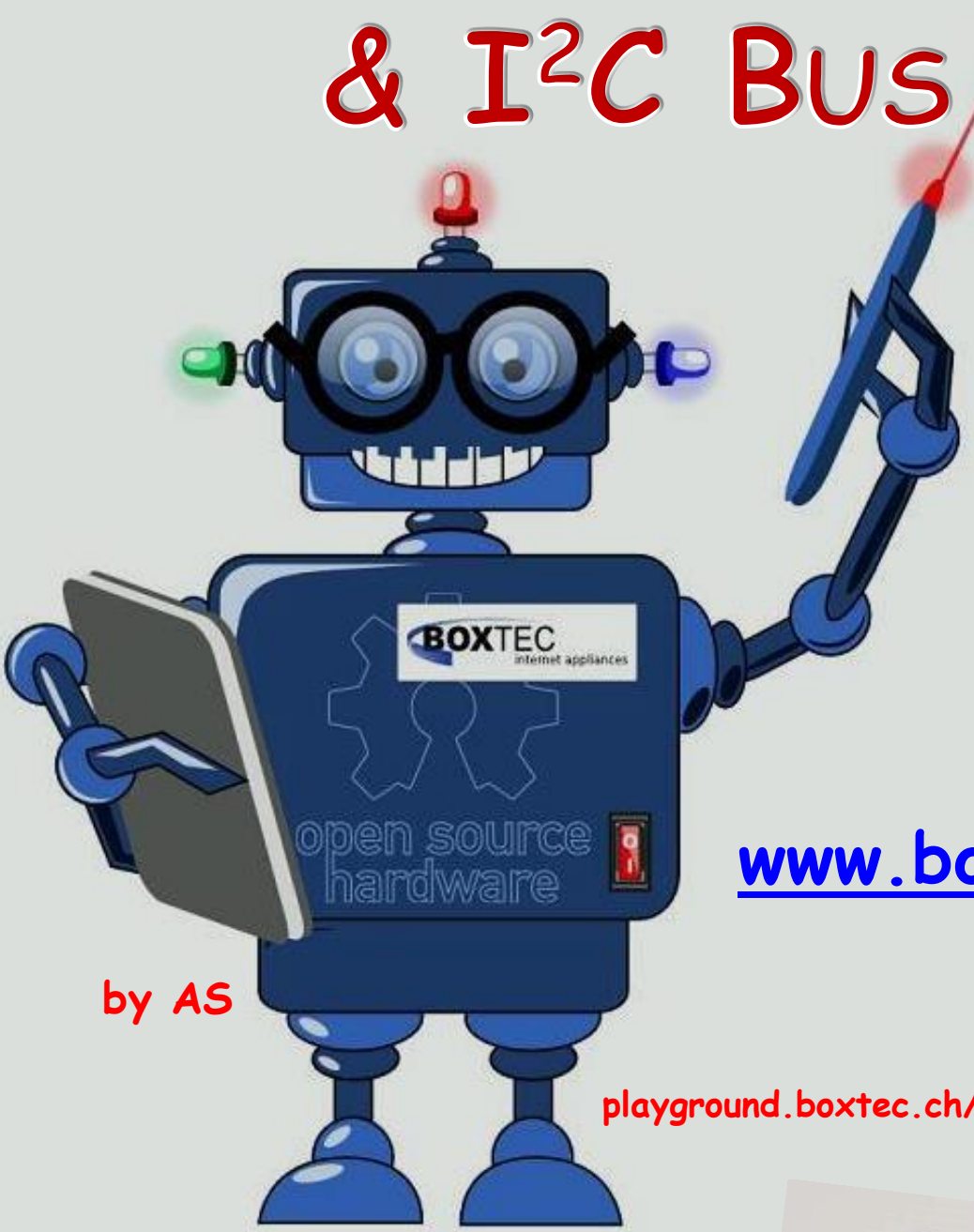
Mikrokontroller & I²C Bus



www.boxtec.ch

by AS

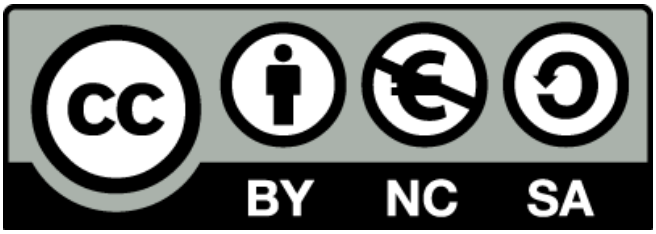
playground.boxtec.ch/doku.php/tutorial

ARM Controller - SAM D21 J17A
(32 Bit Controller)
Der I²C Bus 1

SAM D21 J17A

Copyright

Sofern nicht anders angegeben, stehen die Inhalte dieser Dokumentation unter einer „Creative Commons - Namensnennung-NichtKommerziell-Weitergabe unter gleichen Bedingungen 3.0 DE Lizenz“



Sicherheitshinweise

Lesen Sie diese Gebrauchsanleitung, bevor Sie diesen Bausatz in Betrieb nehmen und bewahren Sie diese an einem für alle Benutzer jederzeit zugänglichen Platz auf. Bei Schäden, die durch Nichtbe-achtung dieser Bedienungsanleitung verursacht werden, erlischt die Gewährleistung / Garantie. Für Folgeschäden übernehmen wir keine Haftung! Bei allen Geräten, die zu ihrem Betrieb eine elektrische Spannung benötigen, müssen die gültigen VDE-Vorschriften beachtet werden. Besonders relevant sind für diesen Bausatz die VDE-Richtlinien VDE 0100, VDE 0550/0551, VDE 0700, VDE 0711 und VDE 0860. Bitte beachten Sie auch nachfolgende Sicherheitshinweise:

- Nehmen Sie diesen Bausatz nur dann in Betrieb, wenn er zuvor berührungssicher in ein Gehäuse eingebaut wurde. Erst danach darf dieser an eine Spannungsversorgung angeschlossen werden.
- Lassen Sie Geräte, die mit einer Versorgungsspannung größer als 24 V- betrieben werden, nur durch eine fachkundige Person anschließen.
- In Schulen, Ausbildungseinrichtungen, Hobby- und Selbsthilfewerkstätten ist das Betreiben dieser Baugruppe durch geschultes Personal verantwortlich zu überwachen.
- In einer Umgebung in der brennbare Gase, Dämpfe oder Stäube vorhanden sind oder vorhanden sein können, darf diese Baugruppe nicht betrieben werden.
- Im Falle eine Reparatur dieser Baugruppe, dürfen nur Original-Ersatzteile verwendet werden! Die Verwendung abweichender Ersatzteile kann zu ernsthaften Sach- und Personenschäden führen. Eine Reparatur des Gerätes darf nur von fachkundigen Personen durchgeführt werden.
- Spannungsführende Teile an dieser Baugruppe dürfen nur dann berührt werden (gilt auch für Werkzeuge, Messinstrumente o.ä.), wenn sichergestellt ist, dass die Baugruppe von der Versorgungsspannung getrennt wurde und elektrische Ladungen, die in den in der Baugruppe befindlichen Bauteilen gespeichert sind, vorher entladen wurden.
- Sind Messungen bei geöffnetem Gehäuse unumgänglich, muss ein Trenntrafo zur Spannungsversorgung verwendet werden
- Spannungsführende Kabel oder Leitungen, mit denen die Baugruppe verbunden ist, müssen immer auf Isolationsfehler oder Bruchstellen kontrolliert werden. Bei einem Fehler muss das Gerät unverzüglich ausser Betrieb genommen werden, bis die defekte Leitung ausgewechselt worden ist.
- Es ist auf die genaue Einhaltung der genannten Kenndaten der Baugruppe und der in der Baugruppe verwendeten Bauteile zu achten. Gehen diese aus der beiliegenden Beschreibung nicht hervor, so ist eine fachkundige Person hinzuzuziehen

Bestimmungsgemäße Verwendung

- Auf keinen Fall darf 230 V~ Netzspannung angeschlossen werden. Es besteht dann Lebensgefahr!
- Dieser Bausatz ist nur zum Einsatz unter Lern- und Laborbedingungen konzipiert worden. Er ist nicht geeignet, reale Steuerungsaufgaben jeglicher Art zu übernehmen. Ein anderer Einsatz als angegeben ist nicht zulässig!
- Der Bausatz ist nur für den Gebrauch in trockenen und sauberen Räumen bestimmt.
- Wird dieser Bausatz nicht bestimmungsgemäß eingesetzt kann er beschädigt werden, was mit Gefahren, wie z.B. Kurzschluss, Brand, elektrischer Schlag etc. verbunden ist. Der Bausatz darf nicht geändert bzw. umgebaut werden!
- Für alle Personen- und Sachschäden, die aus nicht bestimmungsgemäßer Verwendung entstehen, ist nicht der Hersteller, sondern der Betreiber verantwortlich. Bitte beachten Sie, dass Bedien- und /oder Anschlussfehler außerhalb unseres Einflussbereiches liegen. Verständlicherweise können wir für Schäden, die daraus entstehen, keinerlei Haftung übernehmen.
- Der Autor dieses Tutorials übernimmt keine Haftung für Schäden. Die Nutzung der Hard- und Software erfolgt auf eigenes Risiko.

SAM D21 J17A – Der I²C Bus

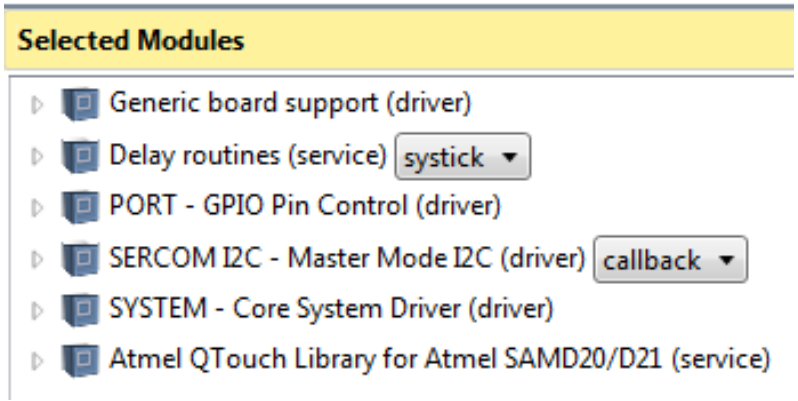
Im nächsten Teil möchte ich den I²C Bus am SAM D21 und ein erstes Programm dazu vorstellen und kurz erläutern. Als Hardware verwende ich wieder den NanoSAMD21 auf der Grundplatine P110 zusammen mit dem Out Modul P36 und dem PCF8574.

Out Modul P36 mit PCF8574

Nano SAMD21 Microcontroller Modul auf der Grundplatine 110

Die Verbindung der Platinen erfolgt durch einen 10 poligen Buchsen Stecker und dem Netzteil NT2 mit 5V und 12V bei jeweils 1,5A. Die Beschreibung zu den entsprechenden Modulen befindet sich im Netz auf der angebenden Seite. Beim Modul P110 mit dem Nano SAM D21 wurde der Pegelwandler bestückt.

Diese Dateien müssen vorher installiert werden



Die Beschreibung bitte den zur Installation bitte den Tutorials entnehmen.

Wir wollen als erstes eine LED auf dem P36 mit einem Taster vom P110 ein- und ausschalten.

Es werden die folgenden Pins verwendet:

Auf dem P110: **TA 4** – schaltet die LED um
LED 1 – Anzeige der Umschaltung

Auf dem P36: **L1** und **L4**, **L5** und **L7**, Angabe der LED kann geändert werden

Als nächstes werde ich einige Teile des Programmes beschreiben. Der komplette Code steht am Ende dieses Tutorials.

Im Programm sieht die Angabe der Pins und Adressen so aus:

```
#define SLAVE_ADDRESS 0x21          // Angabe Slave Adresse 0x21
#define LED1  PIN_PB05              // PB05 -  LED auf P110
#define Button1   PIN_PB23          // PB23 - Taster auf P110
```

Das sind die einzelnen Ports/Pins die benutzt werden:

LED1 – Led4 – P1/17 – PB05 (mit Angabe der Verbindungspunkte)
Button1 – Taster4 – P1/2 – PB23

Zusätzlich gebe ich hier noch die Adresse **(0x21)** des Slaves im I²C Bus an.

```
void configure_i2c(void)                          // configuration SERCOM 2 als I2C-Master
 {
   struct i2c_master_config config_i2c_master;    // generate the configure-struct
   i2c_master_get_config_defaults(&config_i2c_master);          // get the defaults
   config_i2c_master.pinmux_pad0 = PINMUX_PA08C_SERCOM0_PAD0;   // PA08 als SDA
   config_i2c_master.pinmux_pad1 = PINMUX_PA09C_SERCOM0_PAD1;   // PA09 als SCL
   config_i2c_master.baud_rate = 100;             // Setzt Baudrate auf 100kHz
   while(i2c_master_init(&i2c_master_instance, SERCOM0, &config_i2c_master) != STATUS_OK);
                                                  //initialiation
   i2c_master_enable(&i2c_master_instance);       // I2C Interface einschalten
 }
```

Mit diesem Teil konfiguriere ich den I²C Bus. Unter anderem wird hier

- Sercom 0 als Master konfiguriert
- Die Pins für SDA und SCL angegeben (PA08 und PA09)
- Die Baudrate eingestellt (100kHz)
- Das I2C Interface eingeschaltet

```
void configure_port_pins(void)                    // konfiguration der Ports/Pins
 {              // Angabe LED
   struct port_config config_port_pin;
   port_get_config_defaults(&config_port_pin);
   config_port_pin.direction = PORT_PIN_DIR_OUTPUT;
   port_pin_set_config(LED1, &config_port_pin);   // Angabe LED1
                  // Angabe Taster
   port_get_config_defaults(&config_port_pin);
   config_port_pin.direction = PORT_PIN_DIR_INPUT;
   config_port_pin.input_pull = PORT_PIN_PULL_UP;
   port_pin_set_config(Button1, &config_port_pin);   // Angabe Button1 (Taster1)
 }
```

Damit konfiguriere ich die Ein- und Ausgänge die ich mit dem SAMD21 schalten möchte. Wenn ich die Belegung ändern möchte, muss ich die Angabe zum Anfang des Programmes ändern.

```
i2c_write_buffer_a[0] = 0xaf;          // Angabe LED für a, 1. Zahl für LED 5-8
i2c_write_buffer_b[0] = 0xf6;          // Angabe LED für b, 2. Zahl für LED 1-4
```

Damit erfolgt die Angabe der LED die ich auf dem Slave schalten möchte.
Bitte beachten, dass die 1.Zahl die LEDs 5-8 und die 2. Zahl die LEDs 1-4 angibt. Die genaue Belegung kann den bereits veröffentlichen Tabellen in anderen Teilen entnommen werden.

Kommen wir nun zum eigentlichen schreiben auf dem Slave. Als erstes habe ich alle notwendigen Befehle zusammengefasst.

```
void write_a_data(void);                    // Angabe Prototyp write_a_data
struct i2c_master_packet wr_a_packet;       // deklarieren einer Struktur mit Typbezeichner
void write_a_data(void)                     // Funktiondefinition void für write_a_data
 {
   uint16_t timeout;
   timeout = 100;
   wr_a_packet.address = SLAVE_ADDRESS;
   wr_a_packet.data_length = DATA_LENGTH;
   wr_a_packet.data = i2c_write_buffer_a;
   while (i2c_master_write_packet_wait(&i2c_master_instance, &wr_a_packet) != STATUS_OK)
    {
      if (timeout ++ == Timeout )
       {
        // Error LED_PB30_ON
        break;
       }
    }
 }
```

- Alle Daten werden in einer Struktur Namens **write_a_data** zusammengefasst. Strukturen dienen dazu, mehrere logisch zusammenhängende Variablen unterschiedlichen Datentyps zusammenzufassen
- Mit dem Typbezeichner **write_a_data** kann man diesen später als Synonym zum Aufruf verwenden.
- Innerhalb der Klammer werden die Elemente mit **wr_a_packet.address,** **wr_a_packet.data_length** und **wr_a_packet.data** der Struktur deklarieret. Ein Element ist nichts anderes als eine normale Variable, die als Teil einer Struktur definiert wird.

  wr_a_packet.address – Angabe der Bus Adresse
  wr_a_packet.data_length – Angabe der Länge der Daten
  wr_a_packet.data – Angabe der Daten

Diese Daten werden im **wr_a_packet** zusammengefasst und zum Slave übertragen mit

```
while (i2c_master_write_packet_wait(&i2c_master_instance, &wr_a_packet) != STATUS_OK)
```

gleichzeitig erfolgt eine Abfrage ob der Slave vorhanden ist und mit ACK antwortet.

```
if (timeout ++ == Timeout )
       {
        // Error LED_PB30_ON
        break;
       }
```

Die Einstellung der Anfragen erfolgt mit **timeout = 100**. Erfolgt innerhalb der eingestellten Abfragen keine Antwort Stoppt das Programm mit **break**. Die Anzeige mit einer LED habe ich auskommentiert (// Error LED_PB30_ON).

```
system_init();                                  // int für notwendige Datein
configure_i2c();
configure_port_pins();
delay_init();
```

Zum Start des Programmes müssen wir die notwendigen Dateien/Bibliotheken aufrufen.

```
while (1)
    {
     // Abfrage Taster
     //if (port_pin_get_input_level(Button1))     // Taster nicht gedrückt
     if (!port_pin_get_input_level(Button1))     // Taster gedrückt
      {                                          // Schaltet LED
        write_a_data();                          // Aufruf Bus schreiben a
        port_pin_toggle_output_level(LED1);      // Toggelt LED1
        delay_ms(500);                           // Zeit 500ms
        port_pin_toggle_output_level(LED1);      // Toggelt LED1
        delay_ms(500);                           // Zeit 500ms
      }
     else
      {
        write_b_data();                          // Aufruf Bus schreiben b
      }
    }
```

Innerhalb der **while Schleife** wird der Taster abgefragt. Bei Taster gedrückt erfolgt ein toggle der LED1 und schalten der LEDs über den I²C Bus mit **write_a_data()**. Wenn der Taster nicht betätigt wird, werden die LEDs mit write_b_data() über den I²C Bus geschaltet. Der Taster kann zwischen **gedrückt** und **nicht gedrückt** umgeschaltet werden.

Das komplette Programm:

```
/* ARM I2C Bus Prg 1 * Angepasst an meine Hardware mit P36 mit 8x LED Adresse 42 und P110 mit
nanoSAMD21 von Rodenhausen   by Achim Seeger und Dirk   */

#include <asf.h>                                 // Einbinden Bibliothek

#define CONF_I2C_MASTER_MODULE SERCOM1
#define SLAVE_ADDRESS 0x21                       // Angabe Slave Adresse 0x21
#define LED1  PIN_PB05                           // PB05 - LED auf P110
#define Button1   PIN_PB23                       // PB23 - Taster auf P110
#define DATA_LENGTH 1
#define Timeout 1000

void configure_i2c(void);                        // Angabe Prototype
void write_a_data(void);
void write_b_data(void);
void configure_port_pins(void);

struct i2c_master_module i2c_master_instance;    // generiere die struct für I2C-Master
struct i2c_master_packet wr_a_packet;
struct i2c_master_packet wr_b_packet;

static uint8_t i2c_write_buffer_a[DATA_LENGTH];  // Angabe der Länge für buffer
```

```
static uint8_t i2c_write_buffer_b[DATA_LENGTH];

void configure_i2c(void)                                    // configuration SERCOM 2 als I2C-Master
 {
   struct i2c_master_config config_i2c_master;              // generate the configure-struct
   i2c_master_get_config_defaults(&config_i2c_master);                  // get the defaults
   config_i2c_master.pinmux_pad0 = PINMUX_PA08C_SERCOM0_PAD0;   // PA08 als SDA
   config_i2c_master.pinmux_pad1 = PINMUX_PA09C_SERCOM0_PAD1;   // PA09 als SCL
   config_i2c_master.baud_rate = 100;                       // Setzt Baudrate auf 100kHz
   while(i2c_master_init(&i2c_master_instance, SERCOM0, &config_i2c_master) != STATUS_OK);
                                                            //initialization
   i2c_master_enable(&i2c_master_instance);                 // I2C interface einschalten
 }

void write_a_data(void)                             // Bus schreiben für a
 {
        uint16_t timeout;
        timeout = 100;
        wr_a_packet.address = SLAVE_ADDRESS;
        wr_a_packet.data_length = DATA_LENGTH;
        wr_a_packet.data = i2c_write_buffer_a;
        while (i2c_master_write_packet_wait(&i2c_master_instance, &wr_a_packet) != STATUS_OK)
         {
                if (timeout ++ == Timeout )
                 {
                        // Error LED_PB30_ON
                        break;
                 }
         }
 }

void write_b_data(void)                                     // Bus schreiben für b
 {
   uint16_t timeout;
   timeout = 100;
   wr_b_packet.address = SLAVE_ADDRESS;             // Angabe Adresse
   wr_b_packet.data_length = DATA_LENGTH;           // Angabe Länge
   wr_b_packet.data = i2c_write_buffer_b;           // Angabe Buffer (LEDs)
   while (i2c_master_write_packet_wait(&i2c_master_instance, &wr_b_packet) != STATUS_OK)
    {
      if (timeout ++ == Timeout )
       {
        // Error LED_PB30_ON
        break;
       }
    }
 }

void configure_port_pins(void)                                  // konfiguration der Ports/Pins
 {              // Angabe LED
   struct port_config config_port_pin;
   port_get_config_defaults(&config_port_pin);
   config_port_pin.direction = PORT_PIN_DIR_OUTPUT;
```

```
  port_pin_set_config(LED1, &config_port_pin);              // Angabe LED1
                  // Angabe Taster
   port_get_config_defaults(&config_port_pin);
   config_port_pin.direction = PORT_PIN_DIR_INPUT;
   config_port_pin.input_pull = PORT_PIN_PULL_UP;
   port_pin_set_config(Button1, &config_port_pin);          // Angabe Button1 (Taster1)
 }

int main(void)
 {
   system_init();                                           // int für notwendige Datein
   configure_i2c();
   configure_port_pins();
   delay_init();
   i2c_write_buffer_a[0] = 0xaf;                            // Angabe LED für a, 1.Zahl für LED 5-8
   i2c_write_buffer_b[0] = 0xf6;                            // Angabe LED für b, 2.Zahl für LED 1-4
   while (1)
    {
      // Abfrage Taster
      //if (port_pin_get_input_level(Button1))             // Taster nicht gedrückt
      if (!port_pin_get_input_level(Button1))             // Taster gedrückt
       {                                                  // Schaltet LED
         write_a_data();                                  // Aufruf Bus schreiben a
         port_pin_toggle_output_level(LED1);              // Toggelt LED1
         delay_ms(500);                                   // Zeit 500ms
         port_pin_toggle_output_level(LED1);              // Toggelt LED1
         delay_ms(500);                                   // Zeit 500ms
       }
      else
       {
         write_b_data();                                  // Aufruf Bus schreiben b
       }
    }
 }
```

Meine Programmbeispiele sollen ein Ansporn für andere sein eigene Programme zu schreiben und mit der Hardware NanoSAMD21 zu arbeiten.

Mein besonderer Dank an Dirk für seine Hilfe bei den Programmen.

Einige Teile des Textes wurden zur besseren Übersicht farblich gestaltet.
Die Nutzung erfolgt auf eigenes Risiko.
Ich wünsche viel Spaß beim Bauen und programmieren
Achim

myroboter@web.de

Quellenangabe

Elektor März und folgend 2015 - Von 8 auf 32 bit von Viacheslav Gromov
https://www.makerconnect.de Pogrammierung SAM D21 mit Atmega ICE und SWD