        B    loop               ; loop
```

Fortsetzung main.s:

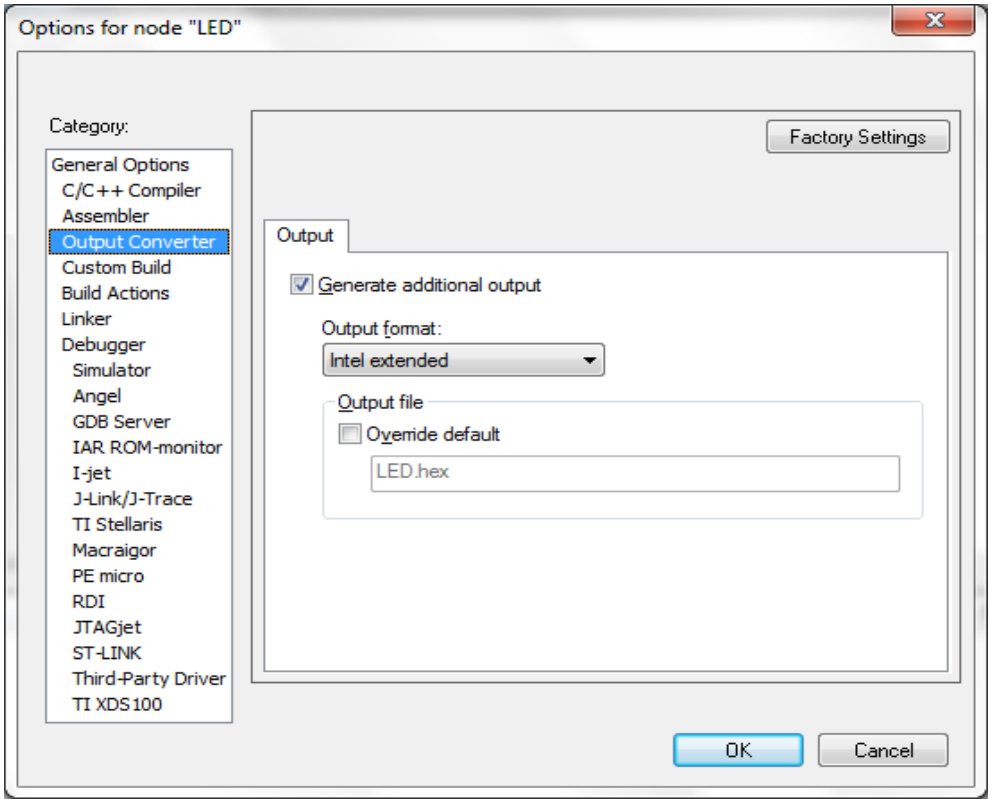
```
; delay about 0.25s : 1/4MHz*2^18*2*2

delay
        PUSH {R0}
        MOV R0, #1
        LSL R0, R0, #18         ; 2^18=262144
count
        SUBS R0, R0, #1
        BNE count               ; <>0
        POP  {R0}
        MOV  PC, LR             ; PC=LR (return)

; constants

        SECTION .text:CODE(2)
        DATA

FIO1DIR
        DC32 0x2009C020
FIO1PIN
        DC32 0x2009C034

        END
```

b) Erstellen Sie die HEX-Datei:

Output Converter -> HEX

c) Kompilieren Sie das Projekt:

Project->Rebuild All

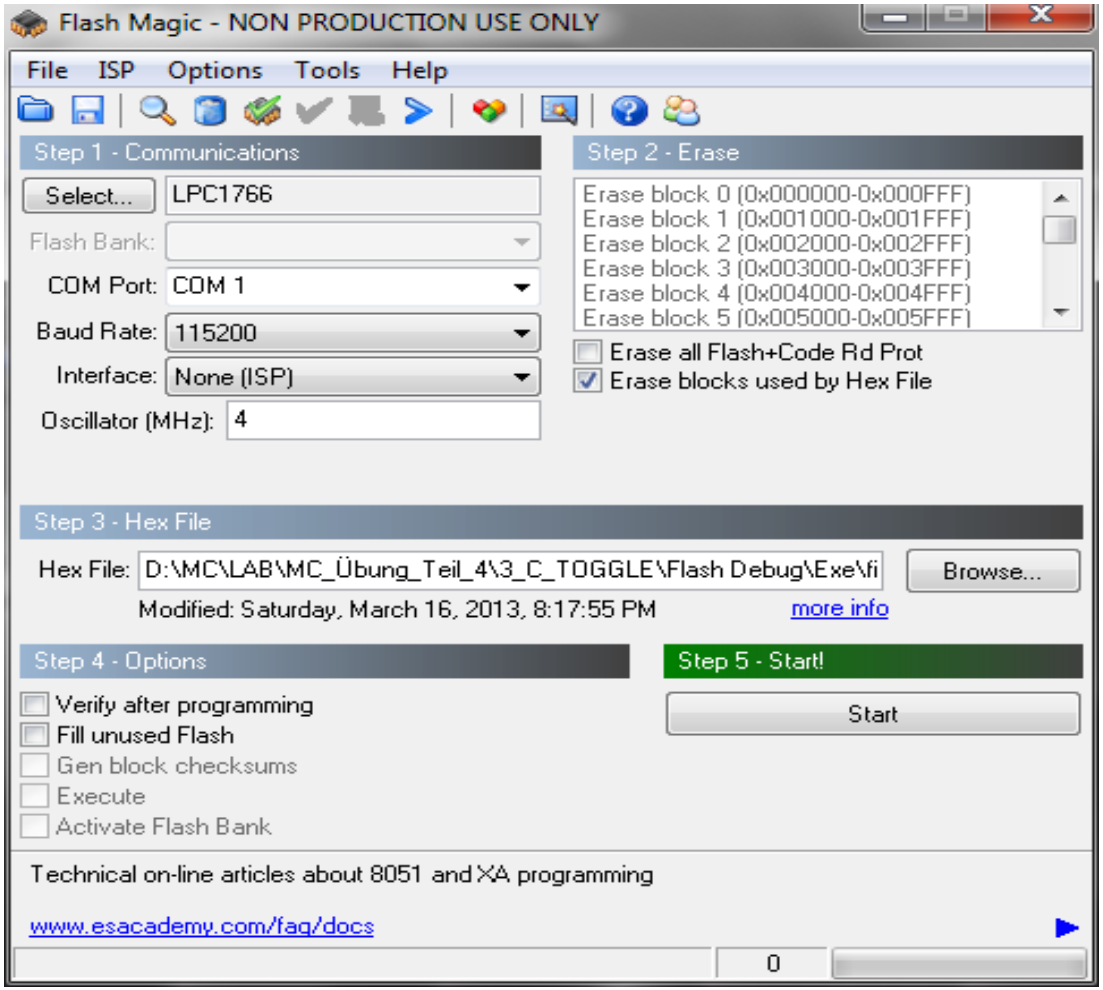


Hinweis:
In der Assembler-Datei main.s wird die Clock CLK nicht initialisiert. Somit läuft der Mikrocontroller mit dem internen RC-Oszillator mit 4 MHz (und somit nicht mit 100 MHz)!

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.



b) Schließen Sie folgende Kabel an das Board an:
   -> Stromversorgung 9V (mind. 6V)
   -> serielles Kabel an RS232_0

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

d) Laden Sie die HEX-Datei:
   -> Projektordner: /Flash Debug/Exe

Auf dem Board sollte nun die LED 1 blinken.

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

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

5. Programmieren in ARM Assembler

a) Öffnen Sie das Projekt 2_ASM_BUTTON/ASM2.eww und betrachten Sie die Änderungen in der Assembler-Datei main.s:

```
main
        …

        MOV  R3, #1
        LSL  R3, R3, #23        ; R3=bit.Px_23

        LDR  R4, FIO0PIN        ; R4=FIO0PIN=0x2009C014
loop
        LDR  R2, [R4]           ; read FIO0PIN
        TSTS R2, R3             ; test FIO0PIN_bit.P0_23
        BNE  over
                                ; =0
        LDR  R2, [R1]           ; read FIO1PIN
        EOR  R2, R2, R0         ; FIO1PIN_bit.P1_25 ^= 1 (LED1 toggle)
        STR  R2, [R1]           ; write FIO1PIN

        BL   delay              ; LR=PC (branch)
over
        B    loop               ; loop
```

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

Auf dem Board sollte nun ein Drücken von BUTTON 1 die LED 1 toggeln.

c) Erklären Sie die Funktion der Assembler-Datei main.s:
Wie wird BUTTON 1 abgefragt?

Hinweis:
BUTTON 1 ist über einen Pull-Up Widerstand mit dem Pin verbunden. Somit ist der Default Pin-Mode %00 geeignet und muss somit nicht initialisiert werden. Bei einem Pull-Down Widerstand müsste der Pin-Mode %11 (oder Open-Drain %10) initialisiert werden!

P0.00MODE		Port 0 pin 0 on-chip pull-up/down resistor control.
	00	P0.0 pin has a pull-up resistor enabled.
	01	P0.0 pin has repeater mode enabled.
	10	P0.0 pin has neither pull-up nor pull-down.
	11	P0.0 has a pull-down resistor enabled.

6. Einstieg in ARM C

a) Öffnen Sie das Projekt 3_C_TOGGLE/CPROG1.eww und betrachten Sie die C-Datei main.c:

```
#include <nxp/iolpc1766.h>

void Delay(volatile unsigned long i) { while (i!=0) i--; }

// blink LED1 at P1_25 by toggle

int main (void)
{
  FIO1DIR_bit.P1_25 = 1;        // output

  while (1)
  {
      FIO1PIN_bit.P1_25 ^= 1;   // toggle
      Delay(1<<18);
  }
  return 0;
}
```

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

c) Öffnen Sie das Projekt 4_C_BUTTON/CPROG2.eww und betrachten Sie die Änderungen in der C-Datei main.c:

```
// toggle LED1 at P1_25 by BUTTON1 at P0_23
int main (void)
{
  FIO1DIR_bit.P1_25 = 1;        // output

  while (1)
  {
    if (FIO0PIN_bit.P0_23 == 0) // check =0
    {
      FIO1PIN_bit.P1_25 ^= 1;   // toggle
      Delay(1<<18);
    }
  }
  return 0;
}
```

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

e) Wie wird BUTTON 1 abgefragt?

7. Programmieren in ARM C: Timer und Interrupts

a) Öffnen Sie Projekt 5a_C_TIMER/TIMER1.eww – betrachten Sie die C-Datei main.c:

```
// blink LED1 at P1_25 by toggle
void TMR0_IRQHandler(void)
{
  FIO1PIN_bit.P1_25 ^= 1;        // toggle

  T0IR_bit.MR0INT = 1;           // timer 0 irq clear irq flag
  CLRPEND0 |= 1<<(TMR0_IRQ);     // timer 0 irq clear irq (NVIC)
}

int main(void)
{
  FIO1DIR_bit.P1_25 = 1;         // output
  TMR0_Init();

  while (1)                      // wait for irq
  {
  }
  return 0;
}
```

Fortsetzung main.c:

```
#define CLK             4000000         // 4 MHz
#define PCLK            (CLK/4)         // 1 MHz

#define TMR0_IRQ        1       // irq number
void TMR0_Init(void) {
  __disable_interrupt();

  PCONP_bit.PCTIM0 = 1;         // PCLK to timer 0

  T0TCR_bit.CE = 0;             // timer 0 stop
  T0TCR_bit.CR = 1;             // set reset
  T0TCR_bit.CR = 0;             // release reset
  T0CTCR_bit.CTM = 0;           // timer 0 mode: every rising PCLK edge
  T0MCR_bit.MR0I = 1;           // enable irq on MR0
  T0MCR_bit.MR0R = 1;           // enable reset on MR0
  T0MCR_bit.MR0S = 0;           // disable stop on MR0

  T0PR = 0;                     // no prescale
  T0MR0 = PCLK/4;               // period =0.25s

  T0IR_bit.MR0INT = 1;          // timer 0 irq clear pending
  SETENA0 |= 1<<(TMR0_IRQ);     // timer 0 irq enable (NVIC)

  T0TCR_bit.CE = 1;             // timer 0 start

  __enable_interrupt();
}
```

b) Betrachten Sie die Vektortabelle in cstartup_M.s:

```
__vector_table
        DCD     sfe(CSTACK)                ; Top of Stack
        DCD     __iar_program_start        ; Reset Handler
        DCD     NMI_Handler                ; NMI Handler
        DCD     HardFault_Handler          ; Hard Fault Handler
        DCD     MemManage_Handler          ; MPU Fault Handler
        DCD     BusFault_Handler           ; Bus Fault Handler
        DCD     UsageFault_Handler         ; Usage Fault Handler
__vector_table_0x1c
        DCD     0                          ; Reserved
        DCD     0                          ; Reserved
        DCD     0                          ; Reserved
        DCD     0                          ; Reserved
        DCD     SVC_Handler                ; SVCall Handler
        DCD     DebugMon_Handler           ; Debug Monitor Handler
        DCD     0                          ; Reserved
        DCD     PendSV_Handler             ; PendSV Handler
        DCD     SysTick_Handler            ; SysTick Handler
        DCD     WDT_IRQHandler             ; Watchdog Handler
        DCD     TMR0_IRQHandler            ; Timer 0 Handler
```

Wo befindet sich die Adresse der Timer 0 Interrupt Funktion?

c) Vergleichen Sie die Timer-Initialisierung mit dem LPC1766 User Manual:

Table 46. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1

Table 429. Count Control Register (T[0/1/2/3]CTCR - addresses 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	Counter/ Timer Mode		This field selects which rising PCLK edges can increment the Timer's Prescale Counter (PC), or clear the PC and increment the Timer Counter (TC).	00
		00	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register. The Prescale Counter is incremented on every rising PCLK edge.	
		01	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		10	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		11	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	

Table 429. Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014) bit description

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	

Prescale register (T0PR - T3PR, 0x4000 400C, 0x4000 800C, 0x4009 000C, 0x4009 400C)

The 32-bit Prescale register specifies the maximum value for the Prescale Counter.

Match Registers (MR0 - MR3)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

Table 427. Timer Control Register (TCR, TIMERn: TnTCR - addresses 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004) bit description

Bit	Symbol	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When 1, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0

Table 426. Interrupt Register (T[0/1/2/3]IR - addresses 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000) bit description

Bit	Symbol	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0

d) Welche Clock nutzt der Timer? Hinweis: CLK vs. PCLK!

e) Welche Funktion hat das Timer Prescale Register?

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

g) Nun sollen die LED 1 und die LED 2 abwechselnd blinken. Welche Programmänderung ist hierfür erforderlich?

Lösung: Projekt 5b_C_TIMER/TIMER2.eww

h) Welche Programmzeile muss abgeändert werden, um die LEDs schneller oder langsamer blinken zu lassen?

8. Programmieren in ARM C: Externe Interrupts

a) Öffnen Sie Projekt 6_C_EXTERN/EXTERN.eww – betrachten Sie die C-Datei main.c:

```
void EINT3_IRQHandler(void)
{
  FIO1PIN_bit.P1_25 ^= 1;        // LED1 toggle
  FIO0PIN_bit.P0_4 ^= 1;         // LED2 toggle

  IO0INTCLR_bit.P0_23 = 1;       // BUTTON1 ext irq clear flag
  CLRPEND0 |= 1<<(EINT3_IRQ);    // ext irq all port pins clear (NVIC)
}

int main(void)
{
  FIO1DIR_bit.P1_25 = 1;         // LED1 output
  FIO0DIR_bit.P0_4 = 1;          // LED2 output
  FIO0PIN_bit.P0_4 = 1;          // LED2 off
  EINT3_Init();

  while (1)                      // wait for irq
  { }
  return 0;
}
```

Fortsetzung main.c:

```
//*********************************************************
// special (fast) irq pins: EINT0, EINT1, EINT2, EINT3
// general (slow) irq pins: all port pins via EINT3
//*********************************************************

#define EINT3_IRQ       21

void EINT3_Init(void)
{
  __disable_interrupt();

  IO0INTENF_bit.P0_23 = 1;      // BUTTON1 ext irq on falling edge
                                // (= button press)
  IO0INTENR_bit.P0_23 = 1;      // BUTTON1 ext irq on rising edge
                                // (= button release)
  SETENA0 |= 1<<(EINT3_IRQ);    // ext irq all port pins enable (NVIC)

  __enable_interrupt();
}
```

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

Auf dem Board sollte nun ein Drücken von BUTTON 1 von LED 1 auf LED2 umschalten.

c) Erklären Sie die Interrupt-Funktion EINT3_IRQHandler():

Wie wird BUTTON 1 abgefragt?

Wie werden Falling Edge und Rising Edge durch Drücken und Loslassen von BUTTON1 ausgelöst?

Pin name	Pin direction	Pin description
EINT0	Input	**External Interrupt Input 0** - An active low/high level or falling/rising edge general purpose interrupt input. This pin may be used to wake up the processor from Sleep, Deep-sleep, or Power-down modes.
EINT1	Input	**External Interrupt Input 1** - See the EINT0 description above.
EINT2	Input	**External Interrupt Input 2** - See the EINT0 description above.
EINT3	Input	**External Interrupt Input 3** - See the EINT0 description above.

Table 50. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
18	34	0x88	External Interrupt	External Interrupt 0 (EINT0)
19	35	0x8C	External Interrupt	External Interrupt 1 (EINT1)
20	36	0x90	External Interrupt	External Interrupt 2 (EINT2)
21	37	0x94	External Interrupt	External Interrupt 3 (EINT3). **Note:** EINT3 channel is shared with GPIO interrupts

Table 52. Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)

Bit	Name	Function
18	ISE_EINT0	External Interrupt 0 Interrupt Enable. See functional description for bit 0.
19	ISE_EINT1	External Interrupt 1 Interrupt Enable. See functional description for bit 0.
20	ISE_EINT2	External Interrupt 2 Interrupt Enable. See functional description for bit 0.
21	ISE_EINT3	External Interrupt 3 Interrupt Enable. See functional description for bit 0.

9. Programmieren in ARM C: Analog-Digital-Konvertierung

a) Welche Bit-Auflösung besitzt der ADC des LPC1766?

b) Berechnen Sie anhand der Bit-Auflösung und Versorgungsspannung die maximale Spannungs-Auflösung.

c) An welchen ADC-Pin ist das Potentiometer angeschlossen?

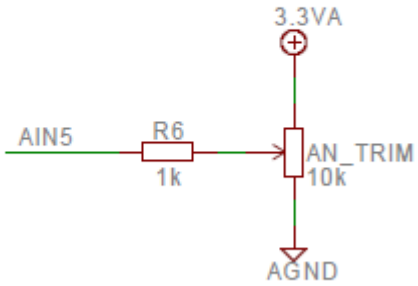
d) Öffnen Sie das Projekt: 7a_ADC_TASK/ADC1.eww
Betrachten Sie den Quellcode. Erzeugen Sie die HEX-Datei und laden Sie diese.

e) Warum funktioniert die Displayausgabe nicht richtig?

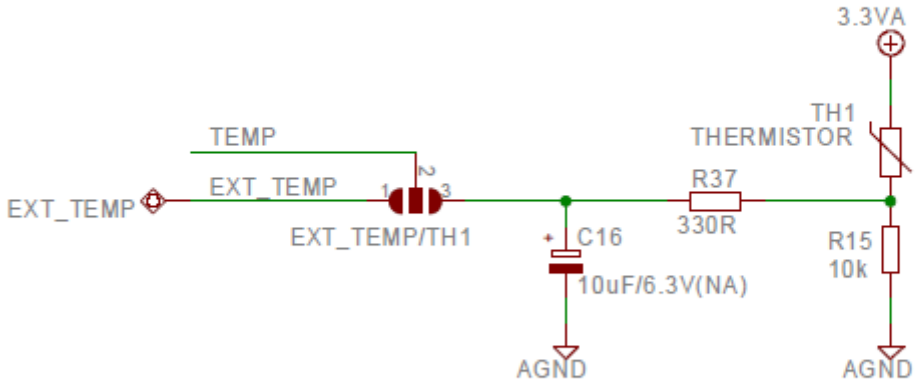
f) Betrachten Sie die Funktionsweise der Funktion itoa() und verwenden Sie diese für eine korrekte Displayausgabe .

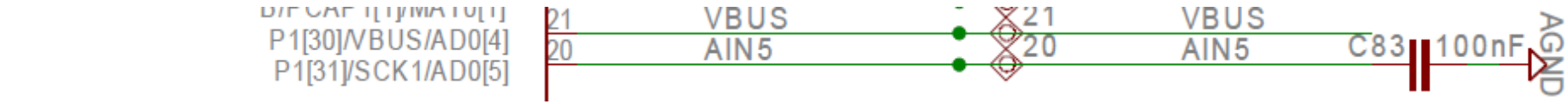
Lösung: Projekt 7b_ADC_SOLUTION/ADC2.eww

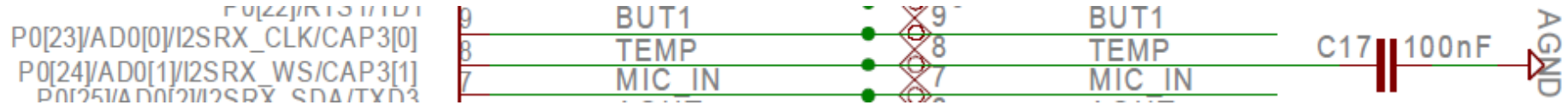
POTENTIOMETER



TEMPERATURE SENSOR







Verbesserung der Displayausgabe

g) Bislang erfolgt die Displayausgabe des ADC-Wertes linksbündig. Ändern Sie die Displayausgabe auf rechtsbündig.

h) Ermitteln Sie aus dem Datenblatt des ARM-Boards Auflösung und Farbtiefe des Displays. Berechnen Sie die Dateigröße eines Vollbildes.
Hinweis: Es können nur Bytes abgespeichert werden.

i) Im Quellcode der Mainroutine finden Sie folgende Zeile:

```
GLCD_PowerUpInit(0);
```

Ändern Sie diese Zeile wie folgt:

```
GLCD_PowerUpInit((pInt8U) Picture_HS.pPicStream);
```

Was ist das Ergebnis?

Temperaturmessung (kein Prüfungsstoff)

a) Wo befindet sich der Temperatursensor auf dem ARM-Board?
   Hinweis: Schaltplan

b) An welchem ADC-Pin ist der Temperatursensor angeschlossen?

c) Erweitern Sie nun den bestehenden Programmcode so, dass beim Drücken des linken Tasters der Potentiometerwert und beim Drücken des rechten Tasters der Temperaturwert auf dem Display angezeigt wird.

   Hinweis: Die Initialisierungsroutine des ADC beinhaltet bereits den Temperatursensor, somit sind hier keine Änderungen erforderlich.

d) Rechnen Sie den Temperaturwert in Grad Celsius um und geben Sie diese Temperatur aus. Nehmen Sie für die Berechnung folgende fiktive Werte an:

   - `Zimmertemperatur 20 ºC`
   - `1 Bit ~ 0.25 ºC`