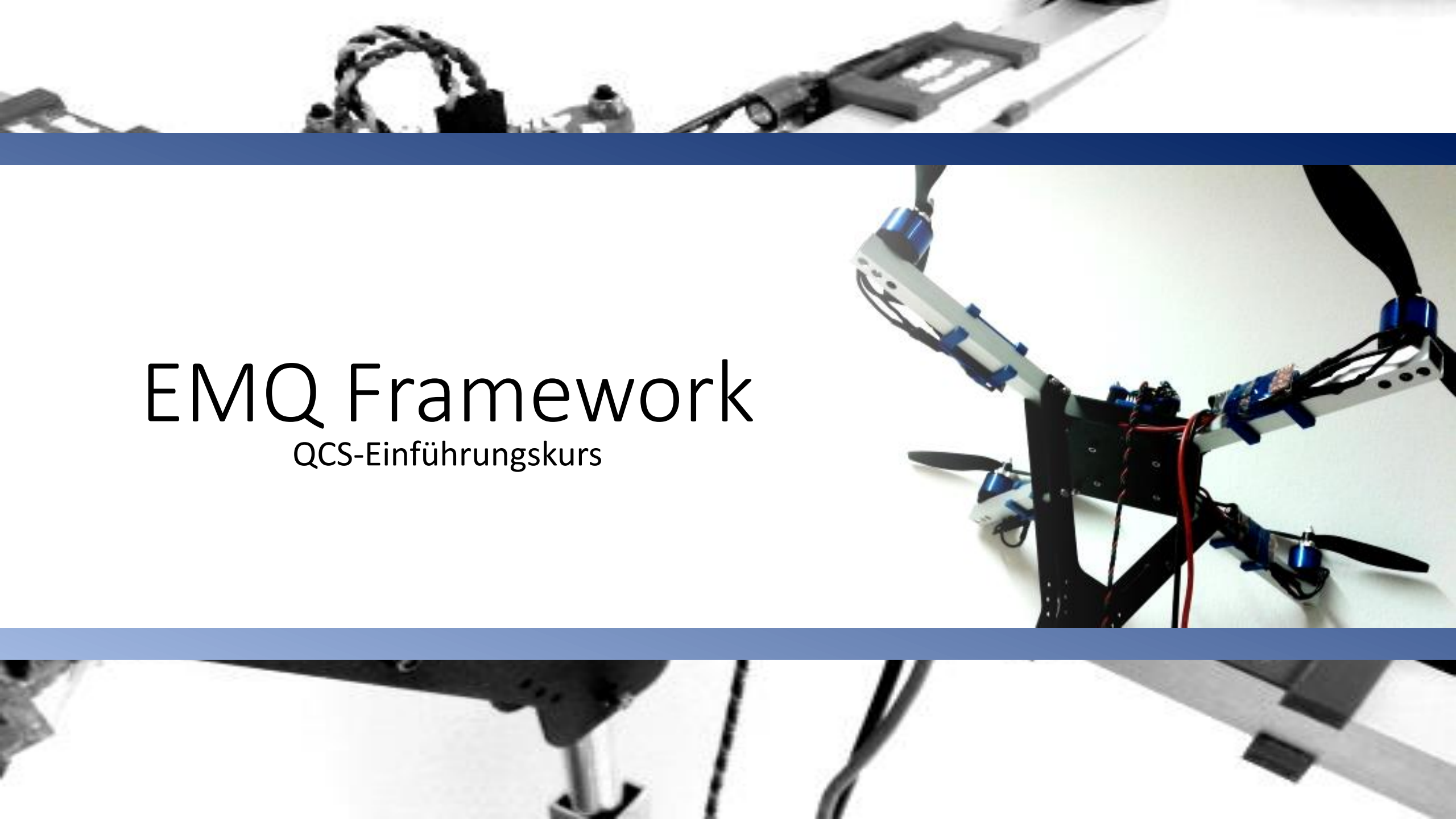


EMQ Framework

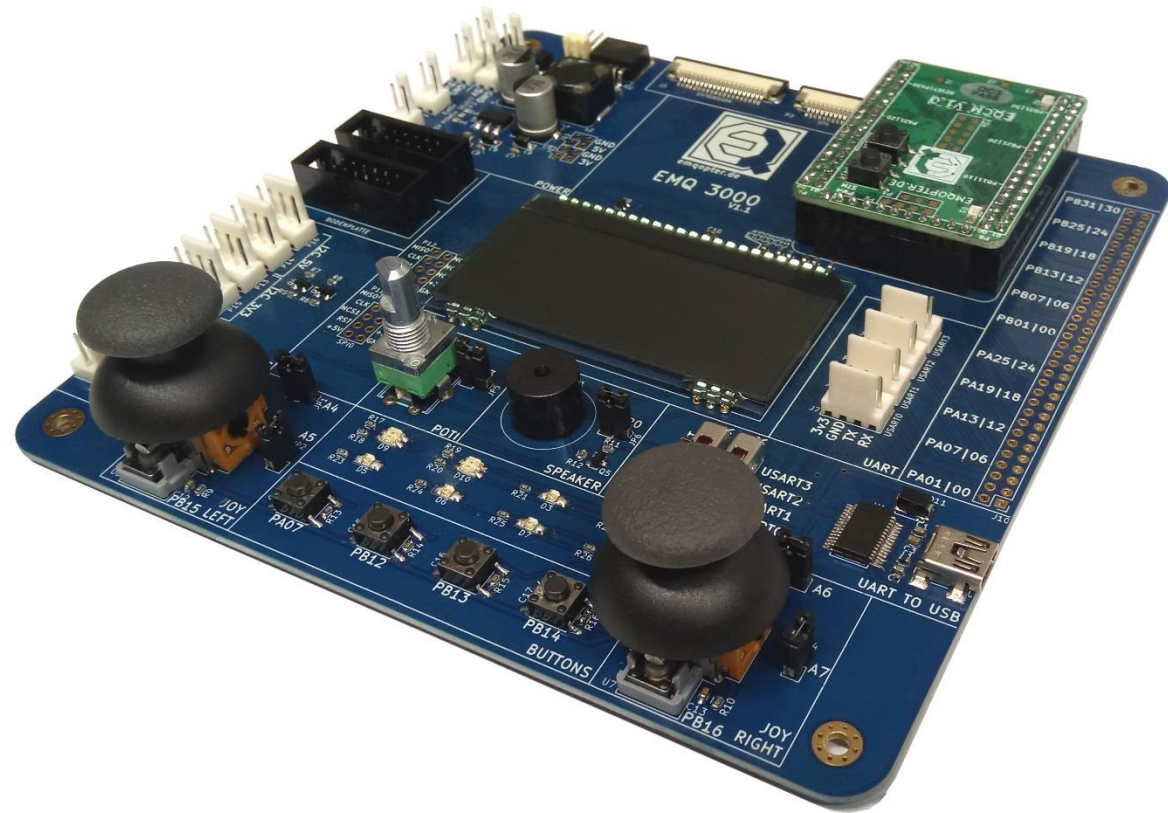
QCS-Einführungskurs



Inhalt

Umfang: ca. 2-3 Zeitstunden

- Was bedeutet Embedded?
- Das Entwicklungsboard EMQ3000
- Grundlagen Programmieren
- Die Software-Bibliothek
- Display DOGM204
- Aufgaben





Embedded

Was bedeutet Embedded?

- Embedded = Eingebettet
- Ein Prozessor / Computer / Rechner wird in einem Prozess oder Ablauf eingebettet
- Beispiele:
 - Waschmaschine: Computer steuert Waschprogramm
 - Cola-Automat: Computer steuert Bestellung
 - Drucker: Computer steuert Druckvorgang
 - Smartphone: Computer steuert telefonieren (u.a.)
 - **Drohne**: Computer steuert Flug
- Embedded finden Sie heute überall





Embedded

Embedded – Besonderheiten am Beispiel Drohne:

- Echtzeit: Drohne muss Ihre Aufgaben (z.B. Lageregelung) **rechtzeitig** erledigen, sonst stürzt Sie ab
- Ressourcenknappheit: nur begrenzte Ressourcen wie Speicher, Rechenpower, weil Rechner muss mitfliegen
- Black-Box: Der Rechner macht etwas, man weiß aber nicht immer so schnell, was er macht, weil meist kein Monitor vorhanden ist, wie z.B. beim PC
- Hardwarenahe Programmierung:
Man hat meist mehr Freiheiten, kann oft aber auch mehr falsch machen
- Typische Hardware: Micro Controller Unit == MCU
Mikroprozessor: Kleiner Prozessor vs. z.B. Intel CPU oder NASA Supercomputer
- Typische Programmiersprache: **C/C++**



Supercomputer der NASA: Etwas groß, für Drohnen

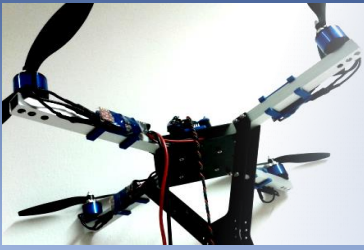


Embedded

Embedded – Besonderheiten am Beispiel Drohne:

- Programm startet, sobald Strom eingeschaltet wird
- Netzteil / Akku anstecken-> Programm läuft sofort
-> Motoren können jederzeit einschalten (theoretisch)
- Programm ist permanent auf der Drohne gespeichert
-> im sogenannten Flash-Speicher
- Wenn Änderungen am Programm erfolgen (sollen),
z.B. nach dem Programmieren = Erstellen neuen Codes,
dann muss man erneut **kompilieren & flashen**





Embedded

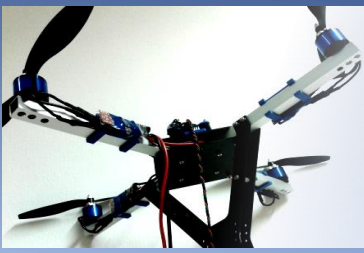
Embedded – Besonderheiten am Beispiel Drohne:

- Vorgehen stets aus vier Schritten: Programmieren, Kompilieren, Flashen & Ausführen
- Kompilieren -> Übersetzen des Programmcodes in Maschinencode:
Menschen verstehen Programmcode (z.B. C), -> Hochsprache
Maschinen sprechen NUR Maschinencode (01010110) -> schlimmer als Assembler

Hochsprache == Englisch -> Versteht jeder

Maschinensprache == Mandschu -> Kompliziert und sehr selten (100 Sprecher)

- Flashen: Programmcode auf Rechner in Flash-Speicher kopieren
Wir benutzen dazu einen USB-basierte Bootloader, der dies übernimmt,
wobei der Prozessor per Knopfdruck gestoppt wird, sonst läuft er immer weiter:
Somit wird ihm mitgeteilt, dass nun ein neues Programm kommt.



Embedded

Flash-Speicher:

- Digitaler Speicher (z.B. USB-Stick)
- Nichtflüchtig
- UC3A1512 (CPU des EVK) hat 512 Kilobyte Flashspeicher (Das ist viel für MCU)
- Flashen meint das Programmieren dieses Speichers zum Laden des Programmes auf MCU

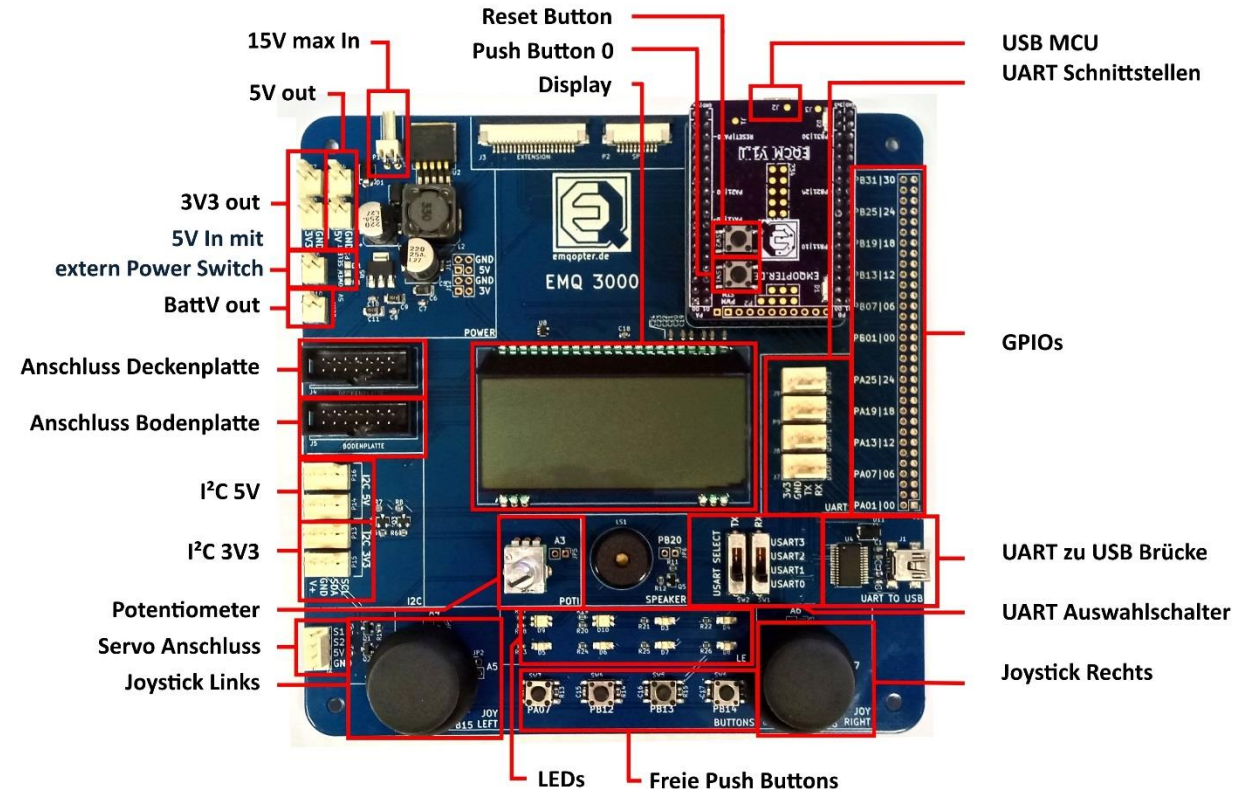
Funktionsweise (vereinfacht):

- Daten = Ladungen auf Floating-Gate eines MISFET:
Metall-Isolator-Halbleiter-Feldeffektransistor
- Floating-Gate durch Dielektrikum isoliert -> kein Abfließen
-> Vereinfacht ausgedrückt: Isolierter Transistor (Transistor = Schalter, hier wie Speicher)
- Heut zutage: Multi-Level-Cell-Speicherzellen

EMQ3000

Entwicklungsboard EMQ3000:

- Alles einfach programmierbar
- Umfangreiche Peripherie
- Peripherie = Alles, was nicht direkt CPU ist
- MCU / Mikroprozessor Name:
32UC3A1512-U

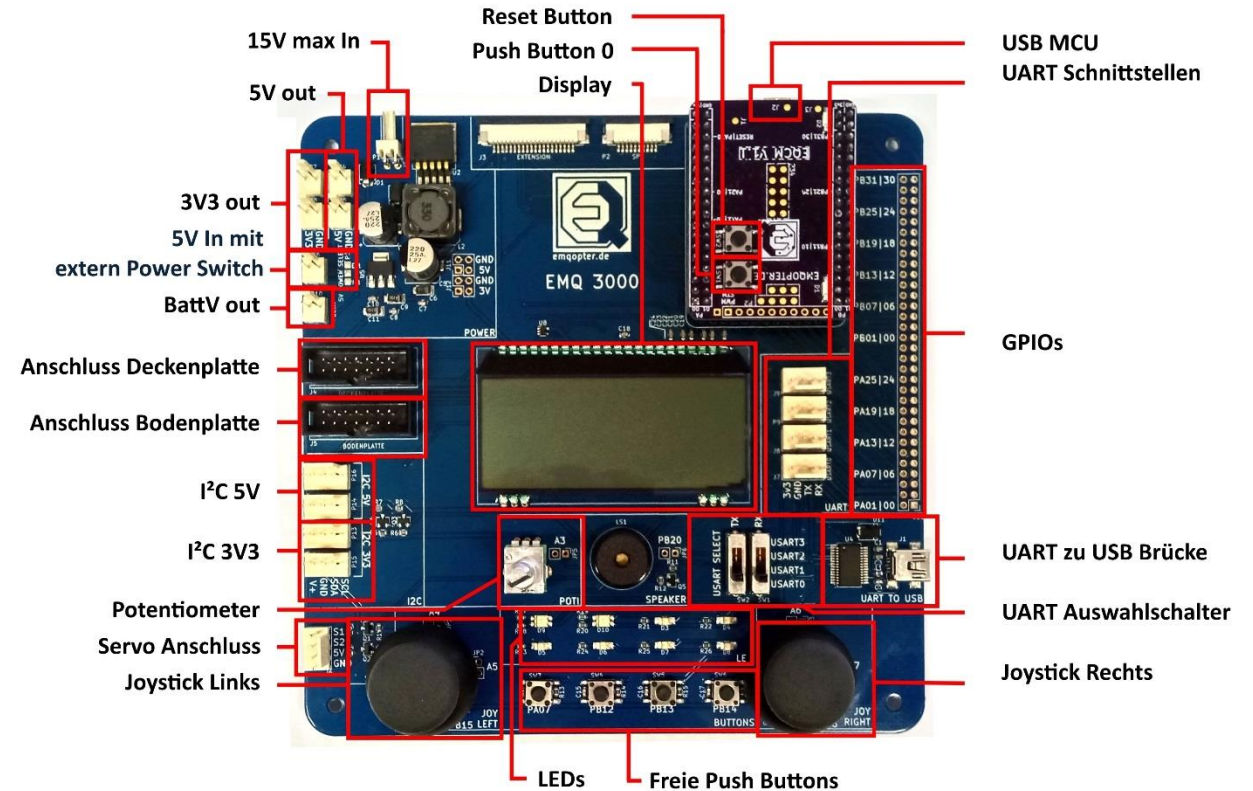


EMQ3000

Entwicklungsboard EMQ3000:

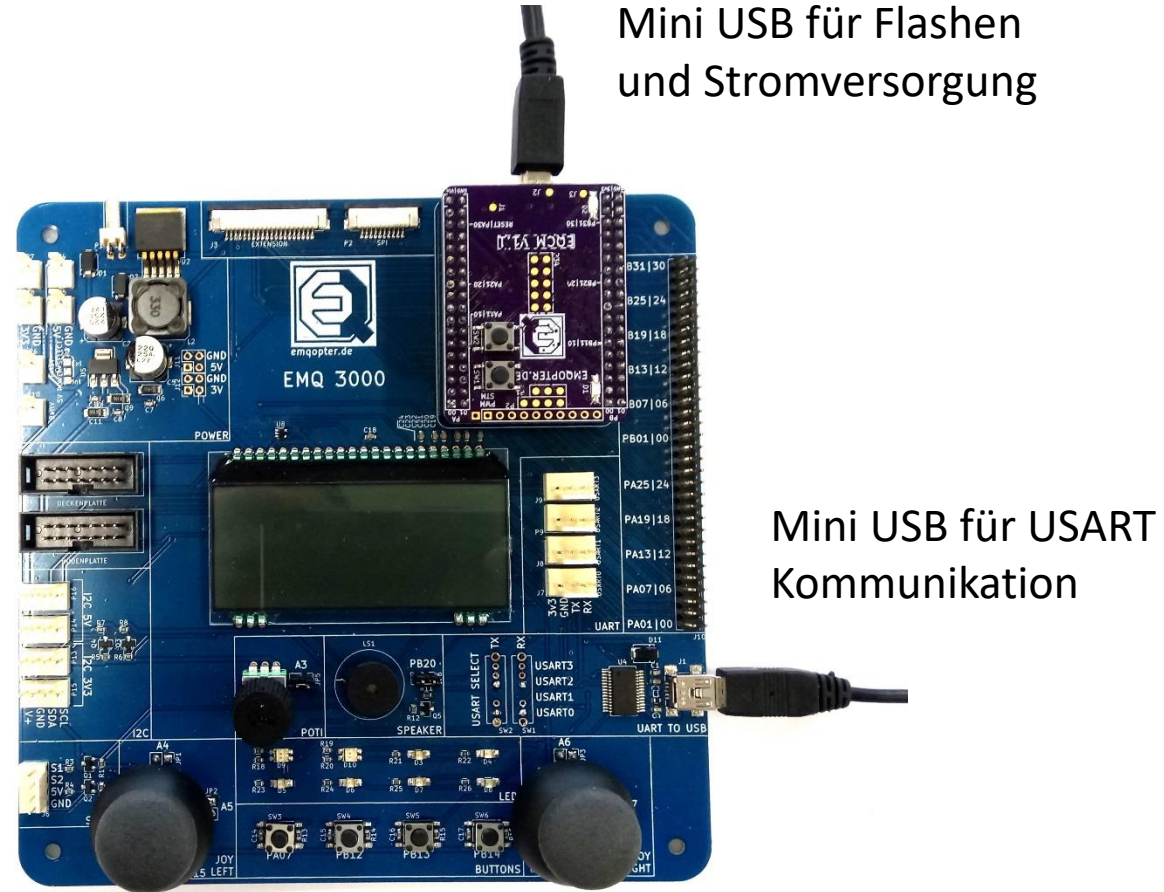
Peripherie:

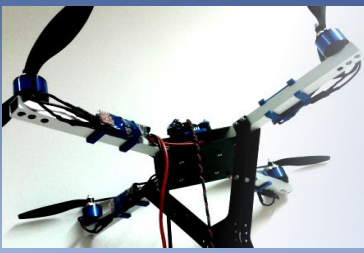
- USART = Serielle Kommunikation RS232, Einfacher Vorgänger von USB
- PINs = Kontakte, für alles = GPIO
General Purpose Input Output
Allgemeines rein und raus
- Klassische Schnittstellen:
 - I²C bzw. TWI
 - SPI
- Knöpfe zum Steuern (Buttons)
- Display für Anzeigen (DIP abgekürzt)



EMQ3000

Anschluss an PC:





Vorbereitungen

Notwendige Hardware:

- EMQ3000
- QCS / QCSF
- Mikro USB Kabel für Strom und zum Flashen

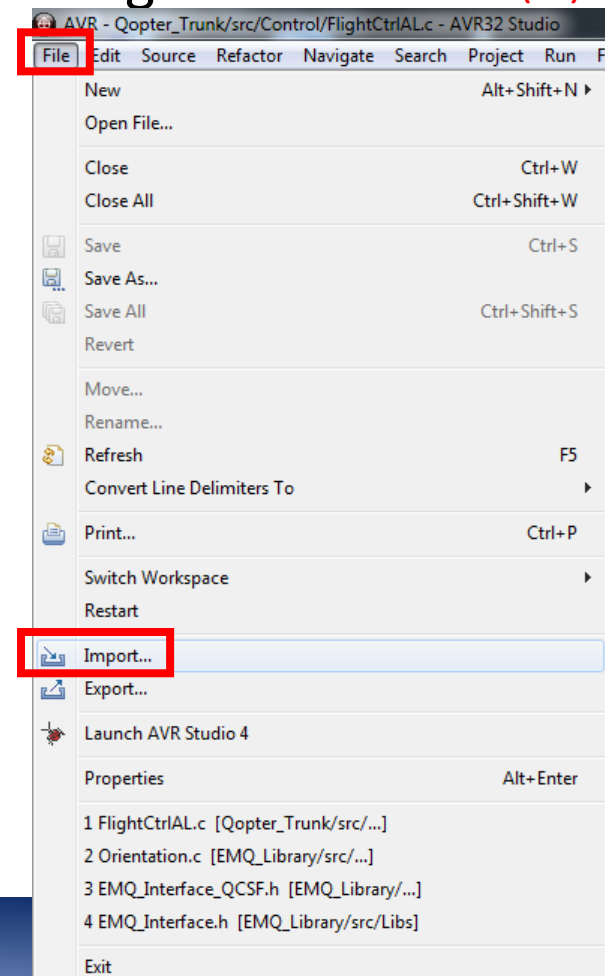
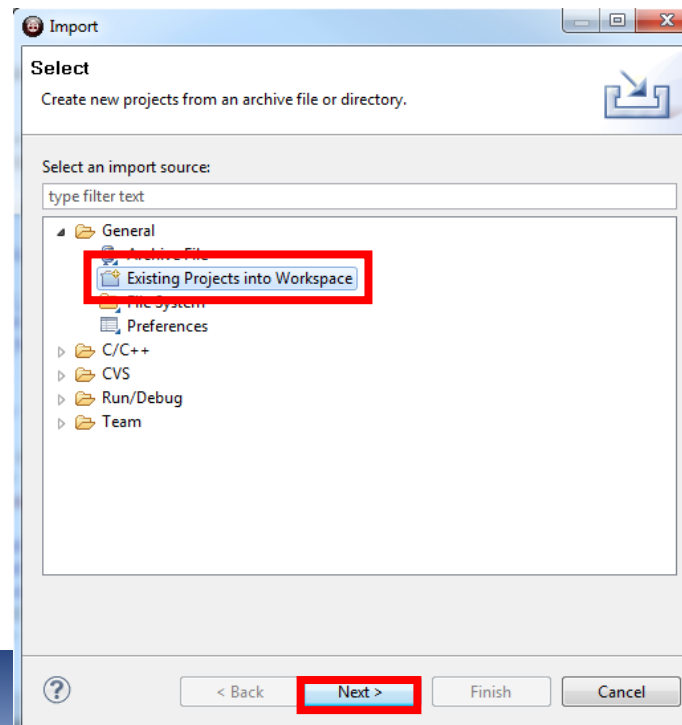
Notwendige Software:

- AVR Studio 32 installiert (mit Tool Chain und FLIP Treiber)
- Projekt-Code (Framework + Lösung) und Library:
EMQ_Framework_Basic -> Legen wir auf „C:\EMQ_Workspace“ ab

Grundlagen zum Programmieren

Los geht's: -> Wir schreiben Hello World

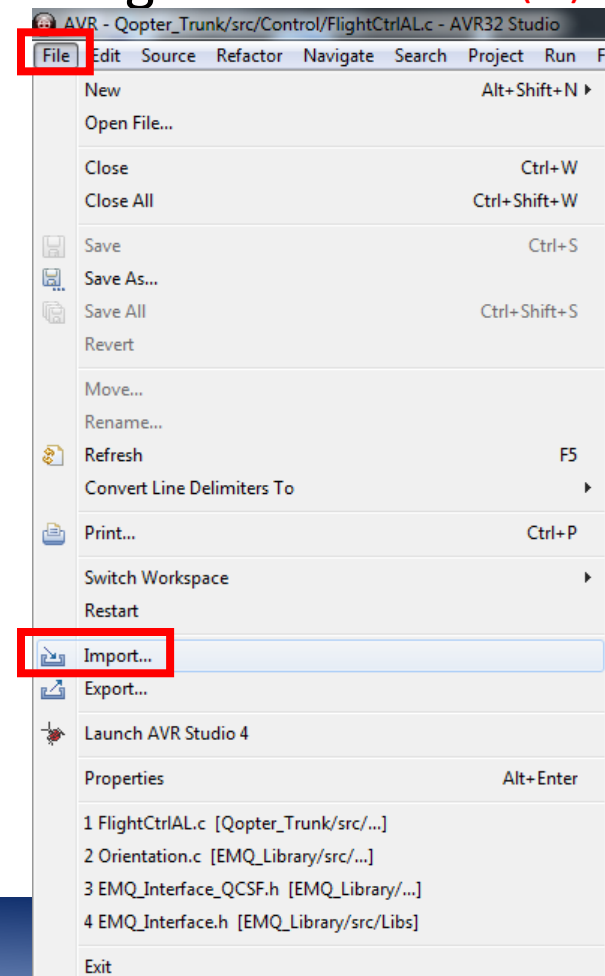
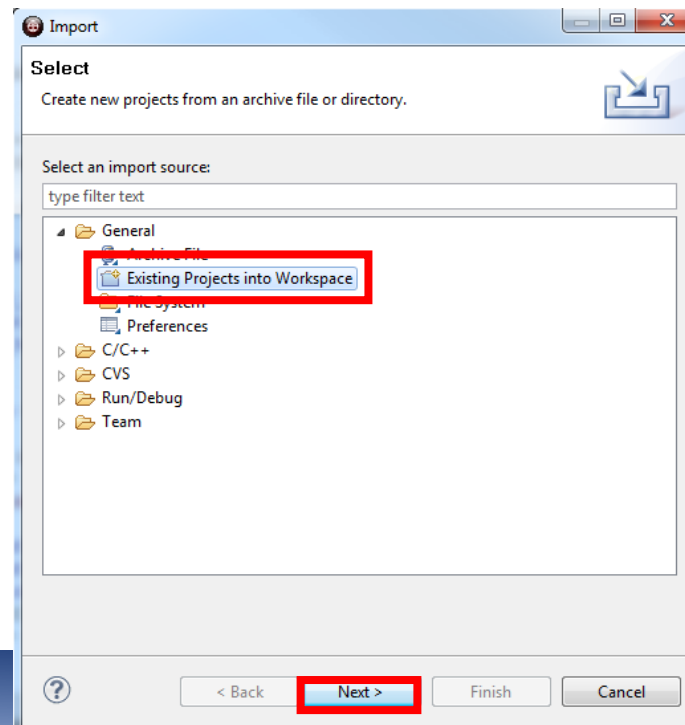
- IDE: AVR32 Studio starten IDE = Integrierte Entwicklungsumgebung
- IDE ist ein Softwarepaket, um einfach programmieren zu können
- Projekt öffnen: File -> Import
- General -> Existing Project into Workspace auswählen
- Dann auf „Next“ klicken



Grundlagen zum Programmieren

Los geht's: -> Wir schreiben Hello World

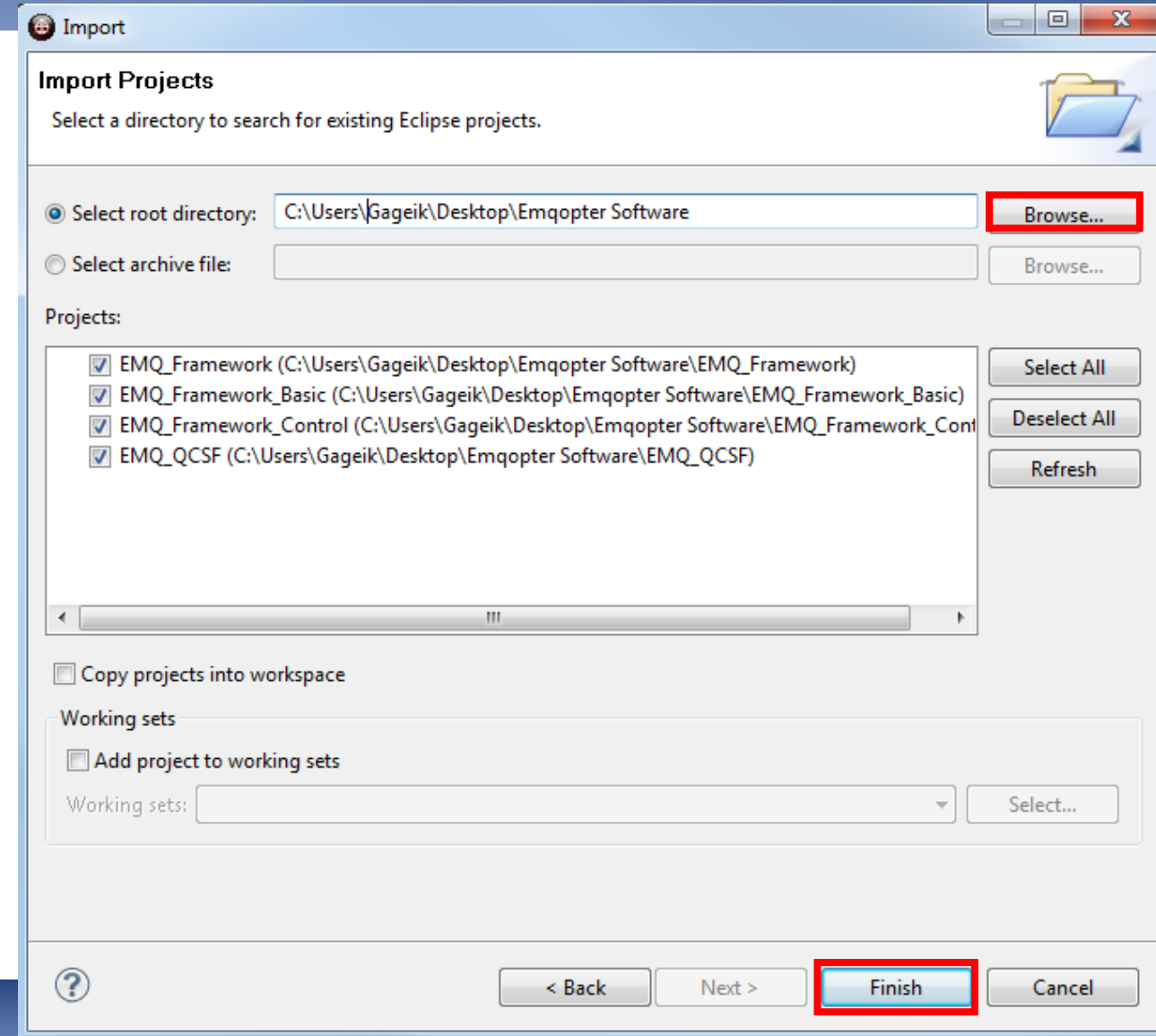
- IDE: AVR32 Studio starten IDE = Integrierte Entwicklungsumgebung
- IDE ist ein Softwarepaket, um einfach programmieren zu können
- Projekt öffnen: File -> Import
- General -> Existing Project into Workspace auswählen
- Dann auf „Next“ klicken



Grundlagen zum Programmieren

Los geht's: -> Wir schreiben Hello World

- Über „Browse“ Ordner in dem die Emqopter Software liegt auswählen
- Es erscheinen die 4 Projekte: Mit Finish werden die vier Projekte importiert.
- Es ist auch möglich einzelne Projekte aus einzelnen Ordnern zu importieren.



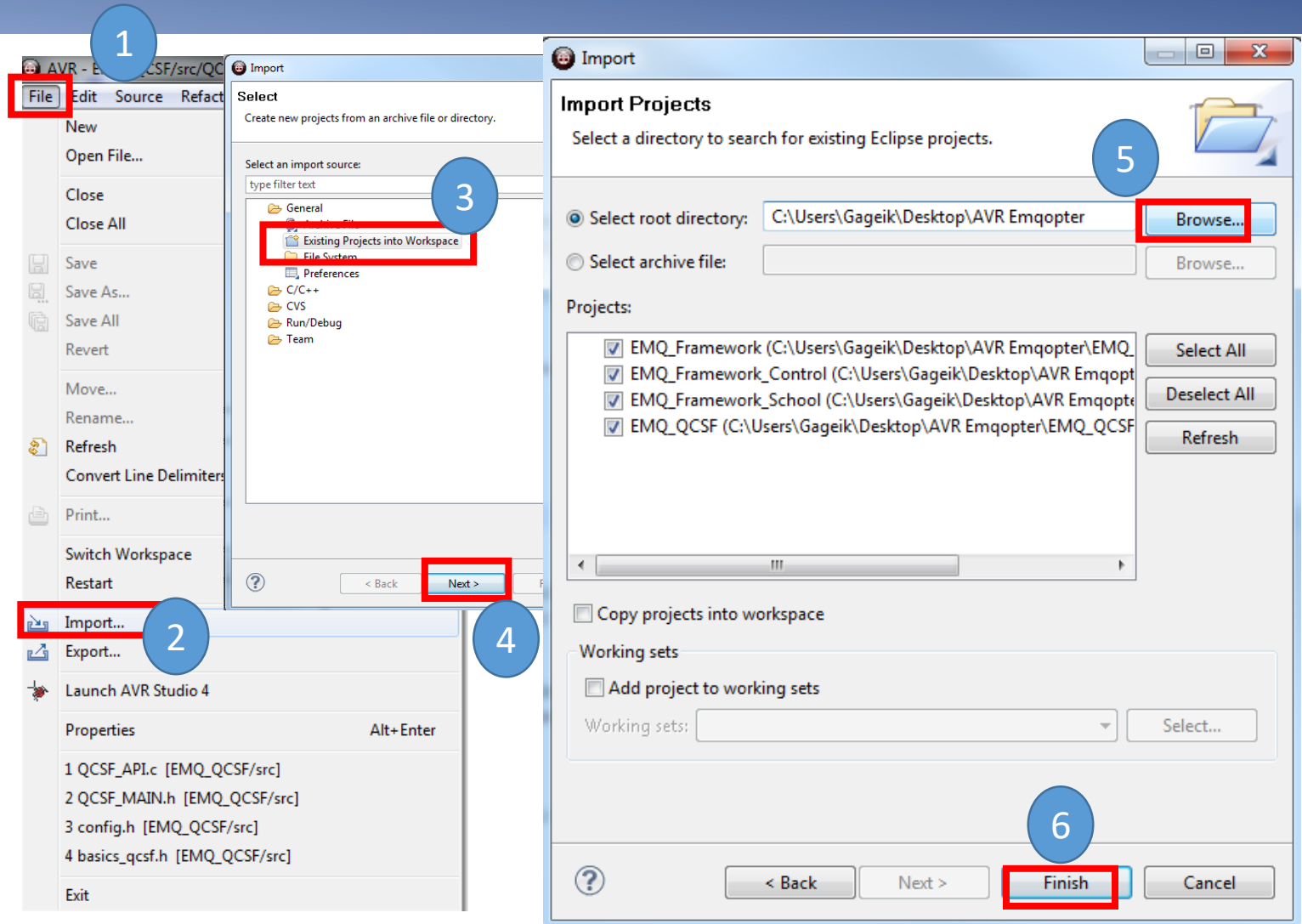
Grundlagen zum Programmieren

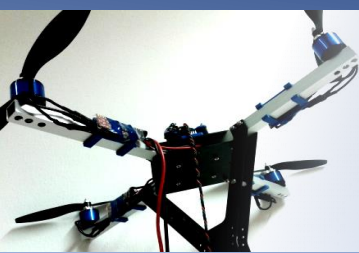
Das ganze auf einer Seite:

Projekt importieren:

- Wähle im Menü: „File -> Import“
- Selektiere „General -> Existing Projects into Workspace“,
- Klicke auf Next>
- Klicke auf Browse (Neben Select root directory) und wähle folgenden Ordner vom Desktop: „AVR Emqopter“
- Klicke Finish
- Die Projekte sollten nun im Project Explorer zu sehen sein

Hinweis: Es kann immer nur ein Projekt mit demselben Namen importiert werden.





Grundlagen zum Programmieren

AVR - EVK1100 - COMPONENTS - DIP204 example/src/dip204_example.c - AVR32 Studio

File Edit Source Refactor Navigate Search Project Run Framework Window Help

Project Explorer

- Attitudecontroller_IMU
- EVK1100 - COMPONENTS - DIP204 example
 - Software Libraries
 - Includes
 - src
 - SOFTWARE_FRAMEWORK
 - dip204_example.c
- EVK1104 - DRIVERS - USART example
 - GierController_AVR2
 - RollController_AVR2
 - RollController_IMU
 - USART_IRQ_2

Projekt-Dateien:
*.c ist die Source-Datei

Hier steht der C-Code

```
/* Section deviceinfo Device info
 * All AVR32UC devices with an SPI module can be used. This example has been tested
 * with the following setup:
 * -- EVK1100 evaluation kit
 *
 * \section setupinfo Setup Information
 * CPU speed: <i>12 MHz</i>
 *
 * \section contactinfo Contact Information
 * For further information, visit
 * <A href="http://www.atmel.com/products/AVR32/">Atmel AVR32</A>.\n
 * Support and FAQ: http://support.atmel.no/
 */

#include "board.h"
#include "compiler.h"
#include "dip204.h"
#include "intc.h"
#include "gpio.h"
#include "pm.h"
#include "delay.h"
#include "spi.h"
#include <avr32/io.h>

/*! define the push button to see available char map on LCD */
#define GPIO_CHARSET GPIO_PUSH_BUTTON_0

/*! define the push button to decrease back light power */
#define GPIO_BACKLIGHT_MINUS GPIO_PUSH_BUTTON_1

/*! define the push button to increase back light power */
#define GPIO_BACKLIGHT_PLUS GPIO_PUSH_BUTTON_2

/*! flag set when joystick display starts to signal main function to clear this display */
unsigned short display;

/*! current char displayed on the 4th line */
unsigned short current_char = 0;

/*!
 * \brief The Push Buttons interrupt handler.
 */
#if __GNUC__
__attribute__((interrupt))
#elif __ICCAVR32__

```

AVR Registers

Register	Address	Description

AVR Targets

Name	Adapter	Board	MCU
AVR32 Simulat...	AVR32 Simulat...	AVR32 Simulat...	UC3...
EVK1100	USB DPU	EVK1100	UC3...

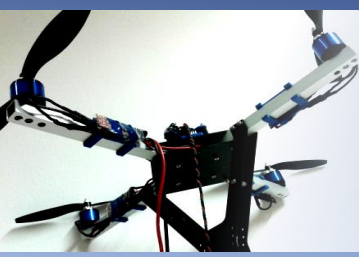
AVR32 Console

Executing cmd.exe /C avr32program -l (WAIT)

Unten Kompilier-Nachrichten:
- Console Klartext
- Problems: Übersicht, aber nicht immer vollständig

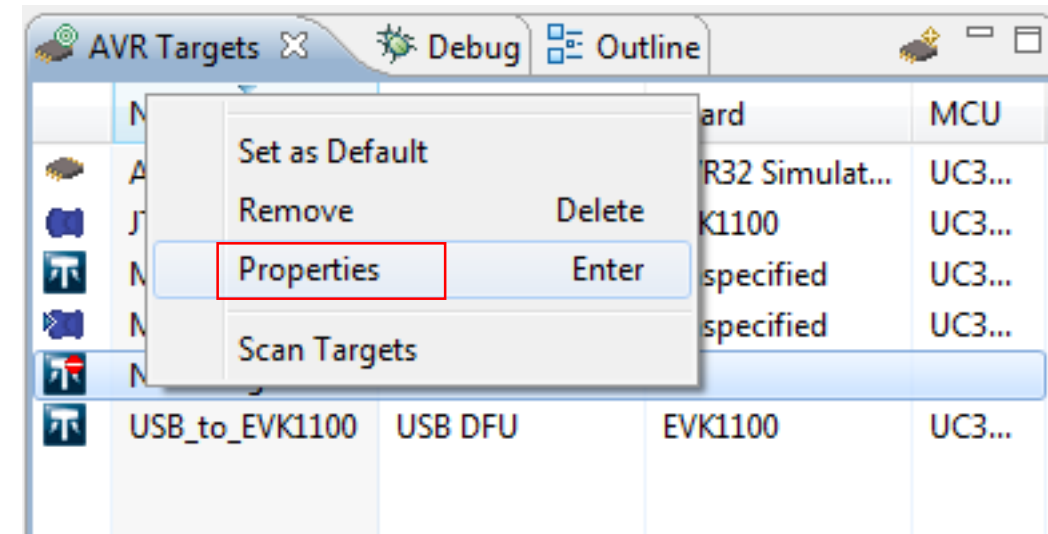
Rechts unten ist das Target zum Flashen

Grundlagen zum Programmieren



Target hinzufügen

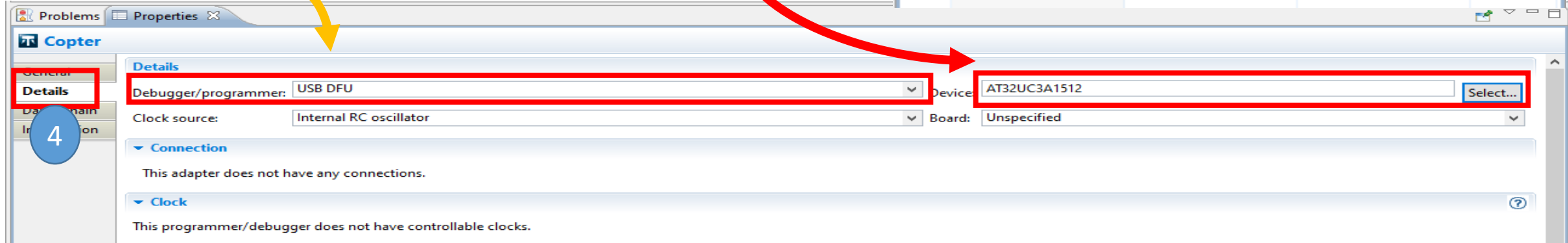
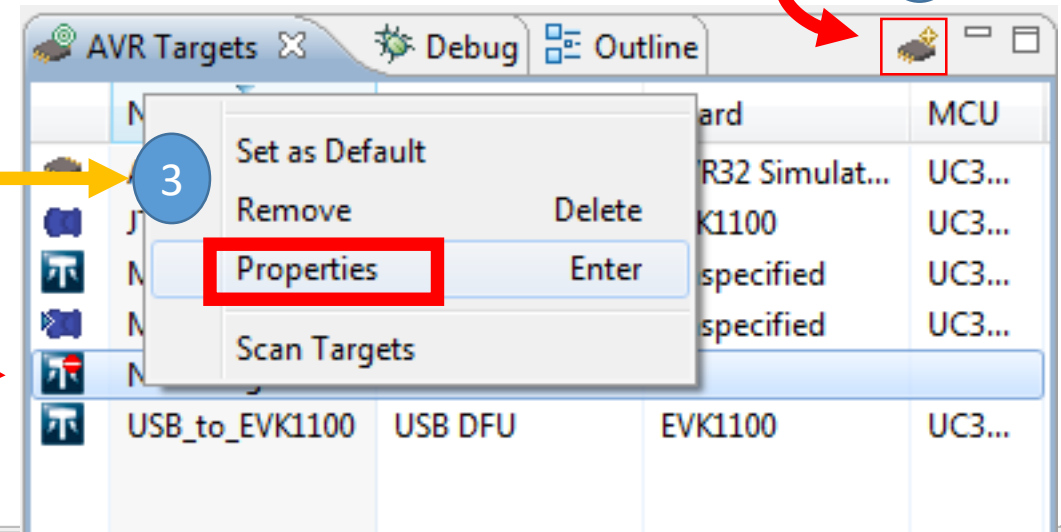
- Target bedeutet Ziel
- Mit einer IDE können verschiedene MCUs programmiert werden
- Wir müssen nun einstellen, was für eine MCU, d.h. welchen Mikrocontroller / Rechner wir haben
- Unserer heißt: *AT32UC3A1512*
 - AT32 wegen Atmel 32bit (Unternehmen + Architektur)*
 - UC3A ist seine Familie*
 - 1 ist der Typ*
 - 512 ist der Flashspeicher (Größe in kB)*
- Wieso? Z.B. der Maschinencode ist bei jedem Target ein ganz anderer
- D.h. z.B. die Sprache muss stimmen
- Beispiel: Deutsche Bewerbungsunterlagen für Stelle in China
-> Das kann nix werden.



Grundlagen zum Programmieren

Target hinzufügen

1. Auf „Create a new target“ links-klicken
Ein „New Target“ erscheint in der Liste
2. Auf „New Target“ rechts-klicken
3. Properties links-klicken
4. Folgendes in Details einstellen
Device: **AT32UC3A1512**
Debugger/ programmer: **USB DFU**



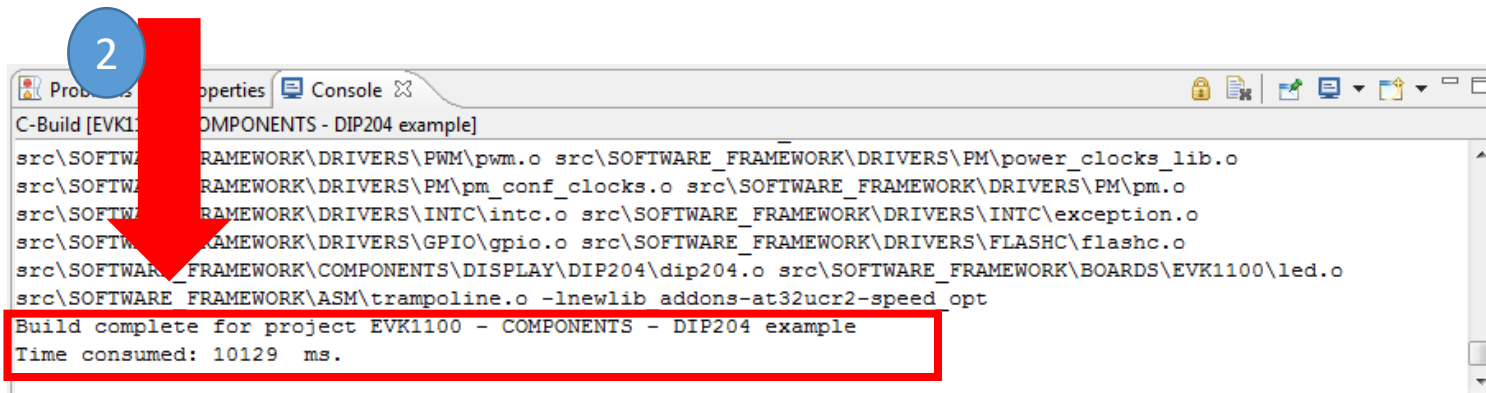
Grundlagen zum Programmieren

Es folgt das Flashen / Programmieren in 4 Schritten

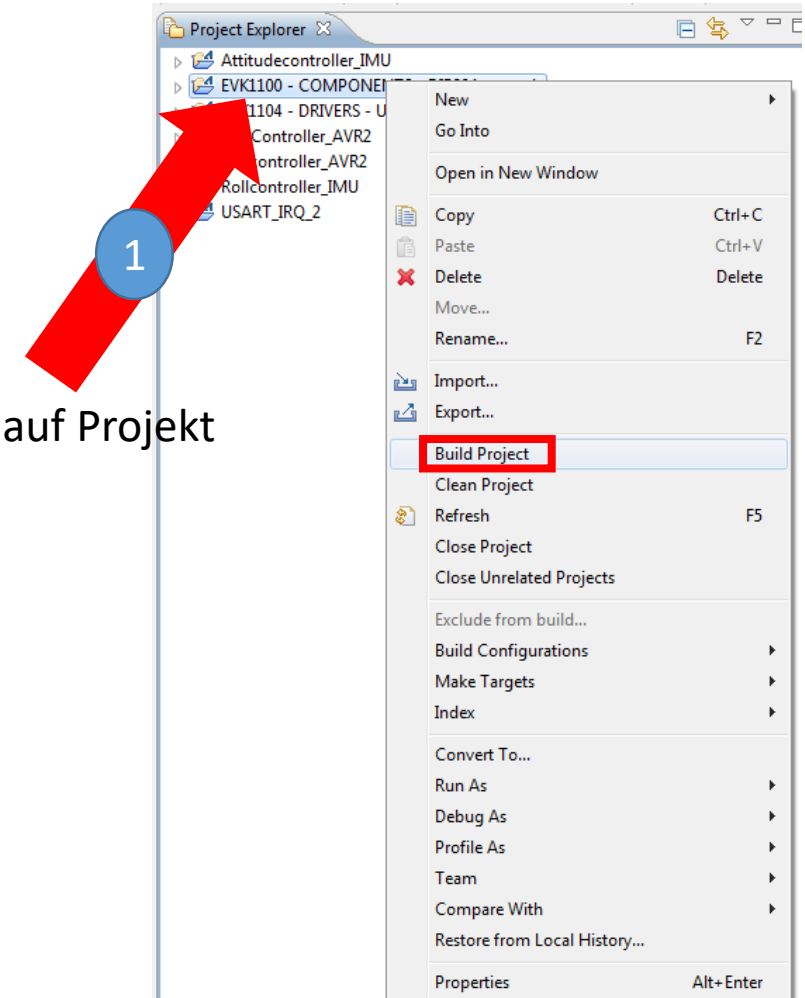
- Schritt 1: Power einschalten:
Board muss Strom haben: Via USB an PC anschließen
und Switch in USB-Stellung (grüne LED an) schalten
- Schritt 2: Code muss korrekt kompiliert werden, wie folgt:
Projekt-Code kompilieren mittels „Build Project“ oder „Strg + B“
Nach dem Kompilieren stets auf Erfolg überprüfen!

Rechte-Maustaste auf Projekt

Erfolgreiches Kompilieren überprüfen



```
C-Build [EVK1100 - COMPONENTS - DIP204 example]
src\SOFTWARE_FRAMEWORK\DRIVERS\PWM\pwm.o src\SOFTWARE_FRAMEWORK\DRIVERS\PM\power_clocks_lib.o
src\SOFTWARE_FRAMEWORK\DRIVERS\PM\pm_conf_clocks.o src\SOFTWARE_FRAMEWORK\DRIVERS\PM\pm.o
src\SOFTWARE_FRAMEWORK\DRIVERS\INTC\intc.o src\SOFTWARE_FRAMEWORK\DRIVERS\INTC\exception.o
src\SOFTWARE_FRAMEWORK\DRIVERS\GPIO\gpio.o src\SOFTWARE_FRAMEWORK\DRIVERS\FLASHC\flashc.o
src\SOFTWARE_FRAMEWORK\COMPONENTS\DISPLAY\DIP204\dip204.o src\SOFTWARE_FRAMEWORK\BOARDS\EVK1100\led.o
src\SOFTWARE_FRAMEWORK\ASM\trampoline.o -lnewlib addons-at32ucr2-speed_opt
Build complete for project EVK1100 - COMPONENTS - DIP204 example
Time consumed: 10129 ms.
```



Grundlagen zum Programmieren

Flashen / Programmieren (4 Schritte: 3.)

- Schritt 3: Flash-Modus: Board zum Flashen vorbereiten, d.h. drücke BT1 Button + (R)esetbutton wie folgt: BT1 gedrückt halten, dann R drücken, R loslassen, dann BT1 loslassen

Wenn MCU anhält, d.h. Display und LED1 bis LED6 ausgeht, dann war ist die MCU erfolgreich im Flash-Modus. Wenn dies nicht so ist, Schritt 3 wiederholen.

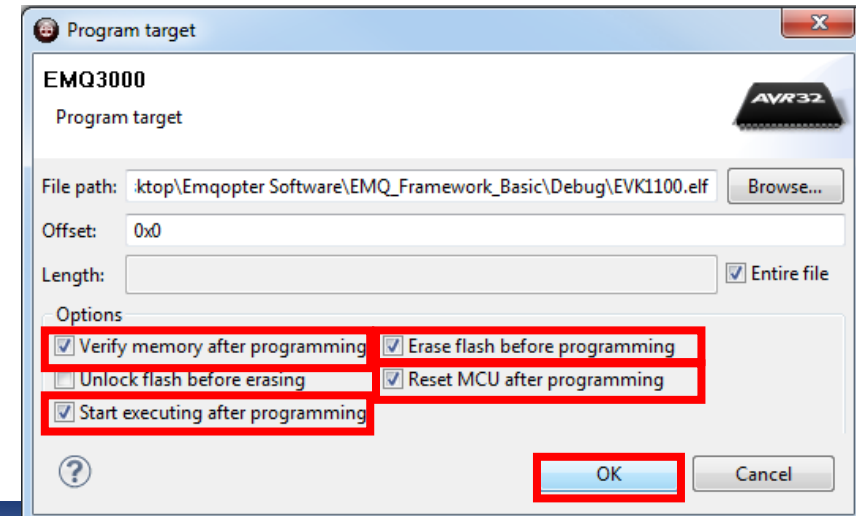
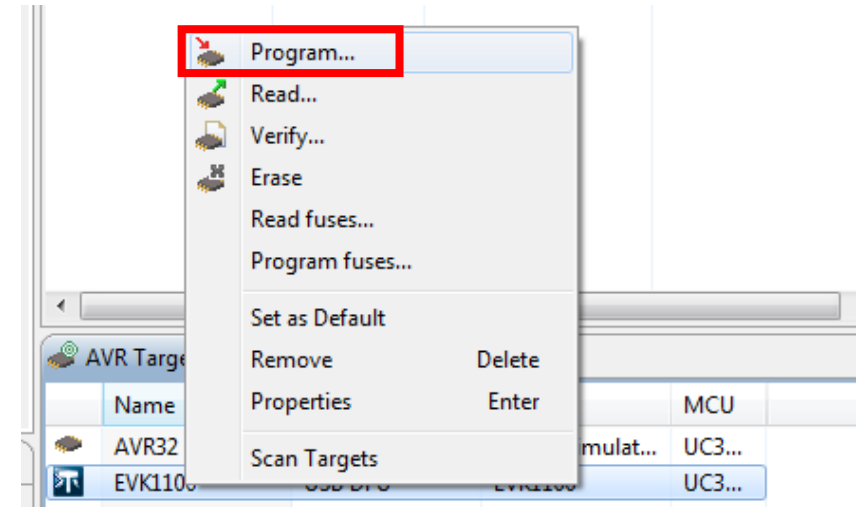


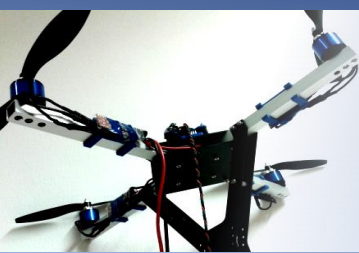
Grundlagen zum Programmieren

Flashen / Programmieren:

Schritt 4: Das eigentliche Flashen

- Nun wollen wir das Programm in den Flashspeicher schreiben, damit es ausgeführt werden kann
- Rechtsklick aufs Target -> Wähle „Program“
- Binary (*.elf Datei) aus z.B. folgendem Ordner auswählen:
„C:\EMQ_Workspace\EMQ_Framework_Basic\Debug,,
- Haken wie folgt setzen:
 - Verify memory -> Check beim Kopieren
 - Start executing -> Starte anschließend
 - Erase flash -> Lösche altes Programm
 - Reset MCU -> Starte Programm von vorne
- OK drücken





Grundlagen zum Programmieren

Batchisp Error:

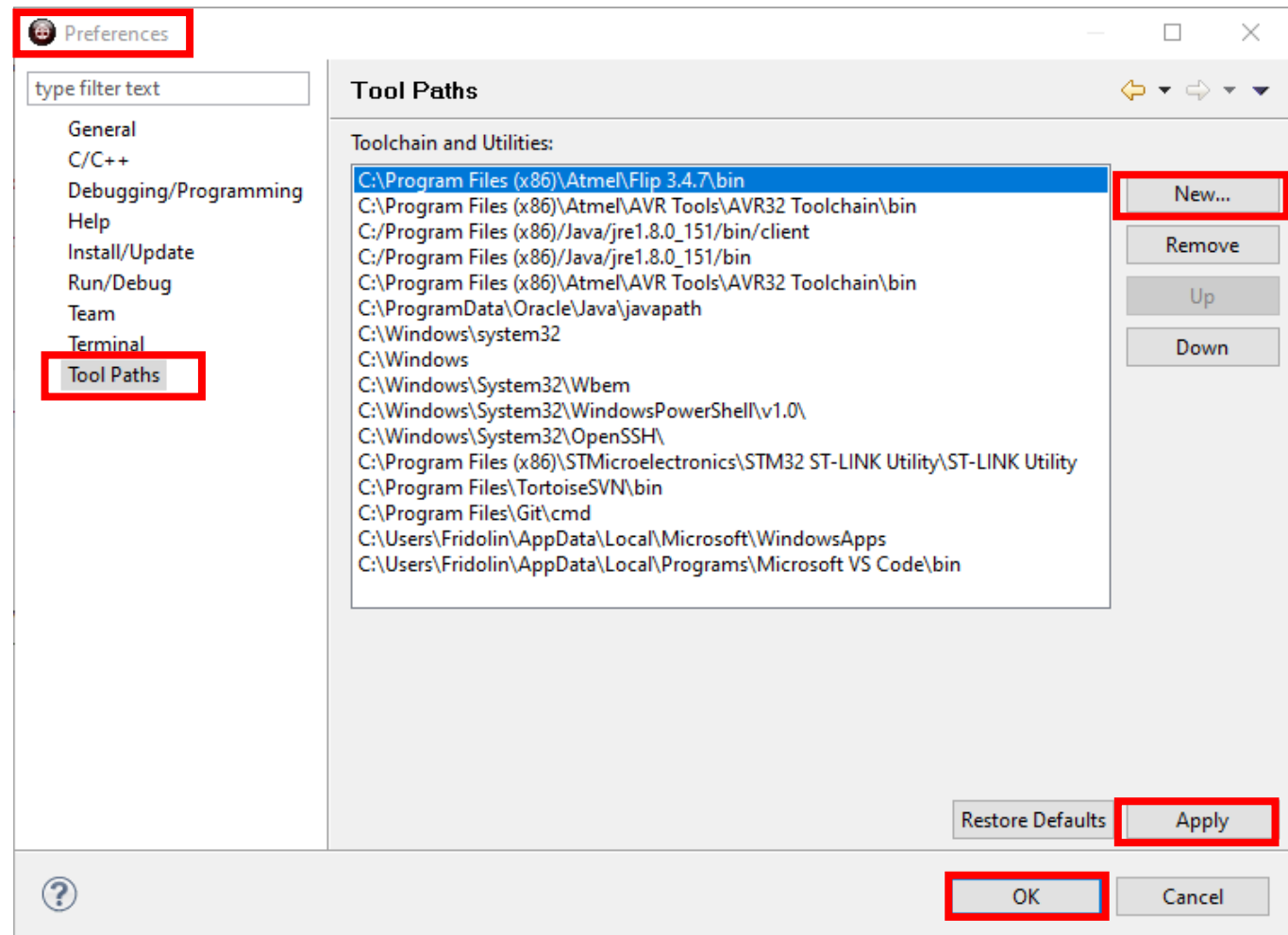
- Der Batchisp-Error ist ein USB-Treiber-Fehler bzw. ein Bug von Windows
- Er kommt einmalig pro Account vor
- Wenn er auftritt und das Flashen fehlschlägt, kann er wie folgt gelöst werden

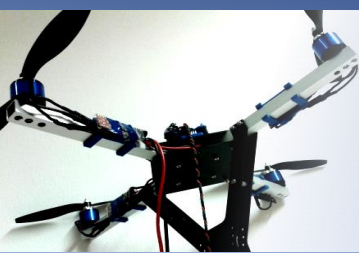
Batchisp Error Lösung:

- Wähle „*Preferences*“ und dort „*Tool Paths*“
- Füge über „*New*“ folgenden Ordner hinzu:
C:\Program Files (x86)\Atmel\Flip 3.4.7\bin

bzw. (je nach Windows Betriebssystem)

C:\Program Files\Atmel\Flip 3.4.7\bin



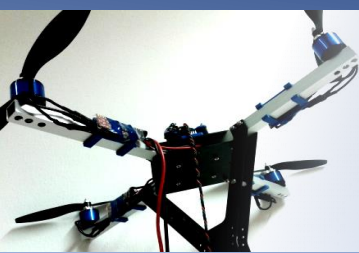


Grundlagen zum Programmieren

Nun zum Hello World - Füge den Text ein:

- Ändere die Anzeige im Display zu Hello World
- Rechts-Klick auf linken Fensterrand
-> „Show-Line Number“ auswählen
- Öffne die Datei my_display.c im Ordner
\\EMQ_Framework_Basic\\src\\Display\\
- Füge in Zeile 19 in der Funktion
my_write_to_display() folgende Zeilen
ein:
snprintf(line_d,20,"Hallo Emqopter");
write_to_display(1,line_d);
- Kompiliere und Flashe den Code:
Wiederhole Schritt 1 bis 4
- Guck dir auf dem Display an,
was passiert ist.

```
15// Exercise Solution Function
16// Function to write to display according to exercises, executed once
17void my_write_to_display(){
18    /*      ----      Add your code here      ----      */
19    snprintf(line_d,20,"Hallo Emqopter");
20    write_to_display(1,line_d);
21}
```



Grundlagen zum Programmieren

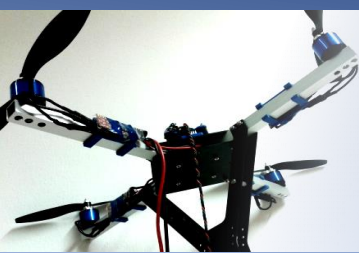
Überblick Projekte

Es stehen 4 Projekte zur Verfügung, die sich im Ausmaß der enthaltenen Funktionen, im Schwierigkeitsgrad und „Lösungen“ unterscheiden. D.h. bei den Frameworks fehlen Module und Funktionen bzw. dort sind dann zu füllende Lückentexte. Auf dieser Grundlage kann der Umfang und Schwierigkeitsgrad beliebig variieren.

- Das Projekt **EMQ_QCSF** enthält die vollständigen Lösungen und ist zunächst NUR für den Betreuer gedacht. Hier können die Lösungen nachgeschlagen werden. Damit kann auch geflogen werden.
- Das Projekt **Framework** enthält Lückentexte, d.h. hier müssen die Lösungen (siehe EMQ_QCSF) von den Teilnehmern eingetragen bzw. erarbeitet werden. Das ist gleichzeitig die Minimal-Fassung.
- Beim Projekt **Framework_Control** fehlt lediglich das Modul Control, das noch zu leisten wäre. Dies ist eine Aufgabe für ein paar Stunden, ideal für Projektarbeit, Praktika oder eine Summer School. Die übrigen Funktionen / Lösungen sind bereits enthalten.
- Das Projekt **Framework_Basic** entspricht Framework mit ein paar (wenigen) Vereinfachungen.

+ Modul mit Lösung
- Modul mit Lückentext
0 Modul nicht enthalten

Projekt \ Modul	COM	Automation	Control	Display	IMU	Quaternion	Kalman	Sensor
EMQ_QCSF	+	+	+	+	+	+	+	+
Framework	-	0	-	-	-	0	-	0
Framework_Control	+	0	-	+	+	+	0	0
Framework_Basic	-	0	-	-	-	0	0	0



Grundlagen zum Programmieren

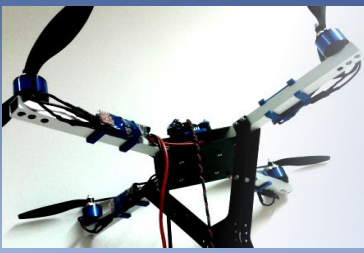
Treiber hinzufügen:

- Wähle: Framework -> Select Drivers / Components
- Klicke 4x „Next“
- Klicke „Finish“



Hinweis:

Bei Compiler-Fehlern in Zusammenhang mit *newlib* sollten die Treibereinstellungen überprüft werden.



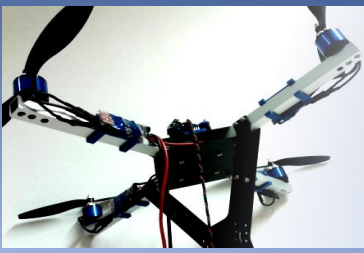
Grundlagen zum Programmieren

EMQ Framework Basic - Code erklärt:

Files / Module:

• COM	->	Kommunikation	Aufgabenteil / Modul
• Control	->	Regelung	Aufgabenteil / Modul
• Display	->	Display (Anzeige)	Aufgabenteil / Modul
• IMU	->	Sensorik / IMU	Aufgabenteil / Modul
• KalmanFilter	->	Kalman Filter	Aufgabenteil / Modul
• <u>Libs</u>	->	Library / Interfaces	
• Quaternion	->	Quaternionen	Aufgabenteil / Modul
• <u>basics_qcs.h</u>	->	Grundeinstellungen	
• QCSF_API	->	Anwendungsverbindungsebene, verbindet Hauptfunktionen mit Treibern	
• QCSF_MAIN	->	Hauptprogramm, oberste Ebene	

Überblick - Framework



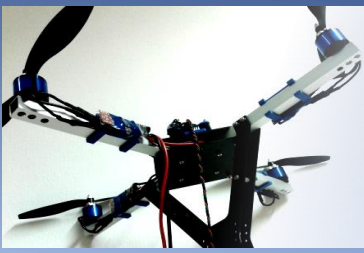
Grundlagen zum Programmieren

EMQ_QCSF- Code erklärt:

Files / Module:

• COM	->	Kommunikation	Aufgabenteil / Modul
• Automation	->	Automatisierung	Aufgabenteil / Modul
• Control	->	Regelung	Aufgabenteil / Modul
• Display	->	Display (Anzeige)	Aufgabenteil / Modul
• IMU	->	Sensorik / IMU	Aufgabenteil / Modul
• KalmanFilter	->	Kalman Filter	Aufgabenteil / Modul
• <u>Libs</u>	->	Library / Interfaces	
• Quaternion	->	Quaternionen	Aufgabenteil / Modul
• SensorsActorsMisc	->	Diverse Sensoren & Add-Ons	
• SituationAwareness	->	OD Modul (Hinderniserkennung)	
• <u>basics_qcsf.h</u>	->	Grundeinstellungen zum Fliegen	
• QCSF_API	->	Anwendungsverbindungsebene, verbindet Hauptfunktionen mit Treibern	
• QCSF_API_PRO	->	Anwendungsverbindungsebene mit Zusatzfunktionen zu Add-Ons	
• QCSF_MAIN	->	Hauptprogramm, oberste Ebene	

Überblick - Framework

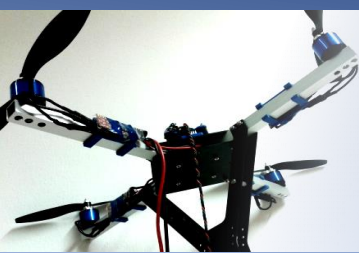


Grundlagen zum Programmieren

EMQ Framework Basic

Libs / Library / Bibliotheksfunktionen:

- Befinden sich im Libs Ordner
- Fertige Funktionen und Vorlagen
- Datentypen (EMQ_Interface_Data):
Beinhaltet die Datentypen der Lösung bzw. unsere Empfehlung.
Wenn die Datentypen in Zusammenhang mit Funktionen der Library eingesetzt werden, dürfen die Datentypen nicht geändert werden.
- Funktionen (EMQ_Interface.h):
In Library fest implementierte Funktionen, die direkt wie gegeben genutzt werden können.
- libemq.a ist die von uns mitgelieferte, bereits kompilierte Library, wo diese Funktionen implementiert sind.



Grundlagen zum Programmieren

basics_qcs.h / Grundeinstellungen:

- Die Motoren lassen sich mit ENGINE_ACTRL_ON aktivieren
-> Standardmäßig deaktiviert aus Sicherheitsgründen



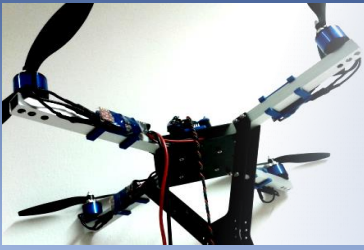
Ausprobieren:

- 1.) Einkommentieren
- 2.) Clean + Build
- 3.) Flashen
- 4.) PB0 drücken

- 5.) Auskommentieren
- 6.) Clean + Build
- 7.) Flashen
- 8.) PB0 drücken

- Weitere Einstellungen
 - Hier stehen Modell und die SW-Version
 - Das LED-PIN-Out hängt vom Board ab:
 - Kein FLUG = EVK
 - FLUG = MCU onBoard
 - Für uns hier: Irrelevant

EMQ Framework Basic

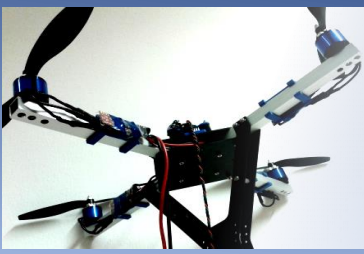


Grundlagen zum Programmieren

Exkurs - Präprozessor Anweisungen:

- Kompiler-Anweisungen, die vor dem Kompilieren ausgeführt werden
- Diese werden verwendet, um Konfigurationen / Parameter einzustellen (Programmeinstellungen, s.o.)
- `#define KENNUNG` Definiert eine Kennung, die z.B. wie folgt abgefragt werden kann:
 `#ifdef KENNUNG`

 `#endif`
 Dann wird der Code (...) nur ausgeführt, wenn die Kennung definiert ist.
- `#define BEZEICHNER ERSATZ` Ersetzt den BEZEICHNER überall durch ERSATZ
 `#define PI 3.14159`
 `double radius = 1;`
 `double umfang = 2 * PI * radius;`
 `double flaeche = PI * radius * radius;`



Die Software Bibliothek

EMQ_Interface_Data

Enthält Datentypen in EMQ_Interface_Data, für:

- PID-Parameter: P, I, D
- Vektoren : X, Y, Z
- IMU-Daten: RPY, RPY_RATE, Quaternion
- IMU-Rohdaten: Vektoren für Gyrometer und Accelerometer
- Motorstellwerte: 4 Bytes für die 4 Motoren
- steerData Datentyp (Steuerdaten) für die Steuerung:
 - engine_on: Aktiviert die Motoren
 - calibrate_on: Gibt an, dass erfolgreich kalibriert wurde
 - flying: Gibt an, dass der Qopter fliegt
 - blctrl_init: Initialisiert die Brushless Controller, nachdem die Motoren aus waren

```
14// PID Parameter
15typedef struct {
16    double p;
17    double i;
18    double d;
19} pid_parameter;
20
21// 3D Vector
22typedef struct{
23    double x;
24    double y;
25    double z;
26} vector;
27
28// IMU Data: RPY Angle, RPY Angular Rate, Qt
29typedef struct {
30    vector rpy;
31    vector rpy_angular_rate;
32
33    double q0;
34    double q1;
35    double q2;
36    double q3;
37} imu_data;
38
39// IMU Raw Data
40typedef struct {
41    vector gyro;
42    vector acc;
43} imu_data_raw;
44
45// Motor Values
46typedef struct {
47    unsigned char M1;
48    unsigned char M2;
49    unsigned char M3;
50    unsigned char M4;
51} motor_data;
```

Die Software Bibliothek

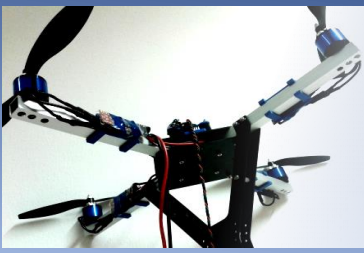
EMQ_Interface_Data



Dx6E

- Fernsteuerung (RC, remote control), 6 Kanäle, 7 Werte

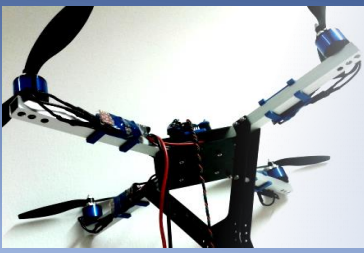
Remote Data	Remote Data Raw (Rohwert)	Erklärung
gas	throttle	Gas-Stick (linker Stick in Richtung oben + unten)
soll_angle_roll	roll	Roll-Stick (rechter Stick in Richtung links + rechts)
soll_angle_pitch	pitch	Pitch-Stick (rechter Stick in Richtung oben + unten)
soll_angle_yaw	yaw	Yaw-Stick (integrativ) (linker Stick in Richtung links + rechts)
angle_yaw	yaw	Yaw-Stick (linker Stick in Richtung links + rechts)
active	-	0, wenn keine aktive Verbindung (z.B. kein Timeout), sonst 1
calibrated	-	1, wenn RC erfolgreich kalibriert, sonst 0
power_sw	gear_switch	Power Switch / Power Schalter -> Startet Motoren (A Switch)
height_sw	gear_switch	Höhen Switch für Höhenregelung, wenn vorhanden (H Switch)
automation_state	flightmode_switch	Zusatzfunktion (B Switch)
automation_state2	flightmode_switch	Zusatzfunktion (G Switch)



Die Software Bibliothek

EMQ_Interface_Data

- Regelungsdaten (Control Data):
Sollwerte für Roll, Pitch und Yaw sowie Motorgaswert
- TWI Packetframe für die Kommunikation mittels TWI / I²C
Der Standard sieht vor, zunächst die TWI Adresse, dann das Kommando und dann die Daten auf den Bus zu legen.
Der Framework erwartet die 7bit TWI Adresse und ergänzt selbst das Lese- bzw. Schreibbit.
chip: TWI Slave Adresse (7bit)
addr: Kommando (meist Registeradresse)
addr_length: Länge des Kommandos in Bytes (meist 1)
buffer: Pointer auf die Daten: Lesen / Schreiben
length: Datenlänge: Anzahl an Bytes, die gelesen oder geschrieben werden
- Datenstrom dataStream
data: Pointer auf die Daten
size: Anzahl an Daten in Bytes
- ggaData
Daten vom GPS-Receiver (Addon): Zeit, Position (3D), Qualität, Anzahl Satelliten

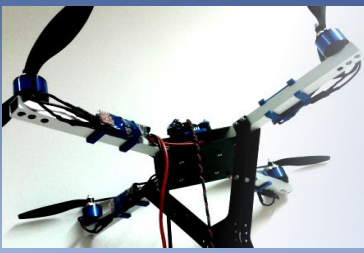


Die Software Bibliothek

EMQ_Interface

Enthält fertige Funktionen, die das Leben einfacher machen!

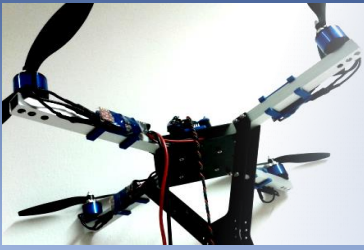
- **Hardware_Init()**
Initialisiert die Hardware (CPU, IRQ, TC, RC, ADC, USART, TWI, Delay) nach Standardeinstellungen
- **Motors_Init()**
Initialisiert die Motoren-Kommunikation
- **Motors_run(motor_data *m)**
Steuert die Motoren an (4 Stellwerte als Parameter)
- **Motors_Start()**
Startet die Motoren, d.h. gibt Sie frei -> Setzt Freigabeflag (Sicherheit)
- **Motors_Stop()**
Stoppt die Motoren, d.h. entfernt Freigabe -> Löscht Freigabeflag, setzt Initialisierungsflag (ESCs)
- **Wait(int miliseconds)**
Warte eine Anzahl an Millisekunden, wirkt blockierend!, auf keinen Fall im Flug verwenden -> sonst Absturz
- **USART_write(char* line)**
Verschickt einen String via serieller USART-Schnittstelle. Mit Sorgfalt und Bedacht verwenden!



Die Software Bibliothek

EMQ_Interface

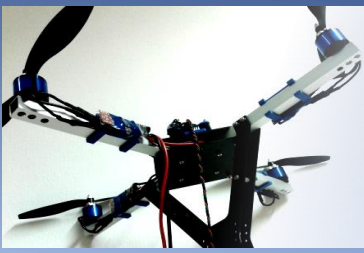
- `Remote_calibrate()`
Kalibriert die RC Fernsteuerung
- `Remote_run(remoteData *remote_data)`
Liest neue Fernsteuerungsdaten aus dem RC DMA aus und schreibt diese in `remote_data`.
- `Remote_get_raw_data(remoteData_raw *remote_raw);`
Als Nebenprodukt generiert `Remote_run()` auch neue Rodaten der Fernsteuerungsbefehle. Diese können, wenn gewünscht, mit dieser Funktion abgerufen werden.
- `ADC_run()`
Führt die Analog-Digital-Convertierung durch. Dies ist erforderlich, um neue Daten für solche Module zu generieren, die über ADC angeschlossen sind. Das sind aktuell die Addons: IR Sensoren, Spannungsmessung
- `Jitter_Check()`
Überprüft das Timing der Hauptschleife und gibt (einmalig) eine Fehlermeldung aus.
- `Filtering_Init()`
Initialisiert die Filter: Mittelwert + Ausreißerfilter
- `Steer_Init(steerData s)`
Initialisiert die Steuerdaten (`steerData`), erforderlich vor dem Fliegen (vgl. Code)



Die Software Bibliothek

EMQ_Interface

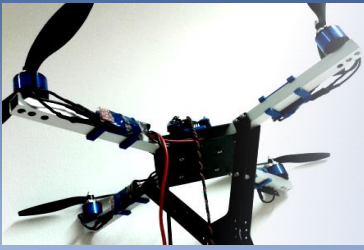
- LED_Control(char led, char command)
Steuert die LEDs: Der erste Parameter *led* gibt die Kennung an, der zweite Parameter *command* gibt den Befehl an.
- Push_Button_1_Pressed()
Gibt den Wert „1“ zurück, wenn seit dem letzten Funktions-Aufruf auf den Push Button 1 gedrückt wurde, sonst den Wert „0“.
- Push_Button_2_Pressed()
Analoge Funktion wie Push_Button_1_Pressed() zu Button 2.
- set_pin(unsigned int pin)
Setzt einen GPIO PIN auf High (1). Mit Bedacht verwenden!
- clr_pin(unsigned int pin)
Setzt einen GPIO PIN auf Low (0). Mit Bedacht verwenden!



Die Software Bibliothek

EMQ_Interface

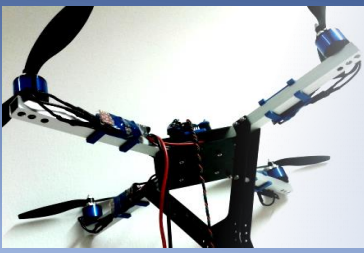
- `sign(double a)`
Gibt das Vorzeichen zurück: 1 oder -1
- `minimum(double a, double b)`
Gibt das Minimum zweier Zahlen zurück
- `saturate(double x, double grenze);`
Saturiert (limitiert betraglich) eine Zahl x , so dass $|x| < grenze$
- `read_from_twi(const twi_package_t *twi, int error_id)`
Liest Daten via TWI-Interface. Error_ID ist ein frei wählbarer Parameter, um Fehler zu erkennen.
- `write_to_twi(const twi_package_t *twi)`
Schreibt Daten via TWI-Interface.
- `write_to_display(char dip_index, char* line)`
Schreibt eine Zeile aufs Display in die Spalte `dip_index`. Maximal 20 Zeichen (pro Zeile) sind erlaubt.
- `Display_Init()`
Initialisiert das Display: Vor der Verwendung des Displays muss diese Funktion ausgeführt werden.
- `Start_Button_Init()`
Initialisiert die Push Button Interrupts (PB0, PB1, PB2)



Die Software Bibliothek

EMQ_Interface

- `USART0_schreiben_double_6(double value1, double value2, double value3, double value4, double value5, double value6)` Sendet 6 Fließkommazahlen an die serielle Schnittstelle (USART 0). Dient dem Debugging. Mit Sorgfalt verwenden!
- `USART0_schreiben_double_6_hs(...)`
Analoge Funktion wie `sendDebugValues()`. Die Ausgabe erfolgt jedoch in hoher Auflösung, d.h. mehr mit mehr Nachkommastellen.
- `usart_read()`
Liest einen Datenstrom von der seriellen Schnittstelle (USART 0) aus. Die Daten liegen dabei im Puffer des DMA und können von da direkt verarbeitet werden. Nachdem die Daten verarbeitet wurden, **müssen** diese mit der Funktion `usart_release()` freigegeben werden, damit der Speicherbereich vom DMA für neue Daten genutzt werden kann. Andernfalls kommt es zu Problemen! Dieses Verfahren ist effizienter, als ein vorheriges Kopieren.
- `usart_release()`
Gibt den zuvor gelesenen Speicherbereich des DMAs wieder frei. Zunächst ist mit `usart_read()` ein Datenstrom zu lesen und nach Wunsch zu verarbeiten, anschließend muss der Speicherbereich mit `usart_release()` freigegeben werden. Wird der Speicher nicht freigegeben, läuft der Puffer des DMAs voll und es kommt zu Problemen, die es zu vermeiden gilt.



Die Software Bibliothek

EMQ_Interface

Benötigt neue Library

Weitere USART-Funktionen (Analog mit USART 3):

- USART1_init(unsigned int usart_baudrate);
- USART1_schreiben(char* line);
- USART1_schreiben_binary(char* data,int len);
- USART1_schreiben_char(char x);
- usart1_read();
- usart1_release();

Initialisiere USART mit Baudrate

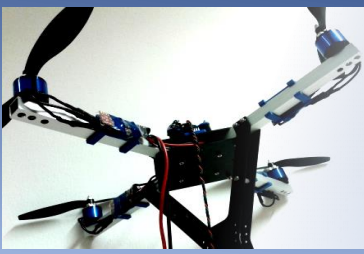
Versende String

Versende String fixer Länge

Sende ein Byte / Charakter

Datenstrom empfangen

Leert Empfangspuffer, nach usart1_read() notwendig



Display und Buttons (DIP)

Display:

- Dogm204
- Wird via SPI angesteuert
- 20 Spalten x 4 Zeilen (80 Zeichen)
- Anzeige von Informationen und Werten (Variablen)
- Beispiel-Code:

```
char line[20];  
int wert = 3;  
sprintf(line, „Wert ist :%i “, wert);  
write_to_display(1,line);
```
- Sprintf Parameter wie beim printf:
Siehe Tabelle

%	Bedeutung	Beispiel
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65

Gute (sichere) Alternative:

snprintf(line, 20, „Wert ist :%i “, wert);

Maximal 20 Zeichen lang.

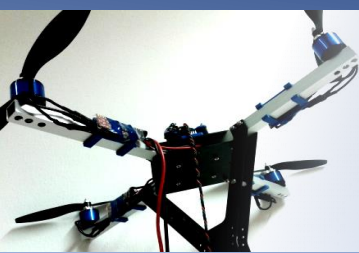


Display und Buttons (DIP)

Buttons

- Das EMQ3000 hat 4 Push-Buttons und 2 Joystickbuttons
- Buttons generieren Interrupts
- Treiber für die Push-Buttons befinden sich in der Library: Push-Button PA07 ist zum Ein-/Ausschalten reserviert. Für die Buttons PB15 und PB16 gibt es Funktionen
- Alternativ kann der Beispiel-Code des AVR Frameworks betrachtet werden, z.B. auch zur Ansteuerung des Joysticks (bisher nicht unterstützt von unserer Library):
DIP 204 wählen: -> AT32UC3A0512 -> EVK 1100 COMPONENTS
DIP204 example
ISR für Joystick lautet: dip204_example_Joy_int_handler()

```
/*!  
 * \brief The joystick interrupt handler.  
 */  
#if __GNUC__  
__attribute__((__interrupt__))  
#elif __ICCAVR32__  
__interrupt  
#endif  
static void dip204_example_Joy_int_handler(void)  
{  
    if (gpio_get_pin_interrupt_flag(GPIO_JOYSTICK_UP))  
    {  
        dip204_set_cursor_position(19,1);  
        dip204_write_data(0xDE);  
        display = 1;  
        /* allow new interrupt : clear the IFR flag */  
        gpio_clear_pin_interrupt_flag(GPIO_JOYSTICK_UP);  
    }  
    if (gpio_get_pin_interrupt_flag(GPIO_JOYSTICK_DOWN))  
    {  
        dip204_set_cursor_position(19,3);  
        dip204_write_data(0xE0);  
        display = 1;  
        /* allow new interrupt : clear the IFR flag */  
        gpio_clear_pin_interrupt_flag(GPIO_JOYSTICK_DOWN);  
    }  
    if (gpio_get_pin_interrupt_flag(GPIO_JOYSTICK_LEFT))  
    {  
        dip204_set_cursor_position(18,2);  
        dip204_write_data(0xE1);  
        display = 1;  
        /* allow new interrupt : clear the IFR flag */  
        gpio_clear_pin_interrupt_flag(GPIO_JOYSTICK_LEFT);  
    }  
    if (gpio_get_pin_interrupt_flag(GPIO_JOYSTICK_RIGHT))  
    {  
        dip204_set_cursor_position(20,2);  
        dip204_write_data(0xDF);  
        display = 1;  
        /* allow new interrupt : clear the IFR flag */  
        gpio_clear_pin_interrupt_flag(GPIO_JOYSTICK_RIGHT);  
    }  
    if (gpio_get_pin_interrupt_flag(GPIO_JOYSTICK_PUSH))  
    {  
        dip204_set_cursor_position(19,1);  
        dip204_write_data(0xDE);  
        display = 1;  
        /* allow new interrupt : clear the IFR flag */  
        gpio_clear_pin_interrupt_flag(GPIO_JOYSTICK_PUSH);  
    }  
}
```



Aufgaben

Hilfe und Hintergrund:

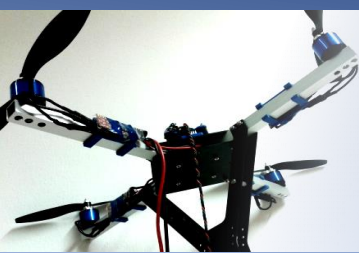
Die Aufgaben sind mit Hilfe des Projekts **Framework_Basic** zu lösen. Fügen Sie Ihre Lösungen im **Ordner Display** in der **Datei my_display.c** hinzu. Dazu dienen die folgenden zwei Funktionen: Die Funktion `my_write_to_display()` wird einmalig aufgerufen, die Funktion `my_renew_display()` wird wiederkehrend (alle 10ms) aufgerufen.

Aufgabe 1:

Als erstes schreiben wir den Text „Hallo Welt“ auf das Display in Zeile 1. Verwenden Sie dazu die Funktionen `write_to_display(char dip_index, char* line)` und `sprintf(...)` oder `snprintf(...)`. Als Hilfe dient Ihnen dazu Folie 39. Fügen Sie den Code in die Funktion `my_write_to_display()` ein.

Aufgabe 2:

Legen Sie eine Variable an und weisen Sie Ihr z.B. den Wert 3 zu. Schreiben Sie nun den Wert dieser Variablen auf das Display. Vergleichen Sie dazu Folie 39.



Aufgaben

Aufgabe 3:

Schalten Sie LED_1 und LED_2 auf dem EMQ3000 Board an. Verwenden Sie dazu die Funktion LED_Control(char led, char command). Als Hilfe dient Ihnen Folie 35.

Aufgabe 4a:

Standardmäßig blinkt LED_4 (LED6 auf dem EMQ3000). Schalten Sie diese aus.

Aufgabe 4b:

Schalten Sie LED_3 (LED5 auf dem EMQ3000) und LED_4 (LED6 auf dem EMQ3000) blinkend. LED_3 soll schnell blinken und LED_4 soll langsam blinken. Verwenden Sie dazu die Funktion LED_Control(char led, char command) und die Hilfe aus Folie 35. Denken Sie daran, dass “blinken” ein wiederkehrender Vorgang ist: an-aus-an-aus oder toggle-toggle-toggle.

Aufgabe 4c:

Was passiert nun, wenn Sie die Lösung aus Aufgabe 4a nun rückgängig machen? Können Sie das Verhalten erklären?



Aufgaben

Aufgabe 5:

Programmieren Sie ein dreiseitiges Display. Auf jeder Seite soll ein anderer Wert angezeigt werden. Mit dem Knopf PB15 soll sich die Seitenzahl ändern lassen. Auf Seite 3 soll Seite 1 folgen. Mit dem Knopf PB16 soll der Wert der aktuellen Seite inkrementiert werden.

Hilfe und Hintergrund:

Mit der Funktion `Push_Button_1_Pressed()` lässt sich abfragen, ob der Knopf PB15 gedrückt wurde. Sie liefert „1“ zurück, wenn der Knopf PB16 seit dem letzten Funktionsaufruf gedrückt wurde. Die Funktion `Push_Button_2_Pressed()` arbeitet analog für Knopf PB16.



FAQ

Typische Fehlermeldungen:

- cannot find -lemq
Die Library libemq.a fehlt und muss in den Ordner src/Libs kopiert werden.
- cannot find -lnewlib_addons-at32ucr2-speed_opt
Die Konfigurationsdaten fehlen oder sind nicht korrekt. Bitte die Lösung auf Folie 21 durchführen.
- Batchisp Error
Ein Fehler mit dem Flip Treiber und Windows User Rights. Bitte die Lösung aus Folie 17 ausführen.