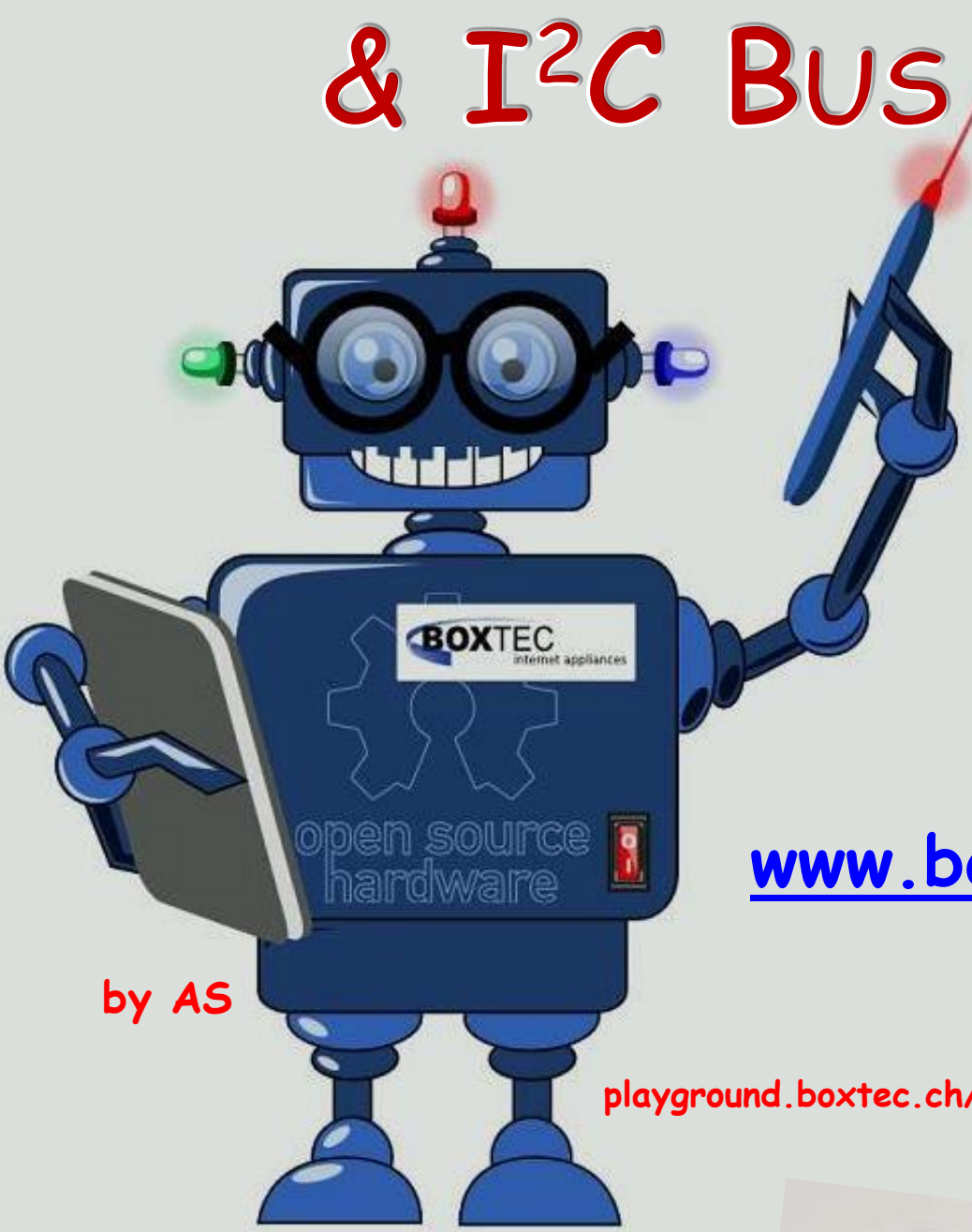
Mikrokontroller & I²C Bus



www.boxtec.ch

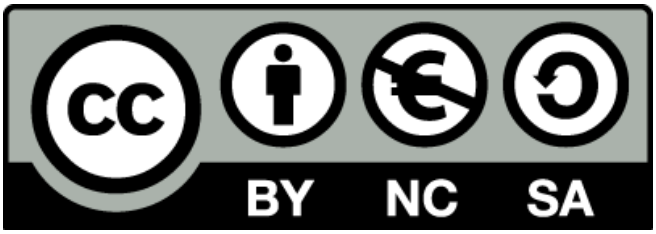
by AS

playground.boxtec.ch/doku.php/tutorial

ARM Controller - SAM D21 J17A
(32 Bit Controller)
Software 2 – Das erste Programm

SAM D21 J17A

Copyright



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 die-ser 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 Versor-gungsspannung 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 Spannungsver-sorgung 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 un-verzüglich 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 Baugrup-pe 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 Ge-fahren, 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 – Das erste Programm

In diesem Teil des Tutorials wollen wir das erste Programm für den SAM D21 erstellen.
Traditionell möchte ich mit dem Programm „Hello World" anfangen.
Die Funktion ist sehr einfach, es soll nur die LED am PB 30 blinken.

Nano SAMD21 Microcontroller Modul auf der Grundplatine 110

Dazu muss ich den PB 30 als **OUTPUT** programmieren um die LED ansteuern zu können.
Der SAM D21 verfügt über eine Reihe von Pins, die auf die Wannenstecker, LEDs und Taster geführt sind. Die LED am PB befindet sich auf der Platine des Nano SAM D21.
Die Ausgangspins werden als **GPIO** bezeichnet.

Der Nano SAM D21 wird mit einer Spannung von +5V versorgt. Auf dem Modul befindet sich Spannungsregler für +3,3V. Alle Ein- und Ausgänge dürfen nur mit einer maximalen Spannung von **+3,3V** betrieben oder angesteuert werden. Jede höhere Spannung kann zu einer sofortigen **Zerstörung** des Moduls führen.
Die Pinausgänge sind nur für einen Strom von 6mA (max.) belastbar. Beim Anschluss von LEDs oder anderer Peripherie muss das unbedingt beachtet werden.
Das gesamte Modul und die angeschlossene Periphere darf mit einem max. Strom von ca. **500mA** belastet werden.

Was sind GPIO`s?

GPIO ist die Abkürzung für General Purpose Input Output. Damit sind Signalleitungen des Mikrocontrollers gemeint die für unterschiedliche Funktionen verwendet werden können, als Eingang oder als Ausgang. Daneben gibt es Signalleitungen welche nur für bestimme Zwecke und Schnittstellen benutzt werden können.

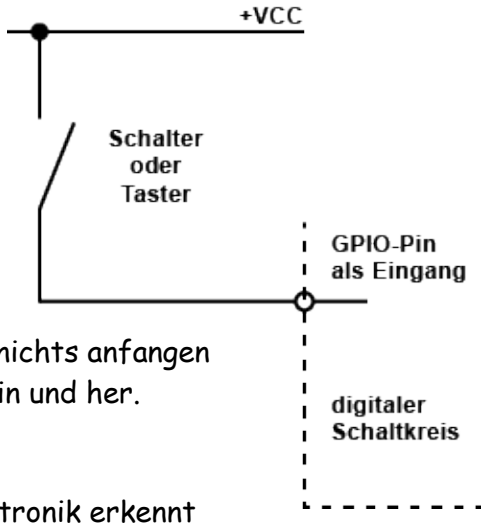
Mit welchen Pegeln (Spannungen) arbeitet die GPIO`s?

Die Spannungspegel am GPIO liegen für „High" bei 3,3V und für „Low" bei 0V. Die „High" und „Low" Pegel unterliegen einer Toleranz. Ein GPIO erkennt eine Spannung unter 0,8V als „Low" und eine Spannung über 1,3V als „High". Wenn keine definierte Spannung an einem GPIO anliegt, neigen sie dazu, zufällig in die eine oder andere Richtung zu schalten. Das kann in unserer Schaltung oder im Programm zu unerwarteten Fehlern führen. In der Regel werden die GPIO`s mit Widerständen beschaltet um sie auf einen definierten Pegel zu setzen (nach Vcc oder GND).

Was sind Pullup- oder Pulldown Widerstände?

Um dem entgegenzuwirken, verhilft man einem GPIO-Eingang zu einem definierten Grundzustand. In der Praxis versieht man den jeweiligen Pin mit einem Pullup- oder Pulldown-Widerstand. Abhängig davon, ob man standardmäßig den Zustand "High" oder "Low" erwartet.
"Pull" heißt "ziehen". Das bedeutet, der GPIO wird auf einen bestimmten Pegel (Spannungswert) "gezogen". Pullup bedeutet auf "High" rauf- und Pulldown auf "Low" runterziehen. Das Hinauf-

ziehen erreicht man, in dem man den GPIO-Pin über einen Widerstand mit einem Pin verbindet, der dauerhaft einen High-Pegel führt. In der Regel wird dafür die Betriebsspannung +Vcc verwendet. Das Hinunterziehen erreicht man, in dem man den GPIO-Pin über einen Widerstand mit dem Ground (GND) verbindet.



Was passiert eigentlich, wenn wir ohne Pullup- und Pulldown-Widerstände arbeiten?

Schalter offen

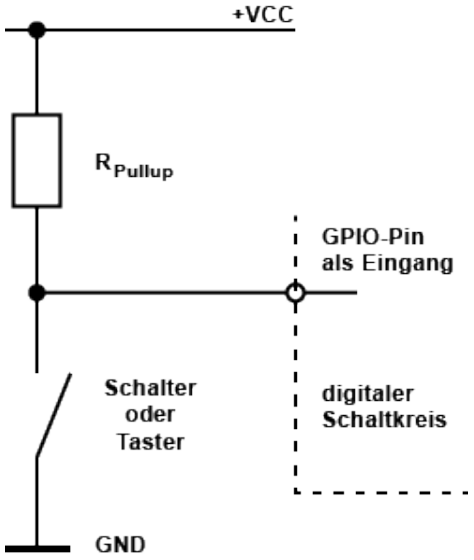
Der GPIO-Eingang hat keine Verbindung zu irgendwas. Er hängt praktisch in der Luft. Die Leitung vom Pin zum Schalter wirkt wie eine Antenne. Und diese Antenne empfängt irgendetwas. Die Eingangs-Elektronik kann damit nichts anfangen und schwingt undefiniert zwischen "low" (0) und "high" (1) hin und her.

Schalter geschlossen

Der GPIO-Eingang wird auf VCC gezogen. Die Eingangs-Elektronik erkennt einwandfrei ein "high" (1).

Welches Verhalten hat ein GPIO-Eingang mit Pullup-Widerstand?

In diesem Schaltbild ist der Widerstand von VCC auf einen GPIO-Eingang geschaltet und der geöffnete Schalter vom GPIO-Eingang auf GND. Das Schaltbild aus Pullup-Widerstand und Schalter kann man sich auch als Reihenschaltung aus zwei Widerständen vorstellen. Wobei der Schalter im offenen Zustand ein unendlich hochohmiger und im geschlossenen Zustand ein niederohmiger Widerstand ist.



Schalter offen

Bei offenem Schalter zieht der Widerstand den GPIO-Eingang gegen VCC. Hier liegen definitiv z. B. +3,3 V, also "high", an. Und deshalb wird dieser Widerstand Pullup-Widerstand genannt. Weil der Widerstand den GPIO auf die Betriebsspannung raufzieht.

Schalter geschlossen

Wird der Schalter betätigt, dann wird der GND mit dem GPIO-Eingang verbunden. Das hat folgenden Effekt: Die Spannung fällt komplett am Pullup-Widerstand ab und dadurch liegt am GPIO-Eingang GND an und somit ein "low".

Wie kann ich eine LED schalten?

Prinzipiell sollte man eine LED nie direkt zwischen 3,3 V und GND schalten. Leuchtdioden haben eine feste Betriebsspannung, die unter 3,3 V liegt und es zusätzlich einer Strombegrenzung bedarf, weil sie sich selbst sonst selbst zerstören würden. Sowohl die Spannung als auch den Strom regelt man mit einem Vorwiderstand, dessen Wert von der Spannung und dem gewünschten Strom durch die Leuchtdiode abhängt.
Außerdem muss man berücksichtigen, dass eine LED zwei unterschiedliche Pole hat. Man muss also darauf achten, wie herum die LED in die Schaltung eingebaut wird.

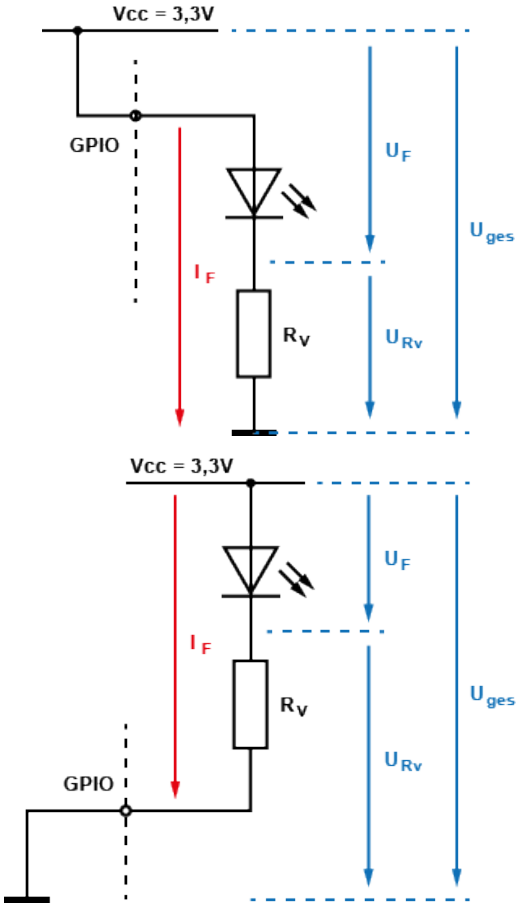
Schaltung 1: LED an Masse

Damit die LED leuchtet, muss der GPIO "intern" auf "high" (1) geschaltet werden. Nur dann fließt ein Strom durch die LED.

Schaltung 2: LED an VCC

Damit die LED leuchtet, muss der GPIO "intern" auf "low" (0) geschaltet werden. Nur dann fließt ein Strom durch die LED.

Die Schaltung 1 wählt man dann, wenn die LED beim GPIO-Zustand "high" (1) leuchten soll. Die Schaltung 2 wählt man dann, wenn die LED beim GPIO-Zustand "low" (0) leuchten soll.



Daten einer LED:

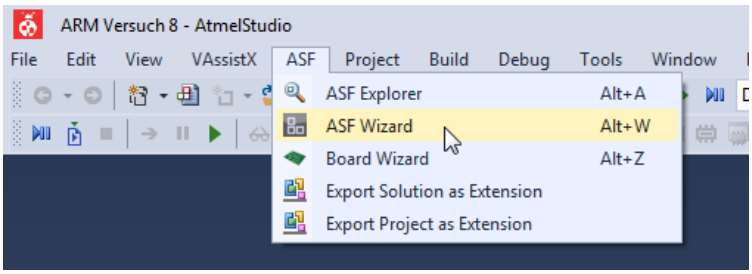
I_F = 2mA
U_F = 2,0V
U_{cc} = 3,3V

$$R_V = \frac{U_{CC} - U_F}{I_F} = \frac{3{,}3V - 2{,}0V}{2\ mA} = 650\ \Omega$$

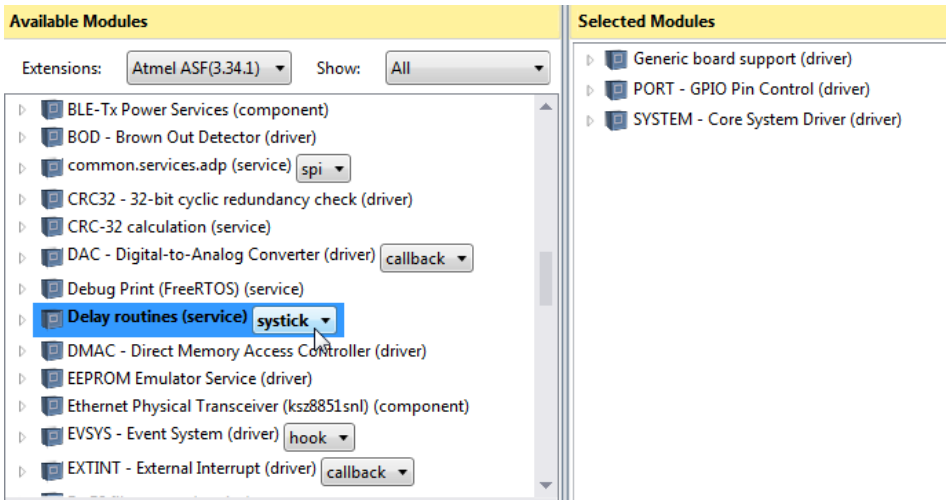
Eine LED mit einem Strom von **2mA** und einem Vorwiderstand von mindestens **650 Ω** (besser einen höheren Widerstand) kann ich direkt am Ausgang des SAM D21 betreiben.

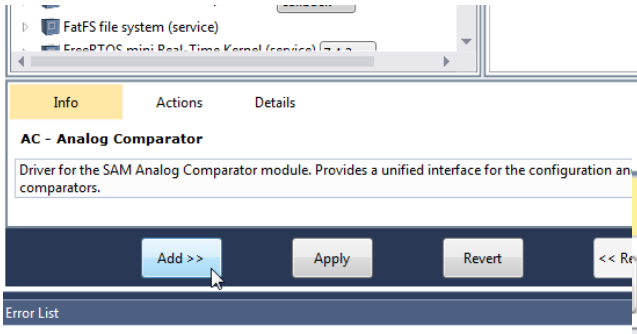
Bevor wir zum ersten Programm kommen, müssen wir noch eine Datei einfügen

- Erstelle ein neues Projekt (siehe Teil Atmel Studio einrichten)
- Wähle den SAM D21 aus
- Wähle im ASF-Wizard aus
- Port - GPIO Pin Control (driver) auswählen und einfügen
- Delay routines auswählen
- mit Add auswählen
- mit Apply bestätigen

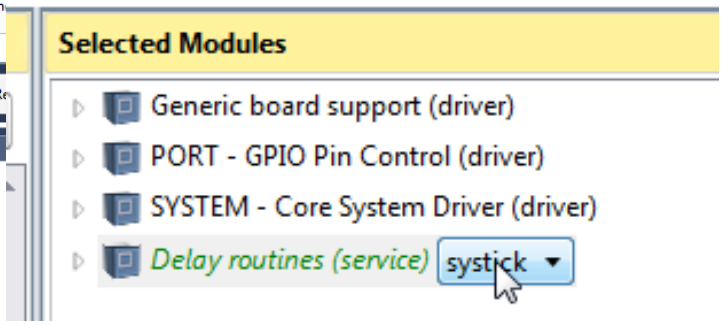


Auswahl von
Delay routines (service)

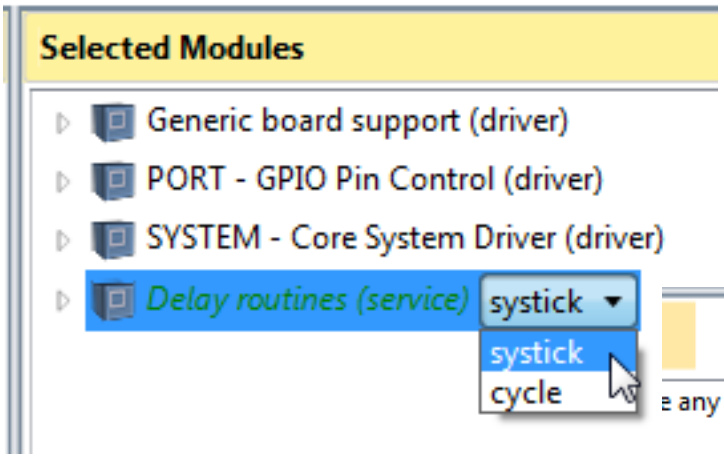




Mit **Add** bestätigen

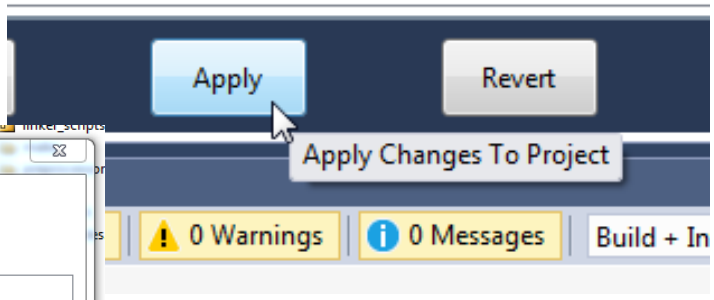


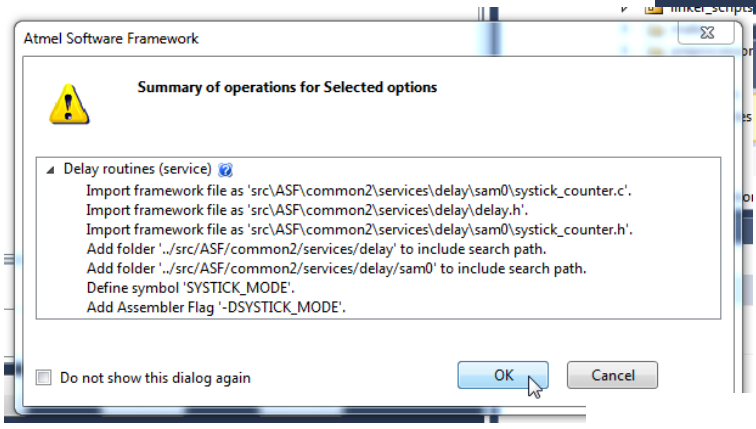
Es erscheint **Delay routines (service)** in grüner Schrift. Es kann zwischen systick und cycle ausgewählt werden.



Bei der Auswahl von systick muss im Programm (main) einmal **delay_init()** aufrufen werden.

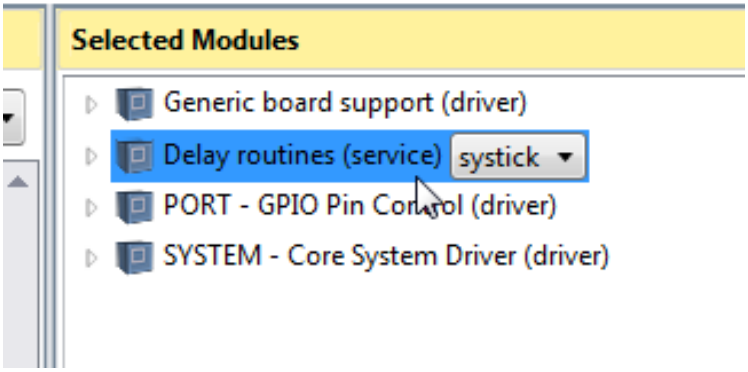
Danach mit Apply bestätigen





Es erscheint noch mal eine Zusammenfassung der auszuführenden Operationen. Bitte mit **OK** bestätigen

Es werden **Delay routines (service)** und **PORT-GPIO Pin Control (driver)** in schwarzer Schrift angezeigt.
Damit können sie in unserem Programm verwendet werden.



```
// ARM Board_3 Programm 2 by HJS

#include <asf.h>                          // Angabe von ASF
#define LED1  PIN_PB30                    // Angabe welcher Port/Pin

void configure_pins(void);                // Bekanntgabe Unterprogramm

void configure_pins(void)                 // Unterprogramm Pins init
 {
   struct port_config config_port_pin;    // erzeugt eine Struktur mit verschiedenen Variablen
   port_get_config_defaults(&config_port_pin);      // wird mit Default Einstellungen initiasiert
   config_port_pin.direction = PORT_PIN_DIR_OUTPUT;          // Variable direction mit
                                                             PORT_PIN _OUTPUT füllen
   port_pin_set_config(LED1, &config_port_pin);  // Es wird ein PIN als Ausgang eingestellt
 }

int main (void)                           // Beginn Hauptprogramm
 {
   system_init();                         // System wird init
   delay_init();                          // delay init
   configure_pins();                      // Pins init Unterprogramm
   while(1)                               // Wiederholung
    {
      port_pin_toggle_output_level(LED1); // LED toggeln (umschalten)
      delay_ms(1000);                     // Pause 1000ms
    }
 }
```

Erläuterung der Funktion:

Als erstes beginnen wir mit dem Einbinden von „**asf.h**“, danach wird für PB30 eine symbolische Konstante (**#define**) definiert.
Vor der Main-Routine befindet sich eine Funktion, die die Pins konfiguriert. Dort wird zuerst mit dem Schlüsselwort **struct** eine Struktur des Typs **port_config** namens **config_port_pins** erzeugt. Die Struktur enthält verschiedene Variablen, welche die einzelnen GPIO-Einstellungen speichern. Auf die einzelnen Variablen einer Struktur kann man mit dem Punktoperator zugreifen. Wenn man einer Funktion platzsparend das ganze Bündel von Einstellungen mitgeben will, muss man nur einen Zeiger (Speicherplatz-Adresse) auf eine solche Struktur als Parameter übermitteln.
Nachdem die Struktur erzeugt ist, wird sie per Befehl **port_get_config_defaults(&config_port_pins)** mit Default-Einstellungen (Eingang mit Pull-up) initialisiert.
Danach folgt die eigentliche, von der Anwendung abhängige Konfiguration. Hier wird mit dem nächsten Befehl die Variable **direction** innerhalb der Struktur mit der Konstanten **PORT_PIN_DIR_OUTPUT** gefüllt. Mit dem nachfolgenden Befehl wird einer der Pins als Ausgang eingestellt: **port_pin_set_config(LEDG, &config_ port_pin)**.
Wie zu sehen, wird dieser Funktion nur der Pin (hier die symbolische Konstante) und ein Zeiger auf die Konfigurations-Struktur übergeben. In der Main-Routine erfolgt der schon bekannte Befehl **system_init()** und dann der Befehl **delay_init()**, der eine Initialisierung des SysTick-Timers für die Delay-Funktion bewirkt. Der Befehl **configure_port_pins** ruft die eben beschriebene Konfigurations-Funktion auf.

```
// ARM Board_3 Prg 3 by HJS

#include <asf.h>                                  // delay und GPIO einbinden
//define the pins
#define LED1  PIN_PA03                            // PA03 / PB30 - 1. Zahl LED+Taster auf P110
#define Button1   PIN_PB23                        // PB23 / PA13 - 2. Zahl LED+Taster auf Nano SAM

void configure_port_pins(void);

void configure_port_pins(void)                    // configuration of the 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();                                 // initialization
   delay_init();
   configure_port_pins();                         // configuration of the pins
   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
         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
       }
    }
 }
```

Funktion des Programmes:

Die **LED1** blinkt in einem Takt von 1 Sekunde (500ms ein, 500ms aus). Mit dem Taster **Button1** wird das blinken ein- oder ausgeschaltet.
In diesem Programm werden wieder Teile aus dem ersten Programm verwendet. Mit **#define** kann ich auswählen welcher Taster oder LED verwendet wird. Bei Taster nicht gedrückt oder Taster gedrückt kann ich auswählen, welchen Taster ich verwende – nach **Vcc** oder **GND**.

Die Funktion Einbindung der LED1 entspricht der Erklärung im ersten Programm. Zusätzlich wird hier noch der Taster (Button1) definiert.

```
/* ARM Board 3 Prg 4 by HJS */

#include <asf.h>

#define LEDG  PIN_PB30                                  // LED PB30
#define LED1  PORT_PB08|PORT_PB09                       // Gruppe LED
#define Button1  PIN_PB22                               // Taster1
#define Button2  PIN_PB23                               // Taster2

void configure_port_pins(void);

void configure_port_pins(void)       // setup of the configuration of the 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(LEDG, &config_port_pin);

   // Angabe Taster 1
   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 Taster 2
   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(Button2, &config_port_pin);

   // Gruppe LED
   port_get_config_defaults(&config_port_pin);
   config_port_pin.direction = PORT_PIN_DIR_OUTPUT;
   port_group_set_config(&PORTB, LED1, &config_port_pin);
 }

int main (void)
 {
   system_init();                           // initialization
   delay_init();
   configure_port_pins();                   // configuration of the pins
   while (1)
    {
      //if (port_pin_get_input_level(Button1))          // Taste 3 gedrückt / nicht
      if (!port_pin_get_input_level(Button1))          // Taste 3 gedrückt / nicht
       {
         port_pin_toggle_output_level(LEDG);           // LED PB30
         delay_ms(500);                                // warten
         port_pin_toggle_output_level(LEDG);           // LED PB30
         delay_ms(500);                                // warten
       }
      //if (port_pin_get_input_level(Button2))          // Taste 4 gedrückt / nicht
      if (!port_pin_get_input_level(Button2))          // Taste 4 gedrückt / nicht
       {
         port_group_toggle_output_level(&PORTB, LED1 );  // LED Gruppe 1
```

```
            delay_ms(500);                                  // warten
            port_group_toggle_output_level(&PORTB, LED1 );  // LED Gruppe 1
            delay_ms(500);                                  // warten
          }
        }
      }
```

Funktion des Programmes:

Es werden eine LED, zwei Taster (3, 4) (**PB22, PB23**) und eine LED Gruppe definiert. Mit dem Taster 3 kann die LED **PB30** blinken / nicht blinken geschaltet werden. Mit dem Taster 4 kann die LED Gruppe 1 blinken / nicht blinken geschaltet werden. Mit LED1 können noch mehr oder andere LEDs angegeben werden. Dabei bitte auf den Port achten.

Durch die Zeiten innerhalb der beiden Schleifen kann es zu unterschiedlichen Anzeigen kommen. Erklärung zur Funktion bitte dem ersten Programm entnehmen. Bei Problemen mit den einzelnen Befehlen bitte in der ASF Doku nachlesen.

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 2015 - Von 8 auf 32 bit von Viacheslav Gromov
https://www.makerconnect.de Pogrammierung SAM D21 mit Atmega ICE und SWD
https://www.elektronik-kompendium.de/sites/raspberry-pi/2006041.htm