

# Installationsanleitung

## OpenOCD

1. Stellen Sie sicher, dass die Pumpe und das Board angeschlossen und angeschaltet sind.
2. Stellen Sie sicher, dass das Board resettet wird.
3. Öffnen Sie die zur Verfügung gestellten Projektordner namens `filling_the_cup`
4. Öffnen Sie im Root Verzeichnis des Projektordners ein Terminal
5. Öffnen Sie ein Terminal im Verzeichnis des Projektordners und führen Sie folgenden Befehl aus:

```
code .
```

6. Nun sollte sich Visual Studio Code öffnen im Projektverzeichnis öffnen. Öffnen Sie in VS Code ein Terminal.
7. Führen Sie in dem VS Code Terminal folgenden Befehl aus

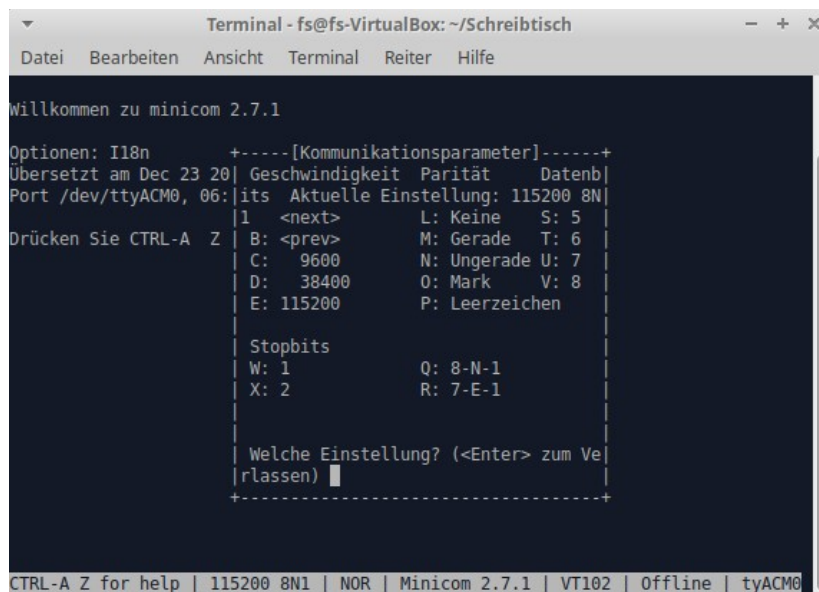
```
make openocd
```

## Minicom

1. Öffnen Sie ein neues Terminal
2. Führen Sie folgenden Befehl aus:

```
minicom
```

3. Im geöffneten minicom Terminal müssen nun folgende Einstellungen angepasst werden:



The screenshot shows a terminal window titled "Terminal - fs@fs-VirtualBox: ~/Schreibtisch". The minicom interface displays the following text:

```
Willkommen zu minicom 2.7.1
Optionen: I18n          +-----[Kommunikationsparameter]-----+
Übersetzt am Dec 23 20| Geschwindigkeit Parität  Datenb|
Port /dev/ttyACM0, 06|its Aktuelle Einstellung: 115200 8N|
Drücken Sie CTRL-A Z |l <next>          L: Keine   S: 5   |
| B: <prev>        M: Gerade  T: 6   |
| C: 9600          N: Ungerade U: 7   |
| D: 38400         O: Mark    V: 8   |
| E: 115200        P: Leerzeichen|
| Stopbits        |
| W: 1            Q: 8-N-1  |
| X: 2            R: 7-E-1  |
| Welche Einstellung? (<Enter> zum Ve|
| rlassen) |
+-----+
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tyACM0
```

4. Drücken Sie STRG + A und danach P

5. Drücken Sie die folgende Zeichen auf Ihrer Tastatur, während der Focus auf das minicom Terminal ist
  - a. W C N
6. Drücken Sie "ENTER"
7. Nun sollte minicom die passenden Einstellungen haben. Überprüfen Sie STRG + A und überprüfen Sie die Einstellungen in der Fußzeile des Terminals. Dort sollte mittlerweile das stehen: 9600 8O1
8. Stellen Sie sicher, dass die richtige Serielle Schnittstelle aktiviert ist. Das tun Sie mit STRG + A und danach O

The screenshot shows a terminal window titled "Terminal - fs@fs-VirtualBox: ~/Schreibtisch". The terminal output is as follows:

```
Willkommen zu minicom 2.7.1
Optionen: I18n
Übersetzt am Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 06:54:54
Drücken Sie CTRL-A Z für Hilfe zu speziellen Tasten

+-----[Konfiguration]-----+
| Dateinamen und Pfade          |
| Protokolle zur Dateiübertragung |
| Einstellungen zum seriellen Anschluss |
| Modem und Wählverhalten       |
| Bildschirm und Tastatur       |
| Speichern als »dfl«           |
| Einstellungen speichern als ... |
| Verlassen                     |
+-----+

CTRL-A Z for help | 9600 801 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyACM0
```

9. Wählen Sie "Einstellungen zum seriellen Anschluss aus

The screenshot shows the same terminal window after pressing 'A' (Ctrl-A). The output is:

```
Willkommen zu minicom 2.7.1
Opti+-----+
Über| A - Serieller Anschluss      : /dev/ttyACM0
Port| B - Pfad zur Lockdatei     : /var/lock
    | C - Programm zur Rufannahme :
Drüc| D - Programm zum Wählen      :
    | E - Bps/Par/Bits            : 9600 801
    | F - Hardware Flow Control    : Nein
    | G - Software Flow Control    : Nein
    |
    | Welchen Parameter möchten Sie ändern? |
+-----+
    | Bildschirm und Tastatur      |
    | Speichern als »dfl«         |
    | Einstellungen speichern als ... |
    | Verlassen                   |
+-----+

CTRL-A Z for help | 9600 801 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyACM0
```

10. Dort angekommen drücken Sie A auf der Tastatur und ersetzen die Zeichenkette

- beginnend mit tty.... mit folgender Zeichenkette: ttyUSB0
11. Drücken Sie ENTER
  12. Drücken Sie ESC

Nun sollte die Serielle Schnittstelle bereit sein. Lassen Sie das Minicom Fenster weiterhin geöffnet. Die Installation ist damit abgeschlossen

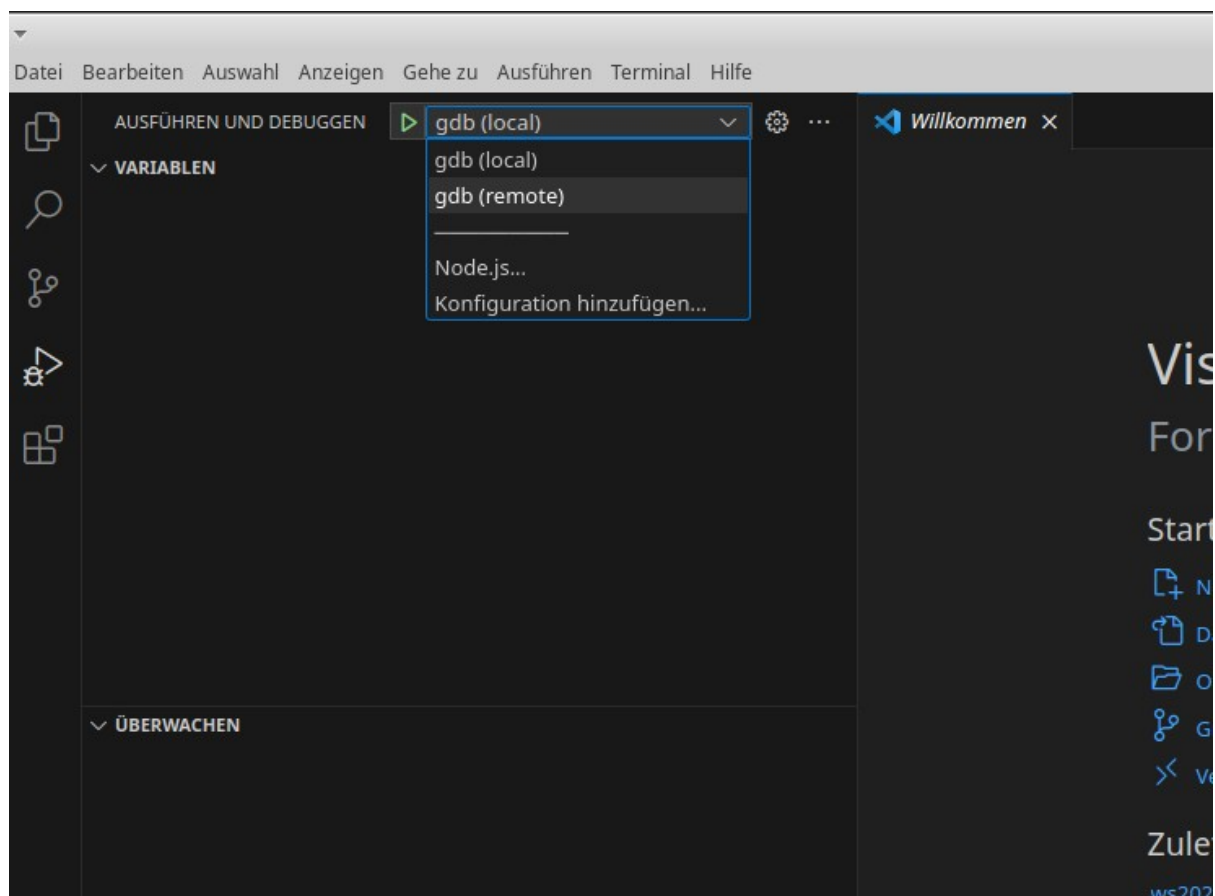
## Benutzerhandbuch

Da wir nun OpenOCD und das Minicom vorbereitet haben, können wir nun das Programm starten.

1. Das geöffnete VS Code Fenster im Rootverzeichnis beinhaltet ein Make File. Mit diesem können wir nun die von VS Code zur Verfügung gestellte RUN Schaltfläche nutzen. Dazu navigieren Sie bitte in der linken Navigationsleiste auf folgendes Symbol:



2. Jetzt sollte folgende Ansicht zu sehen sein:



5. Das nächste Dropdown beinhaltet nun alle Verfügbaren Arbeitsplätze. Dort wählen Sie bitte "Arbeitsplatz 1" aus.
6. Sobald das Programm gestartet wird, werden Sie nun von der Anwendung instruiert

### Programmausführung

1. Sie werden nun zunächst gebeten, den Becher zu kalibrieren
2. Ihnen wird das Gewicht des Bechers angezeigt
3. Über die USART Schnittstelle (minicom) können Sie nun mit der Taste P anfangen, das Wasser in den Becher zu füllen.

Falls der Becher vorzeitig von der Waage genommen wird, sollte hier die Wasserpumpe automatisch stoppen.

Falls das Wunschgewicht erreicht wurde, sollte man die Instruktion erhalten, den Becher wieder aufzuheben.

ACHTUNG: Der Becher muss vor einer neuen Iteration wieder geleert werden!

# Funktions- und Programmbeschreibung

## LED & Inputs PIO (Termin 2)

### Aufgabe 1 LED einschalten:

```
int main() {  
  
    PIOB<>PIO_OER = P27;  
    // Initial ist alles auf Eingang geschaltet, mit dem Befehl schaltet man es dann auf  
    // Ausgang  
    while (1)  
    {  
        PIOB<>PIO_CODR = P27;  
        // Ausschalten // Frage1) Mit welchen Anweisungen wird die gelbe LED „L“  
        // ein- bzw. ausgeschaltet und warum?  
        PIOB<>PIO_SODR = P27; // Einschalten  
    }  
}
```

Frage 1: Mit welchen Anweisungen wird die gelbe LED „L“ ein- bzw. ausgeschaltet und

warum?

PIOB->PIO\_CODR will turn the LED off, because CODR stands for Clear Output Data Register, and the LED isn't LOW Active  
PIOB->PIO\_SODR will turn the LED on

Frage 2: Auf welchen Speicherstellen und/oder in welchen Registern können Sie erkennen ob die gelbe LED "L" ein oder ausgeschaltet ist=  
Man sieht den Status der LED im PDSR Register

## Aufgabe 2 Inputs

Frage) In welchen Registern müssen welche Bit gesetzt sein/werden, damit der Zustand der Taste über das Pin Data Status Register (PDSR) erfasst werden kann?

Diese Register müssen gesetzt werden:

```
PMC->PMC_PCER0 = 0x01 << ID_PIOA | 1 << ID_PIOB;  
PIOB->PIO_OER = P27;
```

Auf welchen Speicherstellen kann man ablesen, ob die Taste gedrückt ist? (Hier Taste 1)

```
((PIOA->PIO_PDSR & 0x1) != 0x1)
```

## Aufgabe 3 Programm zum blinken der LED mit 0.5Hz

Taktfrequenz des Prozessor = 84MHz

```
include <sam.h>
#define P27 1u << 27 //0.5 Hz is 2 sec (fast)
int main()
{
    PIOB<>PIO_OER = P27; // Initial ist alles auf Eingang geschaltet,
    mit dem Befehl schaltet man es dann auf Ausgang
    int waitingtime = 5000000;
    /*
    Mit welcher Taktfrequenz läuft der Prozessor? 84 MHz Untersuchen
    Sie die Funktion ihres Programms auch mit
    verschiedenen Optimierungsstufen. */

    while (1) { for (int i = 0; i < waitingtime; i++){
        PIOB<>PIO_CODR = P27; // Ausschalten
        for (int i = 0; i < waitingtime; i++){
            PIOB<>PIO_SODR = P27; // Einschalten
        }
    }
}
```

#### Aufgabe 4

Kombinieren Sie die Programme aus Aufgabe 2 und 3. Schalten Sie bei jedem Tastendruck die LED „L“ aus.  
Was erwarten Sie, wie der Tastendruck reagiert?

```
#include <sam.h>
#define P27 1u << 27

int main()
{
    PMC->PMC_PCER0 = 0x01 << ID_PIOD | 1 << ID_PIOB; //Activating
the PMC, in order to work with inputs

    PIOB->PIO_OER = P27;
    int waitingtime = 8000000;
    while (1)
    {

        /**
         * Termin 2 Aufgabe 4: If button pressed, then blinking
         *                               If button !pressed, then led is turned
on constantly
         */
        /*
         Button is low active so we check for being not 1
         Led is high active, so SODR activates the led, CODR
ausschaltet
         */
        while ( (PIOD->PIO_PDSR & 0x1) != 0x1) {
            for (int i = 0; i < waitingtime; i++){};
            PIOB->PIO_CODR = P27; // Ausschalten

            for (int i = 0; i < waitingtime; i++){};
            PIOB->PIO_SODR = P27; // einschalten
        }
        //Led eingeschaltet (low active)
        PIOB->PIO_SODR = P27; //Led eingeschaltet
    }

    return 0;
}
```

Antwort:

While the button is pressed, it goes to the routine with the loops and turn the LED on and off (blinking)

While the button is NOT pressed, the LED stays on because the program don't reach the routine of the "blinking"

### Antwort

Solange der Button gedrückt ist, können die Schleifen durchgeführt werden und die LEDs blinken.

Ist die Taste nicht gedrückt, können die Blink-Schleifen nicht mehr erreicht werden und die LED behält den Zustand

## TIMER WAVE MODE (Termin 3)

### Aufgabe 1:

Es soll eine Kolbenhubpumpe, welche über PB27/TIOB0 angesteuert wird, betrieben werden. Die Pumpe benötigt ein symmetrisches Rechtecksignal mit einer Frequenz von ca. 50Hz. Sie könnten eine Zeitschleife programmieren. Hiermit würden Sie aber den Prozessor blockieren (Erinnerung an Termin 2). Besser ist es, Sie initialisieren einen Timer (Timer0) so, dass dieser selbstständig das Signal für die Pumpe an TIOB0 erzeugt.

**ACHTUNG: Die Pumpe darf kein Dauerhighsignal erhalten.**

Vervollständigen Sie die das gegebene Programm *Termin3Aufgabe1.c* entsprechend. Ergänzen und berichtigen Sie auch die Kommentare.

```
void generate_pump_signal() {
    /**
     * Termin 3 Aufgabe 1
     * Generate a signal at the frequency of 50Hz
     */
    TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKDIS; // Counter Clock
    Disable Command

    TC0->TC_CHANNEL[0].TC_CMR = TC_CMR_WAVE |
    // Waveform mode is enabled
                                TC_CMR_WAVSEL_UP_RC |
    // UP mode with automatic trigger on RC Compare
                                TC_CMR_TCCLKS_TIMER_CLOCK1 |
    // MCK/2
                                TC_CMR_EEVT_XC0 |
    // Use xc0 as external event instead of TIOB so that TIOB can also
    // be used as an output.
                                TC_CMR_BCPB_SET |
    TC_CMR_BCPC_CLEAR; // Toggle TIOB output on RC Compare

    TC0->TC_CHANNEL[0].TC_RC = SystemCoreClock / 2 / 50; // 50Hz,
```



```

so we divide by the frequency we want
TC0->TC_CHANNEL[0].TC_RB = TC0->TC_CHANNEL[0].TC_RC >> 1;

TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKEN | TC_CCR_SWTRG; //
Counter Clock Enable and Software Trigger Command
PIOB->PIO_PDR = 1u << 27; // three columns, I want to work
with column B, not with Parallel IO
PIOB->PIO_ABSR = 1u << 27; // select peripheral b
}

```

Berechnung

Prescaler =  $f_{CLK} / (f_A \cdot RC)$

$84000000 / (50 \cdot 2^{32}) = 0,0003911554813385009765625$

## Aufgabe 2:

Erweitern Sie Ihr Programm so, dass die Pumpe durch Betätigung von Tasten eingeschaltet und abgeschaltet werden kann.

Welche Möglichkeiten haben Sie gefunden, um das Pumpensignal ein- bzw. auszuschalten?

..

Für welche Lösung entscheiden Sie sich und warum?

..

```

int main() {
    PMC->PMC_PCER0 = 1u << ID_TC0; // I let TCLK0 pass durch
multiplexer

    PIOB->PIO_PER = 1u << 27; // pin controlled by pio
    PIOB->PIO_OER = 1u << 27; // pin as output
    PIOB->PIO_CODR = 1u << 27; // pin low

    while(1) {
        if((PIOD->PIO_PDSR & 0x1) != 0x1) {
            TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKDIS; // Counter Clock
Disable Command

            TC0->TC_CHANNEL[0].TC_CMR = TC_CMR_WAVE | // Waveform mode
is enabled
                                TC_CMR_WAVSEL_UP_RC | // UP
mode with automatic trigger on RC Compare
                                TC_CMR_TCCLKS_TIMER_CLOCK1 | //
MCK/2
                                TC_CMR_EEVT_XC0 |
// Use xc0 as external event instead of TIOB so that TIOB can also

```

be used as an output.

```
TC_CMR_BCPB_SET |
TC_CMR_BCPC_CLEAR; // Toggle TIAB output on RC Compare

TC0->TC_CHANNEL[0].TC_RC = SystemCoreClock / 2 /
desired_frequency; // 50Hz, so we divide by the frequency we want
TC0->TC_CHANNEL[0].TC_RB = TC0-
>TC_CHANNEL[0].TC_RC >> 1;

TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_CLKEN |
TC_CCR_SWTRG; // Counter Clock Enable and Software Trigger Command
PIOB->PIO_PDR = 1u << 27; // three columns, I want
to work with column B, not with Parallel IO
PIOB->PIO_ABSR = 1u << 27; // select peripheral b
} else if((PIOB->PIO_PDSR & 0x1) != 0x2) {
PIOB->PIO_PER = 1; // three columns, I want to
work with column B, not with Parallel IO
}

}
// }

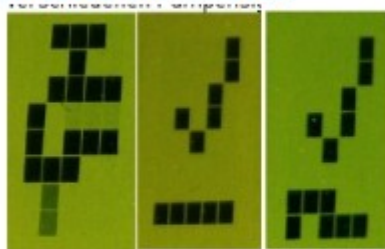
return 0;
}
```

Wir können die Pumpe mit `PIOB->PIO_PER = 1` ausschalten, da damit Kontrolle auf die PIO gelegt wird

### Aufgabe 3:

Beschäftigen Sie sich mit der Initialisierung des Timer und versuchen Sie jedes Symbol der verschiedenen möglichen Pumpensignale darzustellen.

The following will make a clear connection between the required pictures and what we modified in our code, so that we could achieved the results.



*es pumpt  
Gewicht  
nimmt zu*

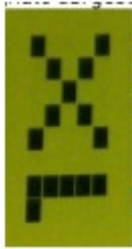
*kein Signal*

*Frequenz  
richtig*

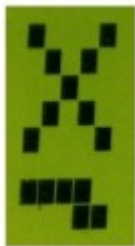
```
- TC0->TC_CHANNEL[0].TC_RC =
  SystemCoreClock / 2 / 50;

TC0->TC_CHANNEL[0].TC_RB =
  TC0->TC_CHANNEL[0].TC_RC / 2;
Those are normal functioning
conditions. Kein Signal refers to the
situation when the "sink" is off.
```

This one was the trickiest to achieve. The secret is that Register B should be null. In terms of hardware, there is no period allowed of low signal, not even the smallest possible. It has to keep a high signal.



You have to remove TC\_CMR\_BCPC\_CLEAR from TC\_CMR. It doesn't allow the signal to be low (never clearing).



When RC (Register C) is too low -> then the Frequency is too high  
If you divide RC by something greater than two and assigned it to Rb -  
> then the High Period is smaller



If you divide Rc by something smaller than two and assigned it to Rb -  
> then Low period is bigger

## TIMER CAPTURE MODE (Termin 4)

### Aufgabe 1:

- Which values (count values, time) for the period duration to be determined at 500Hz do you expect?
  - Count value = 83986 (for both Timer Components at 500Hz frequency)
  - time = 200ms (1/500Hz)
- Which frequencies can we record with the selected initialisation?
  - Basically this question is left in the protocol with respect to old ARM processor with the sole purpose of determining how high can the frequency go until it overflows. So with Cortex M3 there is no need to answer that.
- Does the period duration change when the balance is loaded?

- Changing the mass (putting on the scale an object with a weight) will automatically change the frequencies (freq\_Pin25 will decrease, freq\_Pin28 will increase, and the distance inbetween is equal)
- zum Beispiel: 250g, freq\_Pin28 = 471.70Hz, freq\_Pin25 = 530.33

## Aufgabe 2:

- Document which frequency decreases with increasing weight and which frequency increases with increasing weight.
  - Weight increases => Freq of PA7 (Pin 28) decreases, Freq of PA4 (Pin 25) increases

## USART (Termin5)

Programmierung für eingebettete Systeme: Pointer, Peripherie,

USART, SWI

**Aufgabe 1) Wie kann die USART-Schnittstelle auf dem Arduino Due initialisiert werden?**

Antwort:

- Zunächst muss die "Peripheral Clock" eingeschaltet werden.
  - PMC\_PCER

*PMC->PMC\_PCER0 = 1u << ID\_USART1;*

- Transmitter und Receiver müssen zurückgesetzt werden
  - US\_CR

*USART1->US\_CR = US\_CR\_RXDIS | US\_CR\_TXDIS | US\_CR\_RSTRX | US\_CR\_RSTTX;*

- Baudrate muss eingestellt werden

- US\_BRGR

*USART1->US\_BRGR = SystemCoreClock / 16 / 9600;*

- Mode muss eingestellt werden

- US\_MR

*USART1->US\_MR = US\_MR\_USART\_MODE\_NORMAL | // Normal mode*

*US\_MR\_USCLKS\_MCK | // Master Clock is selected*

*US\_MR\_CHRL\_8\_BIT | // Character length is 8 bits*

*US\_MR\_PAR\_ODD | // Odd parity*

*US\_MR\_NBSTOP\_2\_BIT | // 2 stop bit*

*US\_MR\_CHMODE\_NORMAL; // Normal channel mode*

- PIO Ports müssen eingeschaltet werden

- PIO\_PDR

*PIOA->PIO\_PDR = PIO\_PA13A\_TXD1 | PIO\_PA12A\_RXD1;*

- Transmitter und Receiver aktivieren

- US\_CR

*USART1->US\_CR = US\_CR\_RXEN | US\_CR\_TXEN;*

**Welche Pins werden für die Kommunikation verwendet, und welcher Controller ist damit verbunden?**

Folgende 2 Pins werden genutzt:

Welcher Controller? USART1

PIOA->PIO\_PDR = PIO\_PA13A\_TXD1 | PIO\_PA12A\_RXD1 -> Pin 16 und 17

**Warum ist es wichtig, die Kommunikationsschnittstelle korrekt einzurichten?**

Es müssen die richtigen Ports freigegeben (PIO) und die USART richtig eingestellt werden, damit gewährleistet ist, dass die Werte gesendet und empfangen werden können.

## **Aufgabe 2)**

**Welches Register wird verwendet, um ein Zeichen über die USART-Schnittstelle zu übertragen?**

Antwort: Um das Zeichen dann zu senden muss das Zeichen in das US\_THR Register gesetzt werden. Dabei ist zu beachten, dass das Register TXRDY auf 1 gesetzt ist und demnach das THR leer ist. Ist TXRDY auf 0 gehen die character verloren

**Wie lässt sich überprüfen, ob die Übertragung erfolgreich war?**

Antwort: US\_CSR\_RXRDY ist das Register, in dem man nachsehen kann, ob das character angekommen ist. Dabei landet es in US\_RHR. Man muss allerdings überprüfen, ob OVRE Bit gesetzt ist, weil es sein kann, dass das RXRDY Bit schon vorher gesetzt war.

Alle Register überwacht man im US\_CSR Register.

USART1->US\_CSR & US\_CSR\_RXRDY

**Warum ist das Channel Status Register (US\_CSR) relevant?**

Antwort: Die ganzen Register die im Rahmen von Transmitting and Receiving genutzt werden, müssen über das CSR Register abgelesen werden (US\_CSR\_TXRDY, US\_CSR\_RXRDY)

### Aufgabe 3)

Welche Register werden genutzt, um ein empfangenes Zeichen zu speichern und zu überprüfen?

Antwort: Zum Einen über das RXRDY Register, um zu prüfen, ob ein neues Zeichen angekommen ist und entsprechend das US\_RHR mit dem Inhalt der Übertragung.

Welche Rolle spielt das Channel Status Register (US\_CSR) beim Empfangsprozess?

Antwort: Über das US\_CSR Register lässt sich ermitteln, ob ein neues Zeichen empfangen wurde. Hier muss man dann prüfen, ob das US\_RXRDY Register gesetzt ist im US\_CSR

```
#include <sam.h>

#define PA12 1u << 12
#define PA13 1u << 13

int main(void)
{
    //PIOA->PIO_PDR = PA12 | PA13;
    //PIOA->PIO_ABSR &= ~(PA12 | PA13);

    PMC->PMC_PCER0 = 1u << ID_USART1;

    USART1->US_CR = US_CR_RXDIS | US_CR_TXDIS |
    US_CR_RSTRX | US_CR_RSTTX; // Receiver and transmitter
    resetted and disabled
    USART1->US_PTCR = US_PTCR_RXTDIS | US_PTCR_TXTDIS;

    USART1->US_MR = US_MR_NBSTOP_2_BIT |
                    US_MR_PAR_ODD |
                    US_MR_CHRL_8_BIT |
                    US_MR_CHMODE_NORMAL |
                    US_MR_USCLKS_MCK;
    USART1->US_BRGR = SystemCoreClock / 16 / 9600;

    USART1->US_CR = US_CR_RXEN | US_CR_TXEN; // Receiver
    and transmitter enabled

    USART1->US_TCR = 0;
```

```

PIOA->PIO_PDR = PIO_PA13A_TXD1 | PIO_PA12A_RXD1;

// init segment
//
#####
#####
//put 1 char

char c = '\n A';
char receivedC = ';';
int x = 0;

while(~USART1->US_CSR & US_CSR_TXRDY);
USART1->US_THR = c;


// int check_mask = 0xffffffff;
// check_mask = check_mask & ~USART1->US_CSR;

while(~USART1->US_CSR & US_CSR_RXRDY);
/// initially CSR is empty (000000) while we have
nothing to read.
/// The whole condition of while is 0.
/// Inverted is 1 and we don't leave the loop.
/// when we have sth. to read,
/// RXRDY turns to 1 => whole expr turns to 1 (sth.
not 0);
/// inverted is 0 and we leave the loop.

receivedC = USART1->US_RHR;
return 0;
}

```