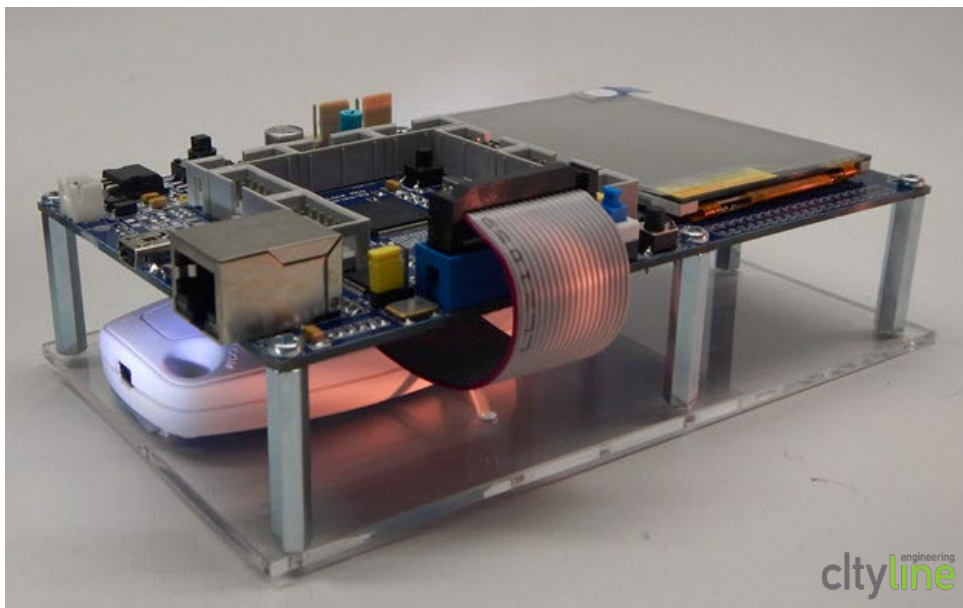


Mikrocontroller MCB32

Erste Schritte



MCB32 - Embedded Programmierung Einstieg

Version: 2107.00 (D)

Bitte beachten. Diese Unterlagen werden ohne Vorankündigung angepasst, verbessert und erweitert. Wünsche und Fehler an: info@mcb32.ch

1 Inhaltsverzeichnis

1	Inhaltsverzeichnis	2
2	Entwicklungsumgebung KEIL aufsetzen	3
2.1	Neues Projekt einrichten mit Keil μ Vision 5	3
2.2	Weitere Projekt- und Prozessor (Target)-Einstellungen	4
3	Erstes Mikrocontrollerprogramm in C auf dem MCB32	9
4	Programmlauf auf dem MCB32 debuggen	10
4.2	Methodik des Debuggens (entlausen)	11
4.3	Programmlauf im PC simulieren	12
5	Umstellung von C51-Code auf ARM32-Code	13
5.1	Wichtig für das Funktionieren neuer Projekte	13
6	ARM Cortex Programmiermethoden	14
6.1	über Registerstrukturen	14
6.2	über CMSIS Funktionen	14
6.3	über P0- und P1-Definitionen (8 Bit-Shield TouchP0P1.lib)	14
7	Programmierkonzepte auf MCB32	15
8	Anhang Touchscreen Kontrolle am μC-Board MCB32	17
9	Anhang Verzögerungen und Laufzeiten im μC-Board MCB32	18
10	Anhang Grafikprogrammierung	19
10.1	Einleitung	19
10.2	Hardwarenahe Beschreibung der Displayansteuerung	19
10.3	Grafikprogrammierung	20
10.3.1	Fonts	20
10.3.2	Grafikfunktionen [3]	20
10.4	Musterprogramm für Grafikfunktionen	23
11	Anhang Anschlüsse am μC-Board MCB32	24
11.1	Stecker Belegung am Beispiel Port A	24
11.2	Entwicklungsanschlüsse	25
11.3	LED auf Board aktivieren	27
12	Anhang: Referenzen	28

2 Entwicklungsumgebung KEIL aufsetzen

Wir arbeiten mit der IDE der Firma Keil. Damit können Programme bis zu 32kB Programmcode geschrieben werden. Das genügt für die ersten Schritte während der Ausbildung. →Link [1]



Keil uVision5

IDE: Integrated Development Environment von

<http://www2.keil.com/mdk5/install>

ARM: Advanced RISC Machines

2.1 Neues Projekt einrichten mit Keil µVision 5

1. Schritt

µVision 5 starten:

Neues Projekt erstellen:

Start / Programme / Elektronik / KeilArm µVision 5

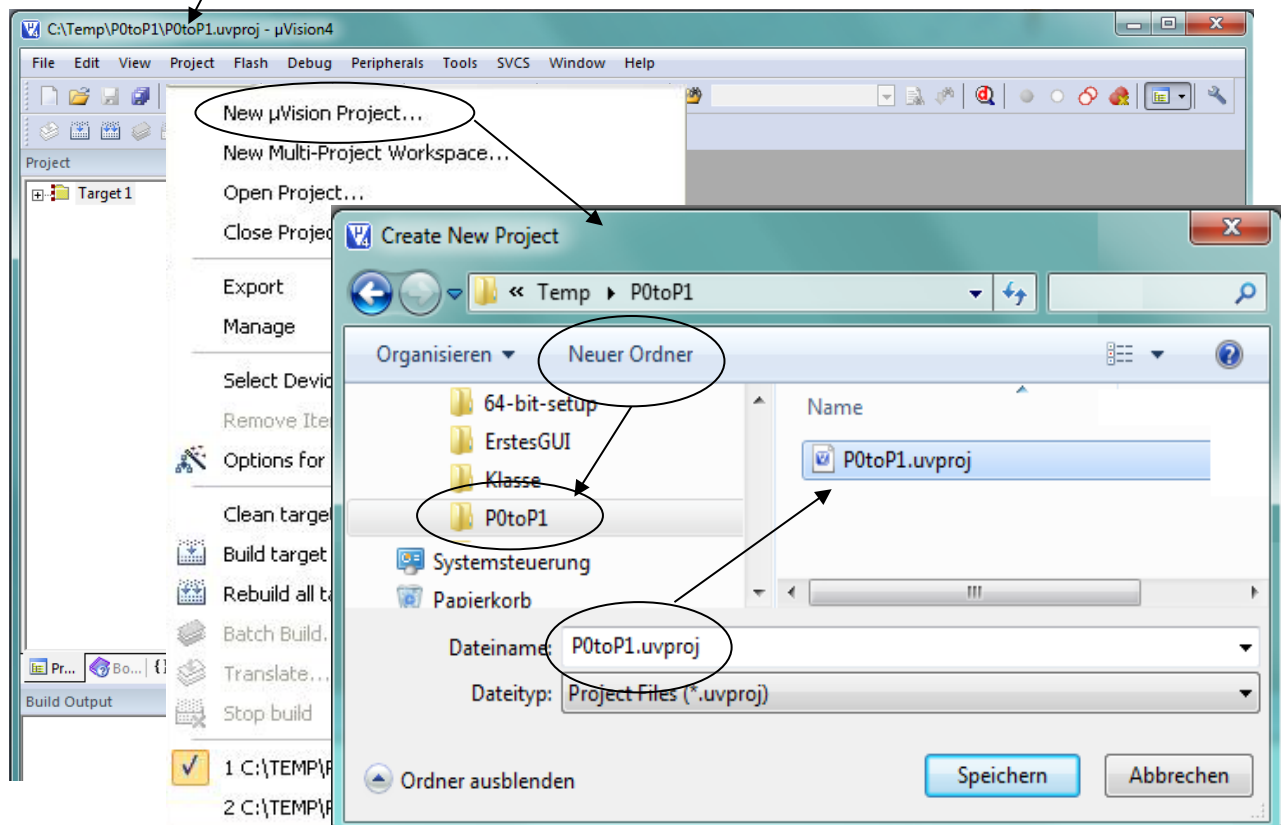
Project / New Project /

Für jedes Projekt

- einen neuen Ordner zB: P0ToP1

- dann Projektname zB: P0ToP1.uvproj

- Speichern



Weitere Menüpunkte der IDE welche es zu beachten gibt.

Altes Projekt öffnen: [Project](#) / Open Project

Projekt schliessen: [Project](#) / Close Project

µVision 4/5 verlassen: [File](#) / Exit

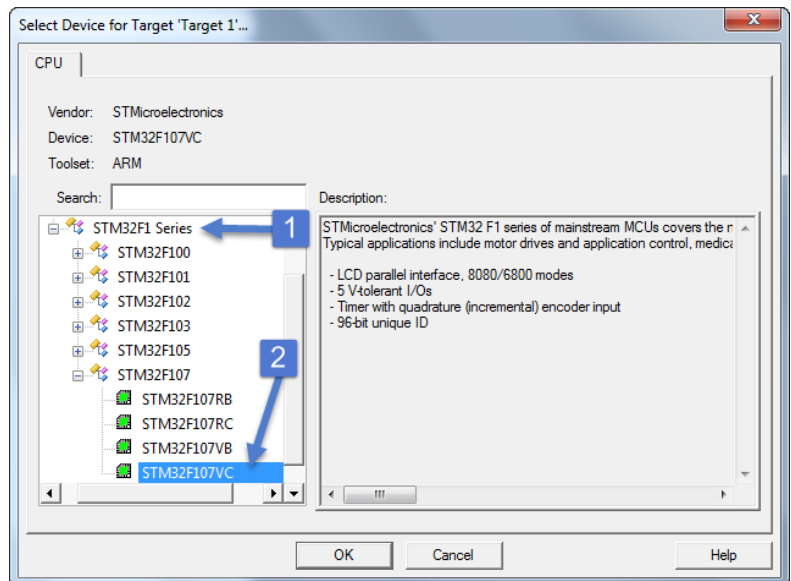
Kopiere: die beiden Files: **TouchP0P1.lib** und **TouchP0P1.h** in das Projekt-Directory. Beide Files müssen vorhanden sein. Kopiere diese Files vom Server (LIB_C oder LIB_D) oder von einem bekannten Ort ins Projekt.

2.2 Weitere Projekt- und Prozessor (Target)-Einstellungen

2. Schritt: Target auswählen

Nun fragt die IDE nach dem Ziel-Controller:

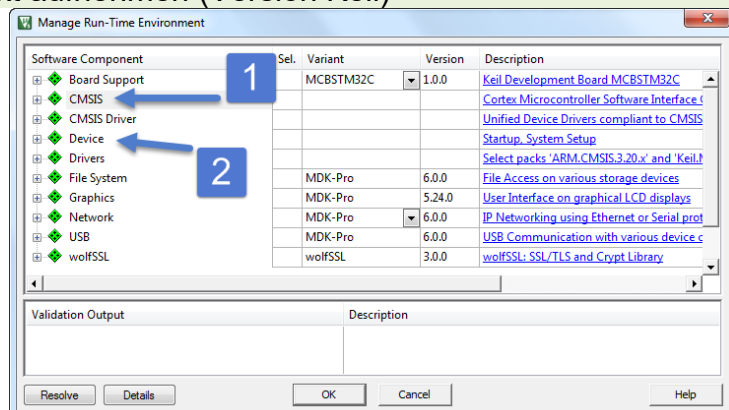
- Ziel-Controller wählen. In unserem Fall der **STM32F107VC**



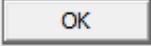
3. Schritt: Startup Code ins Projekt aufnehmen (Version Keil)

Nun benötigt die IDE mehr Informationen und auch „Files“ um nachher während der Kompilation alle Verknüpfungen herstellen zu können:

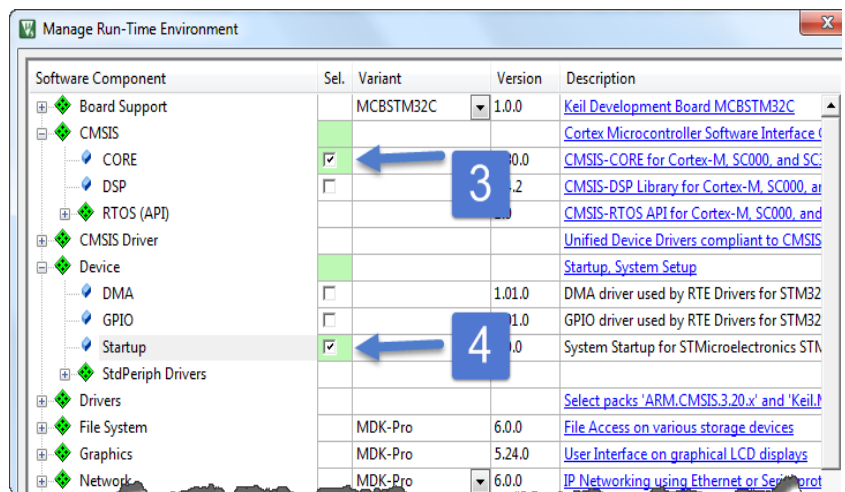
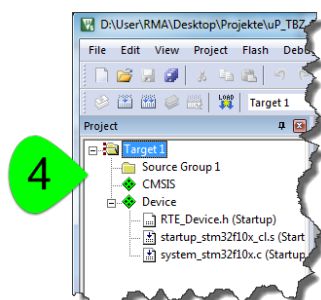
- CMSIS – Basics laden
- Device – Files laden



- CMSIS: Core auswählen [3]
- Device – Startup auswählen [4]

Und  klicken.

Die Projektumgebung sollte nun so aussehen:

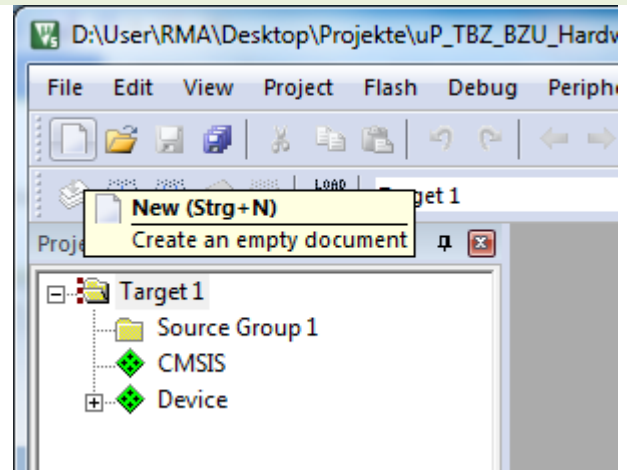


Im Prinzip ist das Projekt bezüglich der grundlegenden Einstellungen bereit. Wir müssen nun im nächsten Schritt die eigentlichen Programmfiles und die Library dazu fügen.

Aufsetzen eines ersten Programmes

4. Schritt: Programm Files aufnehmen.

- Klicke auf NEW (oder STRG-N) und speichere das File mit dem Namen P0toP1.c ab.



5. Schritt: Programm schreiben.

- Schreibe nun folgenden Code.
- Speichere alles ab.



Dieses File können wir mit anderen Files nun noch dem Projekt hinzufügen.

```

1 //-----
2 // MCB32 (BZU / TBZ)
3 // Autor: P0toP1.c / 2.1.14 / rw, mal / BZU,TBZ
4 // Thema: Schaltet am uC-Board MCB32 die Schalter an P0
5 // (GPIOC0..7) zu den LEDs an P1 (GPIOE8..15) durch. Mit Touchbedienung
6 //-----
7 #include <stm32f10x.h> // Mikrokontrollertyp
8 #include "TouchP0P1.h" // Library mit P0-, P1-Definition und Touch
9
10 int main(void) // Hauptprogramm
11 {
12     InitTouchP0P1("1"); // P0,P1 auf Touchscreen ON
13     while(1) // Endlosschleife
14     {
15         P1 = P0; // Portdurchschaltung
16     }
17 }

```

6. Schritt: Alle Programmfiles zum Projekt hinzufügen.

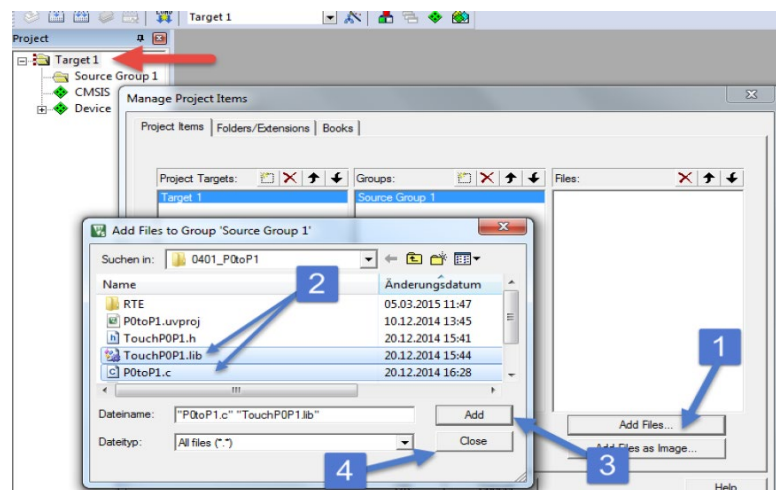
Mit der rechten Maustaste auf „Target1“ klicken und **„Manage Project Items“** auswählen..



- Nun die Schritte 1-4 abarbeiten:

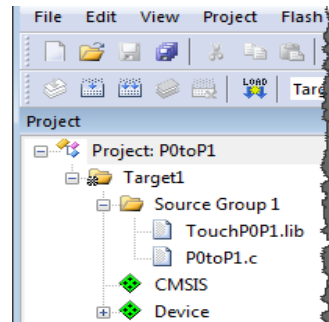
Das File: **P0toP1.c** und die Library **TouchP0P1.lib** werden dem Projekt hinzugefügt.

Wichtig: Die Files können mit CTRL-Click alle zusammen ausgewählt werden.



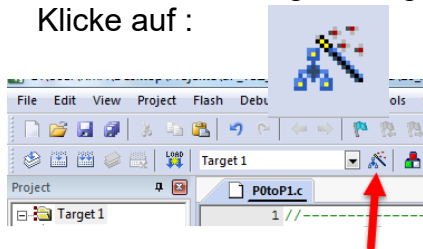
Aufsetzen eines ersten Programmes

So sollte das Projekt nun aussehen:



7. Schritt: Setup vervollständigen (Debugger und JTAG)

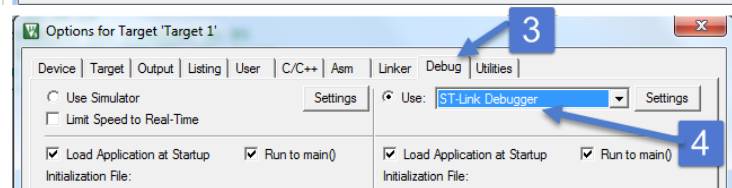
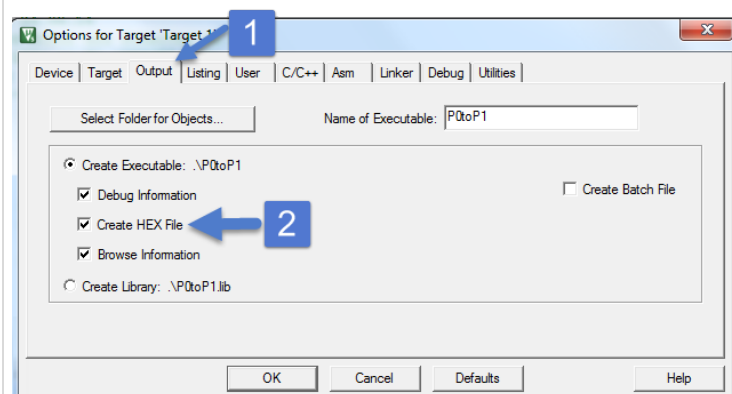
Damit das Programm nach dem Kompilieren in den ARM geladen werden kann sind weitere Einstellungen nötig. Klicke auf :



Damit werden die „Options for Target1“ ausgewählt.

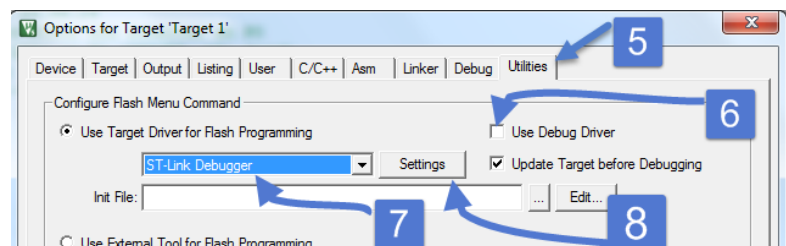
Wähle: [3] „Debug“ und selektiere [4] „STLink Debugger“ aus dem Menü aus.

Wähle: [1] „Output“ und selektiere [2] „Create HEX File“.



Wähle: „Utilities“, [5] Klicke „Use Debug Driver“ aus [6] und selektiere dann „STLink Debugger“. [7]

Am Schluss [8] die Settings auswählen



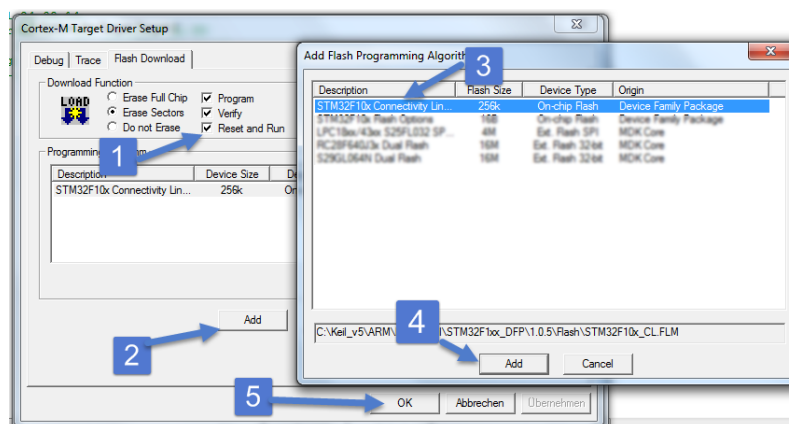
8. Schritt: Settings für ST-Link Download vornehmen

Aufsetzen eines ersten Programmes

Bei den Settings nun „Reset and Run“ auswählen. [1]

Mit „Add“ [2] sagen wir dem Treiber welcher Programmieralgorithmus gewählt wird. Also den für den STM32F10x [3]. Alle anderen Einträge löschen.

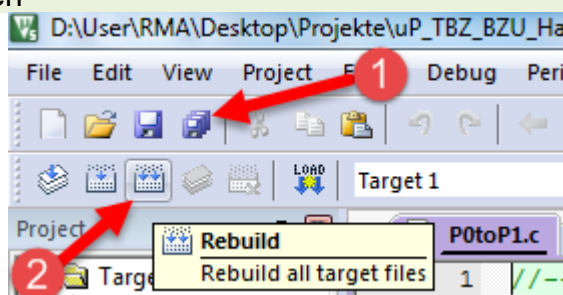
Mit [4] abschliessen und mit OK [5] das Menü „Driver Setup“ schliessen.



Das Fenster Options for Target 'Target 1' schliessen und fertig.

9. Schritt: Programm kompilieren

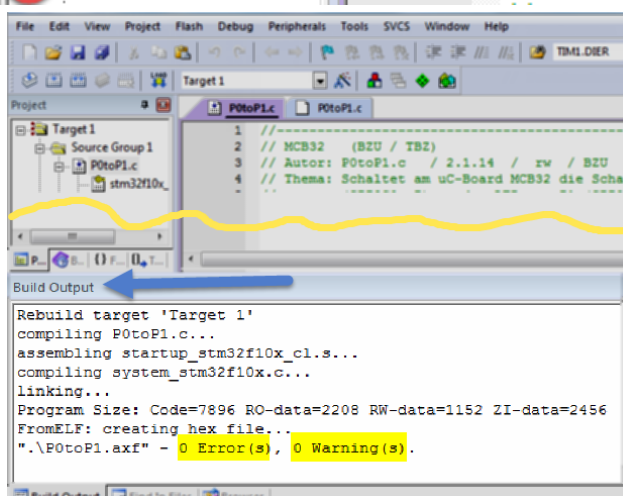
Sichere das Projekt [1] und kompiliere [2] mit dem Knopf „Rebuild all target files“



Im Fenster „Build Output“ sollte in etwa nebenstehende Meldung erscheinen.

Wenn keine Fehler vorkommen kann das erzeugte HEX-File in den Prozessor geladen werden.

FEHLER: Bei Fehlern Doppelklick auf die **erste** Fehlerzeile, diese korrigieren und neu übersetzen.

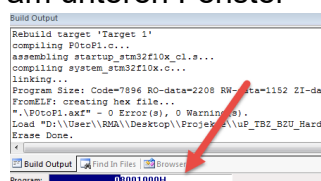
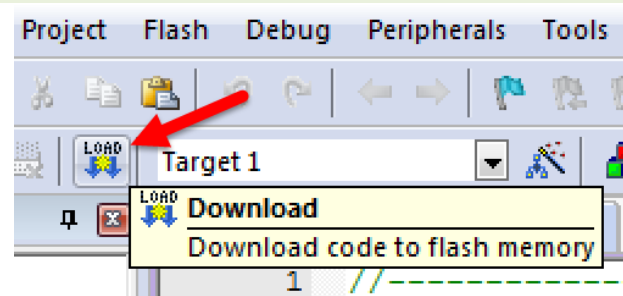


10. Schritt: Programm in den Prozessor laden.

Mit Load wird nun das File in den Kontroller geladen.

Wenn alles funktioniert erscheint ein Balken am unteren Fenster- rand.

Dieser zeigt den Download an.



11. Beachten

1. Das **Potentiometer** ist an Port PC[4] resp. aus Sicht der Library an P0_4 angeschlossen und kann die Funktion dieses I/O-Portes beeinflussen. Bei digitalen I/O-Funktionen mit P0 resp. P0_4 das Potentiometer immer im **Uhrzeigersinn** auf Anschlag drehen.

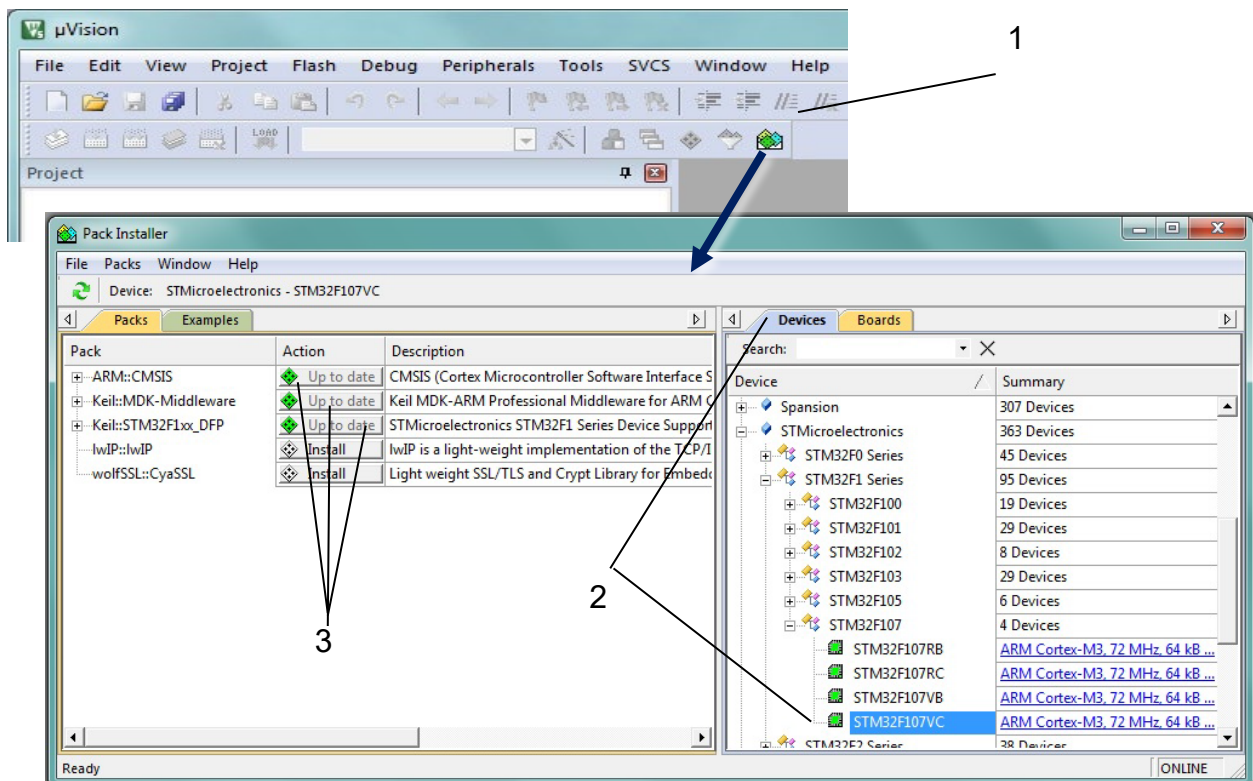


12. KEIL IDE installieren

Die Installation der KEIL IDE ist in einem separaten Dokument beschrieben. [2]

Nach der Installation von μ Vision sind einmalig die μ C-spezifischen Dateien herunterzuladen und einzurichten:

1. Menü / Projekt / Manage / Pack Installer
2. Ziel-Controller wählen / Device:
 - STM32F107VC
3. Pakete installieren /Packs:
 - ARM-CMSIS
 - Keil-MDK
 - Keil-STM32F1xx



3 Erstes Mikrocontrollerprogramm in C auf dem MCB32

```

/*****
Titel:      Port Durchschaltung am MCB32
Datei:      P0toP1.c
Ersteller:   R. Weber / BZU / 14.1.14
Funktion:    Liest die Schalter an P0 ein und gibt diese an Port 1 aus
*****/
#include <stm32f10x.h>           // Mikrokontrollertyp
#include "TouchP0P1.h"          // P0-, P1-Definition

void main (void)               // Hauptprogramm
{
    InitTouchP0P1("1");        // Touchscreen aktiv
    while(1)                   // Endlosschleife
        P1 = P0;               // Portzuweisung von P0 nach P1
}

```

Grundauftrag:

Machen Sie sich mit der Entwicklungsumgebung vertraut, indem Sie obiges Programm von der Eingabe bis zum Test auf dem Zielsystem durcharbeiten.

- Editieren, Speichern
- Compilieren, Simulieren
- Download und Hardwaretest
- Zielsystemdebugging (freiwillig)

Übungsaufgaben:

1. Wie kann ich die Schalter "invertiert" an den Ausgang schalten. Bsp. Wenn der Schalter 0 EIN ist soll die LED 0 nicht leuchten.
Löse es arithmetisch, logisch und mathematisch!
2. Verwende eine Verzögerungsschleife für folgende Ausgaben an die LEDs von Port 1!

```

long t;
for (t = 1200000; t > 0; t - - ) ;           // Verzögerung 100ms (1.2Mio)

```

- ein Blinklicht an Port 1 (alle LED miteinander ein und aus)
- einen Binärzähler an Port 1 (Port 1 zählt von 0 bis ?)
- ein Lauflicht an Port 1 (besondere Problematik?)
- eine Lightshow (Knight Rider) an Port 1

4 Programmlauf auf dem MCB32 debuggen

Periphere Funktionswerte mit Systemviewer

GPIOC

Property

- CRH
- CRH
- IDR
- IDR0
- IDR1
- IDR2
- IDR3
- IDR4
- IDR5
- IDR6
- IDR7
- IDR8
- IDR9
- IDR10
- IDR11
- IDR12
- IDR13
- IDR14
- IDR15
- ODR
- ODR0
- ODR1
- ODR2
- ODR3
- ODR4
- ODR5
- ODR6
- ODR7
- ODR8
- ODR9
- ODR10
- ODR11
- ODR12
- ODR13
- ODR14
- ODR15

IDR [Bits 31..0] RO (@ 0x4001180C) data register (GPIOC_IDR)

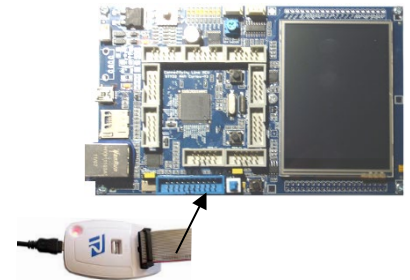
ODR [Bits 31..0] RW (@ 0x4001180C) Port output data register (GPIOC_ODR)

Das Programm wird unter Kontrolle des PC's auf dem Zielsystem ausgeführt und mit SingleStep, FunctionStep, Run, Break, Stop und Variablenbetrachtung verfolgt.

Beachte:

Jeder Programmschritt verursacht grossen USB-Datenverkehr!

- 1) Das Programm läuft bis 1000x langsamer als in Echtzeit
- 2) Während RUN werden keine Bildschirminfos aufgefrischt
- 3) Warteschlaufen immer überspringen!
- 4) Touch P0-, P1-Ein-/Ausgaben werden nur alle 25ms erfasst und angezeigt; also davor und danach je 25ms einbauen!



BREAK1: Touchfeld gedrückt halten, dann RUN
BREAK2: Touchfeld lösen, dann RUN

```
for(t=0; t<500000; t++);
P1 = P0;
for(t=0; t<500000; t++);
```

Tip: Wenn für das Debuggen kein Grafikdisplay benötigt wird, kann mit dem ("0") im Befehl `InitTouchP0P1 ("0")`; der Display abgeschaltet werden.
Keine Unterbrechung durch den Timer-Interrupt des Displaytreibers.

4.2 Methodik des Debuggens (entlausen)

Prinzip: Das Debugging bezweckt das Auffinden von logischen Fehlern, die sich erst bei Laufzeit mit falschem Verhalten bemerkbar machen.

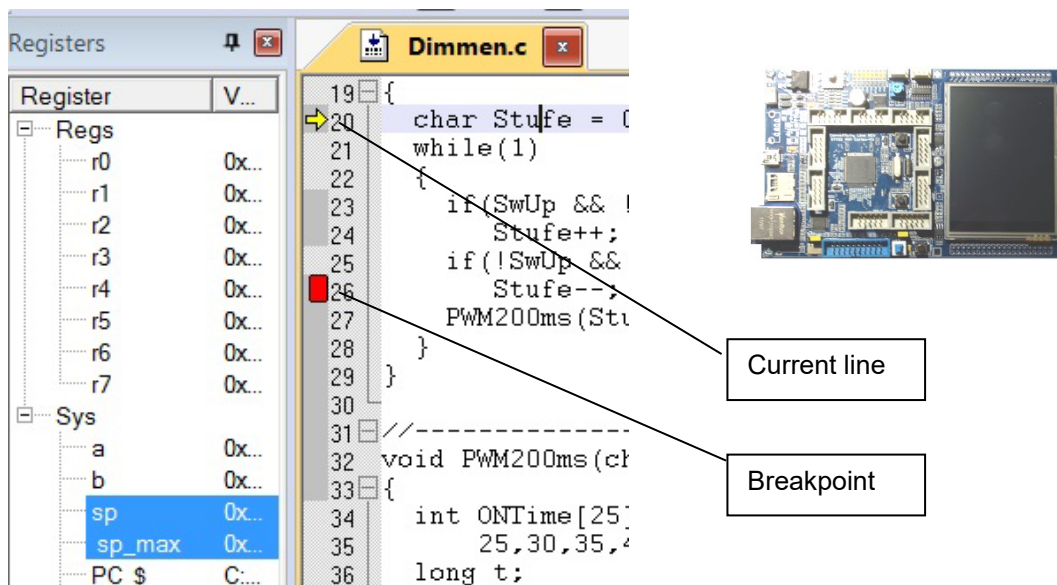
Hochsprachendebugging

verfolgt Daten und Programmablauf im Hochsprachensourcecode auf dem PC. Der uController und seine Peripherie werden auf dem PC simuliert.



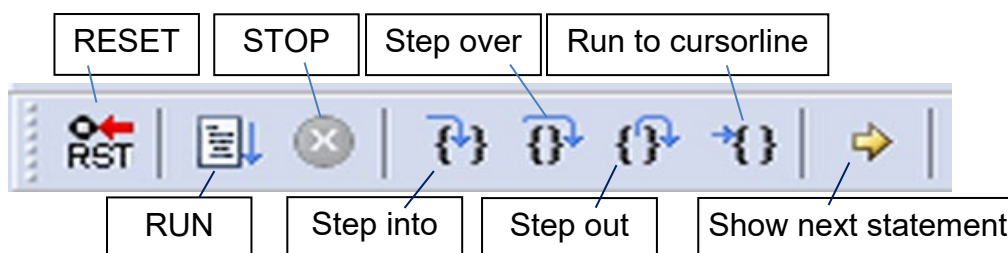
Zielsystemdebugging

Lässt jeden Programmschritt im real angeschlossenen Mikrocontroller ausführen. Der PC steuert und zeigt die realen Daten des Controllers und seiner Peripherie.



a) Verfolgen des Programmlaufs

Werden die **Programmschritte** in richtiger Reihenfolge erreicht und ausgeführt, **Verzweigungen** ausgeführt, **Schleifen** wiederholt und **Unterprogramme** erreicht und ausgeführt?



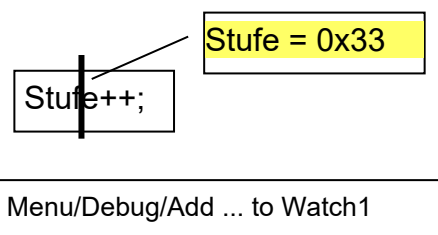
b) Überprüfen der Daten

Im Stepbetrieb am Cursor und im Watchfenster beobachten:

- Variablenwerte: Rechenresultat, Bereichsüberschreitung,
- Überlauf, erreichbare Bereichsgrenzen
- Logische Entscheidung True/False,
- logische Bitverknüpfungswerte 00...FF
- Übergabe- und Rückgabewerte von Unterprogrammen
- Ein-/Ausgabewerte am richtigen Ort mit richtigem Wert.

Im Runbetrieb mit Breakpoints auf den Problemzeilen

- Korrekte Verzweigung,
- Anzahl Wiederholungen,



- Unterprogrammein:

4.3 Programmablauf im PC simulieren

Keil unterstützt **nicht** die Simulation des STM32F107VC!

Für unterstützte Controller siehe: <http://www.keil.com/support/docs/3726.htm>

Mit folgenden Targeteinstellungen können jedoch Programme mit Schaltern und LEDs simuliert werden (ohne InitTouchP0P1" " und keine P0 und P1).

Menü\Projects\Options for Target:

Target\Device:	STM32F103ZE	
Target\Debug\Dialog DLL:	DARMSTM.DLL	(anstelle von DCM.DLL)
Target\Debug\Dialog DLL Parameter:	-pSTM32F103ZE	(anstelle von -pCM3)

5 Umstellung von C51-Code auf ARM32-Code

Folgende Angaben dienen für Kenner der alten Prozessoren, wie sie früher an den Berufsschulen eingesetzt wurden. Sie zeigen die wichtigsten Unterschiede auf.

```

/*****
* Titel:      Umstellung von C51-Code auf ARM32-Code
* Datei:      C51toARM32.c / 12.03.2016 / Version 1.0
* Ersteller:   E. Malacarne (CITYLINE AG) / R. Weber (Uster)
* Funktion:    Die wichtigsten Umstellungen sind in den Kommentaren dokumentiert
*****/

```

```

#include <stm32f10x.h>           // Mikrokontrollertyp
#include " TouchP0P1.h"         // P0-, P1-Definition

#define Start P0_0              // Input und Outputbits
#define Alarm P1_7              // an Ports benennen
char bTemp = 0 ;               // 'Bit'-Variablentyp char
long lt;                       // Zeitvariable

```

- uc-Typ nur bei uVision4
- Neu: **TouchP0P1.h**

Keine sfr
Sbit mehr!

```

int main ( void )              // Hauptprog., ohne return bei Keil
{
    InitTouchP0P1 ("1");       // Touchscreen aktiv,
                                // horizontal gedreht
                                // LSB rechts

    while(1)                   // Endlos-schleife
    {
        P1_0 = 0;              // Bitverarbeitung wie bisher
        Alarm = 1;              // Zuweisung, Invertierung,
        bTemp = ! Start;        // &, &&, |, ||, ^, !, ==, !=

        while ( P1 < 100 )      // Byteverarbeitung wie bisher
            P1 += 2;             // Kurzformen wie bisher
        P1 = P0 & 0x0F;         // Maskierungen wie bisher

        for(lt=120000; lt>0; lt--); // Verzögerung 10ms
    }
}

```

Main verlangt **int**
aber kein return (bei Keil)

InitTouchP0P1 ("0"),
Touchscreen AUS
P0: In an GPIO C Pin7 .. 0
P1: Out an GPIO E Pin15 .. 8

Verzögerung
vom Typ long mit
Wert 12 / μ s

5.1 Wichtig für das Funktionieren neuer Projekte

Wichtig: Bei der Erstellung eines neuen Projektes im Schulbereich, also Vorbereitung für 8Bit-Programme „Elektroniker“ mit Port P0, P1 und Touchscreen ist folgendes zu beachten.

Kopiere in jedes neue Projektverzeichnis diesen zwei Dateien:

- TouchP0P1.h (ab REV D)
- TouchP0P1.lib (ab REV D)

6 ARM Cortex Programmiermethoden

6.1 über Registerstrukturen

Projekt: BlinkReg

```
int main(void)                // Hauptprogramm
{
    long t;
    RCC->APB2ENR |= 1<<6;      // GPIOE Clock
    GPIOE->CRH = 0x33333333;    // PE15..8 Output,
                                // .. PushPull 10MHz
    while(1)                   // Endlosschleife
    {
        GPIOE->ODR = 0x8000;    // GPIOE Bit15 ON
        for(t=1200000; t>0; t--); // Verzög 100ms
        GPIOE->ODR = 0;         // GPIOE Bit15 OFF
        for(t=1200000; t>0; t--); // Verzög 100ms
    }
}
```

6.2 über CMSIS Funktionen

Cortex Microcontroller Software Interface Standard

Projekt: BlinkCMSIS

```
int main(void)
{
    long t;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    SystemInit();
    while (1)
    {
        GPIO_SetBits(GPIOE, GPIO_Pin_15);
        for(t=100000; t>0; t--);
        GPIO_ResetBits(GPIOE, GPIO_Pin_15);
        for(t=100000; t>0; t--);
    }
}
```

6.3 über P0- und P1-Definitionen (8 Bit-Shield TouchP0P1.lib)

```
#include "TouchP0P1.h"        // P0-,P1-,Touchscreen
int main(void)                // Hauptprogramm
{
    long t;
    InitTouchP0P1("1");       // P0P1-Touchscreen ON
    while(1)                   // Endlosschleife
    {
        P1 = 0x80;             // P0 Bit7 ON
        for(t=1200000; t>0; t--); // Verzög. 100ms
        P1 = 0;                // P0 Bit7 OFF
        for(t=1200000; t>0; t--); // Verzög. 100ms
    }
}
```

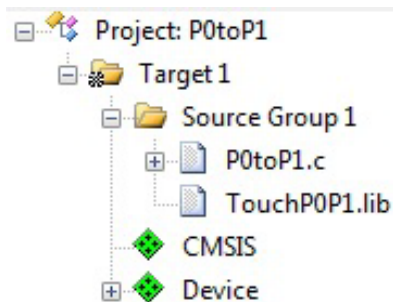
Projekt: BlinkP0P1

7 Programmierkonzepte auf MCB32

a) 8Bit Programmierung

mit P0- und P1-Ports
über Touchscreen

- TouchPOP1.h und .lib ins Projektverzeichnis legen
- TouchPOP1.lib ins Projekt aufnehmen
- TouchPOP1.h im Code einbinden



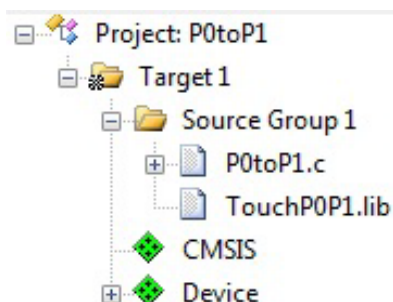
```
#include "TouchPOP1.h"

int main(void)
{
    InitTouchPOP1("1");
    while(1)
        P1 = P0;
}
```



b) Erweiterte 8Bit Programmierung

P0-, P1-Ports über Touchscreen sowie Grafik und integrierte Peripherie



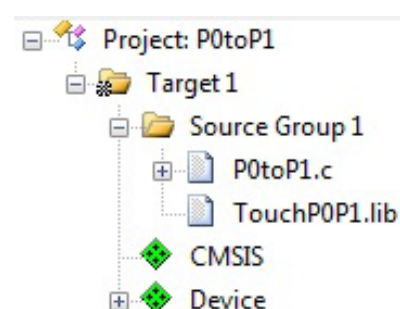
```
#include "TouchPOP1.h"

int main(void)
{
    InitTouchPOP1("1");
    ADCInit(1, "PC4");
    while(1)
        P1 = ADCGetVal(1);
}
```

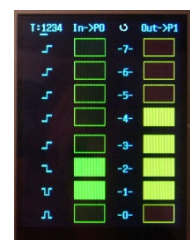


c) 8Bit Programmierung in 32Bit Registern

mit P0-, P1-Ports und Touchscreen

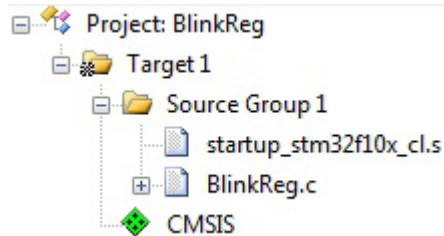


```
#include "TouchPOP1.h"
int main(void)
{
    InitTouchPOP1("1");
    RCC->APB2ENR |= 1<<9;
    ADC1->SQR3 = 14;
    while(1)
        P1 = ADC1->DR>>4;
}
```



d) Reine 32Bit Registerprogrammierung mit 8MHz

- SystemInit() leer definieren
- stm32f10x.h, Startup und NUR CMSIS einbinden

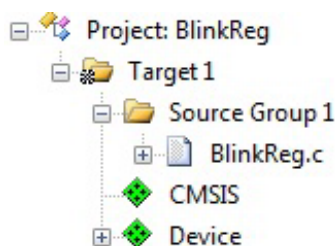


```
#include <stm32f10x.h>
void SystemInit(void) {}

int main(void)
{
    RCC->APB2ENR |= 1<<6;
    while(1)
        GPIOE->ODR = 0xAA00;
}
```

e) 32Bit Registerprogrammierung mit SystemInit und 72MHz

- Für SystemInit() im Device startup



```
#include <stm32f10x.h>

int main(void)
{
    RCC->APB2ENR |= 1<<6;
    while(1)
        GPIOE->ODR = 0xAA00;
}
```

f) CMSIS - Programmierung

- Der Systemtakt beträgt 72MHz

- use_STD_Periph_Driver beachten
- stm.....conf und alle benötigten Periph...h



```
#include "stm32f10x_gpio.h"

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_SetBits(GPIOE, GPIO_Pin_14);
}
```

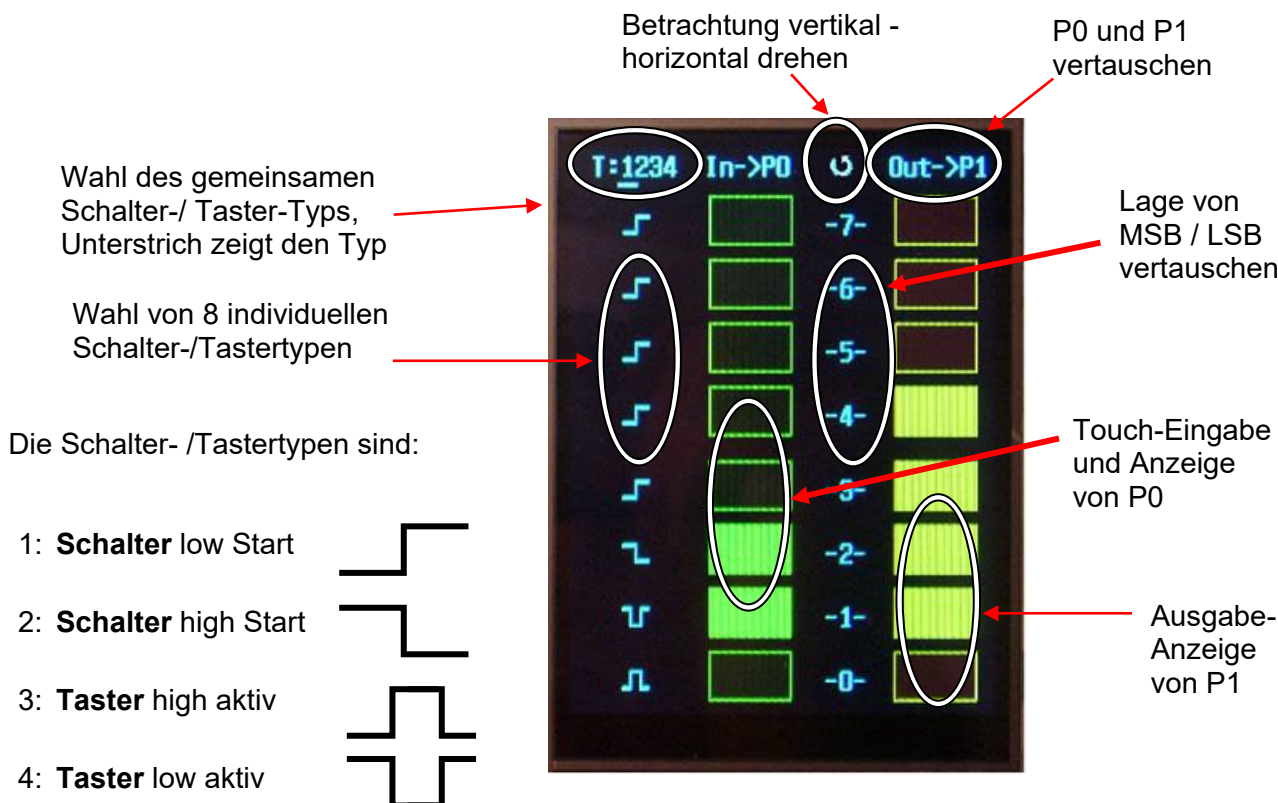
Professionelle hersteller- und controllerübergreifende CMSIS-Programmierung
Die benötigten StdPeriph_Driver Header- und Sourcedateien sind aus
Keil_v5/ARM/Pack/Keil/STM32F1xx_LIB/1.0.5/Device/StdPeriph_Driver/inc und /src
ins Projektverzeichnis zu kopieren und die Sourcedateien ins Projekt einzubinden.

alle) Hinweis zu Systemtakt und Verzögerungsschleifen:

SystemInit() schaltet den SystemClock von 8 MHz auf 72 MHz und baut für den Flash-speicherzugriff WaitStates ein. Dadurch laufen Verzögerungsschleifen nur 6x schneller. Der Compiler erlaubt sich, gleiche Verzögerungsschleifen an verschiedenen Programmstellen verschieden zu übersetzen. Wichtige Zeiten also immer überprüfen!

8 Anhang Touchscreen Kontrolle am µC-Board MCB32

Beschreibung der Touchscreen Oberfläche



Touchscreen Kontrolle aus dem Quellcode

Der Projekt-Ordner muss TouchP0P1.h und TouchP0P1.lib enthalten:

Im Projekt-Manager sind die Quelldatei.c und die Lib „TouchP0P1.lib“ aufzunehmen.

```
/* Beschreibung der 8 Bit P0- und P1-Kontrolle über den Touchscreen des MCB32
*****/
#include <stm32f10x.h> // Mikrokontrollertyp
#include "TouchP0P1.h" // P0-, P1-Definition. Angepasst für REV C oder D

void main(void) // Hauptprogramm
{
    InitTouchP0P1 ("1"); // Touchscreen aktivieren. Bei „0“ ist SysTick
                        // -Timer abgeschaltet.
    while(1) { } // Benutzerprogramm
}
```

1) **InitTouchP0P1 ("0");**

Der Touchscreen bleibt ausgeschaltet

P0 ist als Input, P1 als Output konfiguriert

2) **InitTouchP0P1 ("1") .. ("1 r m p");**

Der Touchscreen wird aktiviert und konfiguriert, einfachste Konfiguration mit InitTouchP0P1 ("1").

1...4: Gemeinsamer Schalter-/Tastertyp

p: P0 aussen, sonst mittig.

m: MSB oben/rechts, sonst unten/links.

r: Rotiert horizontal, sonst vertikal.

9 Anhang Verzögerungen und Laufzeiten im µC-Board MCB32

a) InitTouchP0P1 ("**0**") ; ohne Touchscreen

<pre> int main(void) // Hauptprogramm { long t; InitTouchP0P1("0"); // P0P1-TScreen OFF while(1) // Endlosschleife { P1_0=1; // Pin0: ON for(t=120000;t>0;t--); // Delay ca.10ms P1_0=0; // Pin0: OFF for(t=1200000;t>0;t--); // Delay ca.100ms } } </pre>	<p>Laufzeit in der while-Schleife:</p> <p>0ms</p> <p>10ms</p> <p>0ms</p> <p>100ms</p> <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin: 2px;">ON</div> <div style="border: 1px solid black; padding: 5px; margin: 2px;">OFF</div> </div>
--	---

long t;	// Immer long und auf 0 prüfend!
for (t = 12; t > 0; t --);	// 1.0µs
for (t = 12000; t > 0; t --);	// 1.0ms
for (t = 120000; t > 0; t --);	// 10.0ms (für Entprellung)
for (t = 1200000; t > 0; t --);	// 100ms (für Blinken)
for (t = 12000000; t > 0; t --);	// 1.00s

P1 = 0;	// Durchschnittliche Operationszeit
P1 = 1;	// je 0.17µs bis 0.5µs

b) InitTouchP0P1 ("**1**") ; Touchscreenauffrischung erfordert zusätzliche Laufzeit

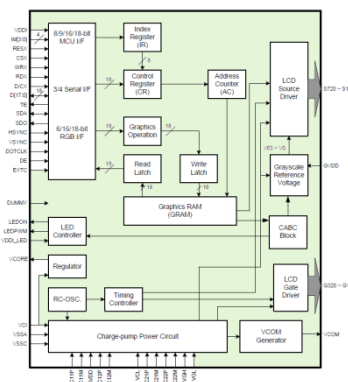
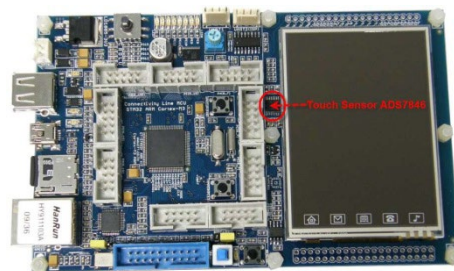
<pre> int main(void) // Hauptprogramm { long t; InitTouchP0P1("1"); // P0P1-TScreen ON while(1) // Endlosschleife { P1_0=1; // Pin0: ON for(t=120000;t>0;t--); // Delay ca.10ms P1_0=0; // Pin0: OFF for(t=1200000;t>0;t--); // Delay ca.100ms } } </pre>	<p>Laufzeit in der while-Schleife:</p> <p>6.4ms</p> <p>10ms</p> <p>6.4ms</p> <p>100ms</p> <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin: 2px;">ON</div> <div style="border: 1px solid black; padding: 5px; margin: 2px;">OFF</div> </div>
---	---

Umzeichnen einer P0-/P1-Fläche	// 6.4ms pro Fläche
Somit kürzester Impuls an P1	// 0.25µs, evtl. + 6.4ms
Touchcheck per Interrupt	// alle 25ms + 3µs
	// bei Touch + 6.4ms (da 1 Fläche)

10 Anhang Grafikprogrammierung

10.1 Einleitung

Der MCB32 Kit arbeitet mit einem TFT –LCD Color Grafik Display 320x240Pixel (3.2") und einer Touch-Sensor (Folie). Der Grafik-Display wird über einen Chip ILI9341 angesteuert und der der Touch-Sensor über einen ADS7846. Beide Chips kommunizieren via die SPI Schnittstelle mit dem ARM-Prozessor resp. der Library.



Der ILI9341 Chip steuert den eigentlichen Bildschirm an. Der Chip hat 720 Source-Ausgänge und 320 Gate-Ausgänge um die einzelnen Dots (Pixels) ein und auszuschalten. Zudem können 172.8KByte RAM genutzt werden.

Der Chip wird über ein spezielles 9Bit SPI Interface angesteuert. Das heisst der Datentransfer ist beschränkt durch die serielle Datenübertragungsrate. Mehr Informationen zum Chip unter: <http://www.adafruit.com/datasheets/ILI9341.pdf> oder andere Quellen.

10.2 Hardwarenahe Beschreibung der Displayansteuerung

Verbindung Display TFT320x240 mit dem ARM: via GPIO Pins und SPI3 (9Bit)				
Pin	Beschreibung	Funktion	Port ARM	
CS	Chipselect	CS#	PC8 (Out)	
SCL	Clock	SPI3:SCK3	PC10 (Clk)	
SDO	Data Out to ARM	SPI3:MISO3	PC11 (Inp)	
SDI	Data IN from ARM	SPI3:MOSI3	PC12 (Out)	
BL	IRQ to ARM	GPIO: IRQ	PD7 (Out)	

Verbindung Touch-Sensor ADS7846 mit dem ARM: via GPIO Pins				
Pin ADS7846	Beschreibung	Funktion	Port ARM	
CS	Chipselect	CS#	PE6 (Out)	
DCLK	Clock	SPI:SCK	PE7 (Clk)	
DOUT	Data Out to ARM	SPI:MISO	PE4 (Inp)	
DIN	Data IN from ARM	SPI:MOSI	PE5 (Out)	
PENIRQ	IRQ to ARM	GPIO: IRQ	PE3 (Inp)	

10.3 Grafikprogrammierung

10.3.1 Fonts


Es steht 1 Font zur Verfügung. Ein „7*11“ ASCII Font.
Der Font kann in der Library „TouchGrafik.c“ individuell erweitert werden.



10.3.2 Grafikfunktionen [3]

Grafikfunktionen:		Version und Stand: 2107.00 (D)	
Funktion	Beschreibung	Parameter	Beispiel
InitTouchP0P1 ()	Hier wird neben dem Display auch der Komfort für P0 und P1 initialisiert. Schulübungen mit Bitmanipulation.	Siehe Kapitel 6	<code>InitTouchP0P1("1");</code>
Für normale Grafikanwendungen können folgende Befehle aus der Lib verwendet werden:			
InitTouchScreen ()	Touchscreen ohne P0P1 für Text, Grafik, Peripherie. Kein SysTick im Hintergrund für Refresh.	-	<code>InitTouchScreen();</code>
setScreenDir (DIR)	Setzt die Schreibrichtung des Displays	HOR und VER	<code>setScreenDir (HOR);</code> <code>setScreenDir (VER);</code>
char getScreenDir ()	Gibt die Schreibrichtung zurück	HOR=0 und VER=1	<code>if(getScreenDir()==VER)</code> <code>{clearScreen (BLUE);}</code>
clearScreen (color)	Löscht den Bildschirm mit der angegebenen Farbe. Funktioniert nur mit Einstellung setScreenDir(VER) .	long color	<code>clearScreen(BLACK);</code>
plotDot (X,Y,color) ○	Zeichnet ein DOT an der Stelle X,Y mit der Farbe color. a,b,c unsigned int.		<code>plotDot(120,120,WHITE);</code>
Circle (X,Y,Radius, Tick, Color, Fill)	Zeichnet einen Kreis an der Stelle X,Y mit dem Radius r. Die Kreislinie wird mit der Dicke „Tick: 0..100“ gezeichnet wenn Fill 0 ist. Fill =1 füllt angegeben ist so wird der Kreis gefüllt den Kreis.	div	<code>circle(50,80,20,2,GREEN,0);</code>
ellipse (h,k,rx,ry,tick,color, fill)	h und k beschreiben den Mittelpunkt der Ellipse. rx und ry die Radien, Tick, Color und Fill wie beim Kreis,		
rectan (x1,y1,x2,y2,tick,color, fill)	Zeichnet ein Rechteck von x1,y1 zu x2,y2.		<code>rectan(100,150,140,180,1, RED,1);</code>
plotFilledRect (x1,y1,dx,dy,color)	Gefülltes Rechteck von x1, y1 nach x1+dx, y1+dy mit Farbe color		<code>filledRect (10, 20, 50, 60, RED);</code>
textxy (String,x,y,For_col, Back_Col)	Schreibt an der Stelle x,y, mit der Farbe For_col und der Hintergrund-Farbe den String.		<code>textxy(" MCB32 Lib</code> <code>Version:", 2, 32,</code> <code>BLACK, YELLOW);</code>
line (x1,y1,x2,y2,thick,color)	Zeichne Linie von X1,y1 nach x2,y2 mit der Dicke und der Farbe		<code>line(5,110,315,110,2,WHITE);</code>
getTSCxy ()	Erfasst die x/y - Werte der berührten Position.		<code>getTSCxy();</code>

Aufsetzen eines ersten Programmes

<code>getTSCx ()</code>	Gibt die x-Position der letzten Erfassung zurück.		<code>xPos = getTSCx();</code>
<code>getTSCy ()</code>	Gibt die y-Position der letzten Erfassung zurück.		<code>yPos = getTSCy();</code>
<code>getTSCtouched ()</code>	Touchscreenberührung: Rückgabe 0 / 1 0: unberührt 1: während Berührung <div>  </div>		<code>if (getTSCtouched ())</code> <code>{</code> <code> }</code> <code>}</code>
<i>Bemerkung: getTSCxy() je nach Fall zuerst ausführen.</i>			

Touchscreen Textfunktionen

vertikal, 20 Zeilen à 30 Zeichen	horizontal, 15 Zeilen à 40 Zeichen
<pre> 0: Text-, Variablenausgaben 1: ----- Variablenwerte dezimal: 0, -444, 1234567890 Variablenwerte binär: 1 Bit: 0, 8 Bit: 1010'1010 16 Bit: 1111'1000'1111'1000 Variablenwerte hexadezimal: 16 Bit: 0x2AFB Textfarbenwechsel: 32 Bit: 1111'1000'1111'1000 1111'1000'1111'1000 18: 19: </pre>	<pre> 0: Text-, Variablenausgaben 1: ----- Variablenwerte dezimal: 0, -444, 1234567890 Variablenwerte binär: 1, 8, 16 Bit 32-Bit: 1111'1000'1111'1000:1111'1000'1111'1000 13: 14: </pre>

Funktion	Beschreibung	Beispiel
<code>InitTouchScreen ();</code>	Initialisiert den Touchscreen ohne P0P1 für Text, Grafik und Peripherie	<code>InitTouchScreen ();</code>
<code>setTextcolor (long color);</code>	Farbwechsel für nachfolgenden Text	<code>setTextcolor (WHITE);</code>
<code>print (char *txt);</code>	Schreibt Text hinter die letzte Position	<code>print ("Text");</code>
<code>println (char *txt);</code>	Schreibt Zeile hinter die letzte Position und springt an den nächste Zeilenanfang	<code>println ("Text");</code> <code>println ("");</code>
<code>printAt (char n, char *txt);</code>	Schreibt Text an den Anfang der Zeile mit Nummer n	<code>printAt (12, "Text");</code>
<code>printBin (char n, long num);</code>	Konstanten- und Variablenwerte im Binär-code wie 1111'0000 mit der Bitanzahl n	<code>printBin (8, 250);</code> <code>printBin (32, variable);</code>
<code>printHex (char n, long num);</code>	Konstanten- und Variablenwerte im Hex-code wie 0xFF00123E mit der Bitanzahl n	<code>printHex (8, 250);</code> <code>printHex (32, variable);</code>
<code>printDec (char form long num);</code>	Ganzzahlige Werte aller Typen mit Feldlänge und Vorzeichen in form - vorgegebene Feldlänge für Typ unsigned - vorgegebene Feldlänge mit Vorzeichen - wertabhängige Feldlänge, Typ unsigned - wertabhängige Feldlänge mit Vorzeichen da zu kurze Feldlängen erweitert werden	<code>printDec (12, variable);</code> <code>printDec (-8, 123456);</code> <code>printDec (1, variable);</code> <code>printDec (-1, -123456);</code>

10.3.2.1 Farbliste

Die **nebenstehende** Farbliste zeigt die vordefinierten Farben. Weitere Farben müssen gemäss dem Muster:

RRRR`RGGG`GGGB`BBBB zusammengestellt werden.

Der 16 Bit Farbcode hat 32 Rot-, 64 Grün- und 32 Blauanteile

in Bit: **5** Bit **R**, **6** Bit **G**, **5** Bit **B**

RRRR`RGGG`GGGB`BBBB

Mischbeispiel:

long sattgrün = 63<<5; 0000`0111`1110`0000

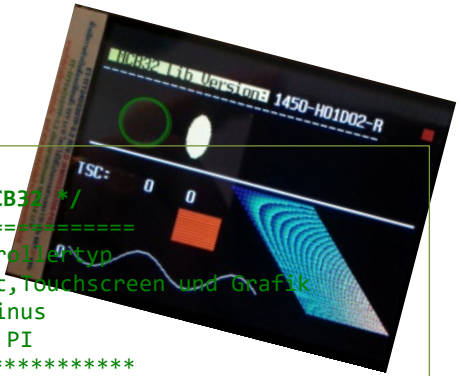
long hellgrün = 15<<11 + 63<<5 + 15; 0111`1111`1110`1111

Die genauere Beschreibung befindet sich im ILI 9341 Manual.

Definiere die Farbe für den Display:		
Color = RRRR`RGGG`GGGB`BBBB		
Name	Color #	
#define no_bg	0x0001	// No Color Back Ground
#define BLACK	0x0000	
#define WHITE	0xFFFF	
#define RED	0x8000	
#define GREEN	0x0400	
#define DARK_GREEN	0x1C03	// weber
#define BLUE	0x0010	
#define YELLOW	0xFF00	
#define DARK_YELLOW	0x8403	// weber
#define CYAN	0x0410	
#define MAGENTA	0x8010	
#define BROWN	0xFC00	
#define OLIVE	0x8400	
#define BRIGHT_RED	0xF800	
#define BRIGHT_GREEN	0x07E0	
#define BRIGHT_BLUE	0x001F	
#define BRIGHT_YELLOW	0xFFE0	
#define BRIGHT_CYAN	0x07FF	
#define BRIGHT_MAGENTA	0xF81F	
#define LIGHT_GRAY	0x8410	
#define LIGHT_BLUE	0x841F	
#define LIGHT_GREEN	0x87F0	
#define LIGHT_CYAN	0x87FF	
#define LIGHT_RED	0xFC10	
#define LIGHT_MAGENTA	0xFC1F	
#define DARK_GRAY	0x4208	
#define GRAY0	0xE71C	
#define GRAY1	0xC618	
#define GRAY2	0xA514	
#define GRAY3	0x630C	
#define GRAY4	0x4208	
#define GRAY5	0x2104	
#define GRAY6	0x3186	
#define BLUE0	0x1086	
#define BLUE1	0x3188	
#define BLUE2	0x4314	
#define BLUE3	0x861C	
#define CYAN0	0x3D34	
#define CYAN1	0x1DF7	
#define GREEN0	0x0200	
#define GREEN1	0x0208	

10.4 Musterprogramm für Grafikfunktionen

Das folgende Programm zeigt die Möglichkeiten der Library.



```

/** @file grafikfunktionen_1.c
 * @brief Zeigt die grundlegenden Grafikfunktionen Version I von MCB32 */
//=====Includes=====
#include <stm32f10x.h> // Mikrocontroller-Typ
#include "TouchP0P1.h" // P0/P1, 8Bit, Touchscreen und Grafik
#include <math.h> // lib für Sinus
#define PI 3.14159f // Konstante PI
//*****Implementation*****
int main(void) // Hauptprogramm
{
    long t; // Verzögerungsvariable
    float rad;
    unsigned char uc_val, color_toggle=0; // Hilfsvariablen;

    char LIBVer[]=dMCB32_LibVersion; // Hole Lib Version und zeige sie an
    InitTouchScreen(); // Init. der Display Hardware
    setScreenDir (HOR); // setze Richtung Display. 0,0 bei Resetbutton

    textxy(" MCB32 Lib Version:", 2, 32, BLACK, YELLOW);
    textxy(LIBVer, 160, 32, WHITE, BLACK);
    printAt(2, "----- "); // Schreibe auf der 2ten Zeile den Text
    circle(50,80,20,2,GREEN,0); // Zeichne Kreis
    ellipse(100, 80, 10,20,1,YELLOW,1); // Zeichne Ellipse
    rectan(100,150,140,180,1,BRIGHT_RED,1); // Zeichne Rechteck
    line(5,110,315,110,2,WHITE); // Zeichne Linie
    for(uc_val =0;uc_val<80;uc_val++){ // Zeichne mit plotDot() ein Muster
        for (t=0; t<100;t++){
            plotDot(140+uc_val+t,115+t,uc_val*t*8);
        }
    }
    for (t=0; t<180;t++){ // zeichne Sinus mit plotDot
        rad = 4*t * PI / 180; // Berechnen des Bogenmaßwinkels
        plotDot(10+t,(210+12*sin((double)(rad))),WHITE);
    }
    plotFilledRect ( 300, 20, 10, 10, RED ); // zeichne ein gefülltes Rechteck
    GPIOInit("PEH",00000000);
    GPIOE->CRH &= 0x00000000; // Konfiguriere GPIOE für
    GPIOE->CRH |= 0x22222222; // General purpose output push-pull, 2MHz
    while(1){
        getTSCxy(); // initialisiert Touch, liest die Werte für getTSCx() und getTSCy() ein.
        printAt(8, "TSC:");
        if(getTSCx() <= 320){printDec(5, getTSCx());} // grenze Bereich für Rückgabewerte ein und gib sie aus.
        if(getTSCy() <= 320){printDec(5, getTSCy());}
        printAt(13, ""); printBin(1,getTSCtouched()); // Schreibe Berührungsstatus auf den Screen
        uc_val = getTSCtouched(); // Hole Touchwert 0,1 Debugging
        GPIOPutByte("PEH",getTSCtouched()); // zeige via LED ob Touch gedrückt wurde
        if(uc_val==1){
            for (t=0; t<220;t++){
                rad = 4*t * PI / 180; // Berechnen des Bogenmaßwinkels
                if(color_toggle==0) {
                    plotDot(10+t,(210+12*sin((double)(rad))),BRIGHT_BLUE);
                } else {
                    plotDot(10+t,(210+12*sin((double)(rad))),WHITE);
                }
            }
            color_toggle=color_toggle^0x01; // Toggle Color für den nächsten Sinus. Spielerei
        }
    }
}

```

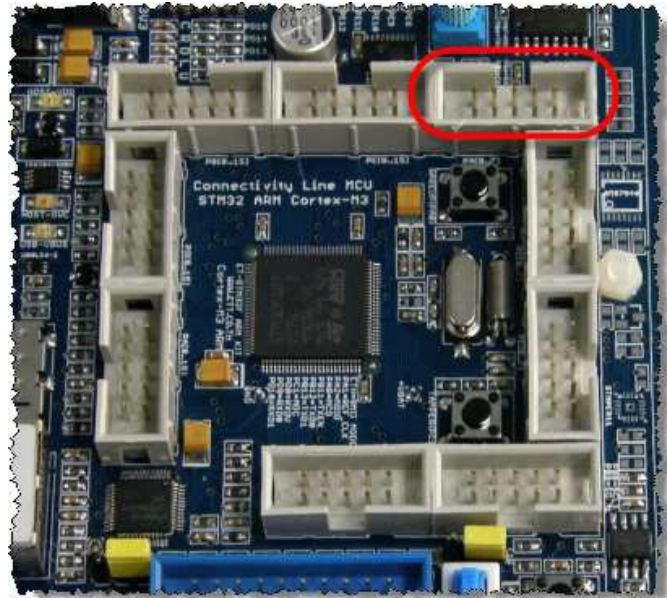
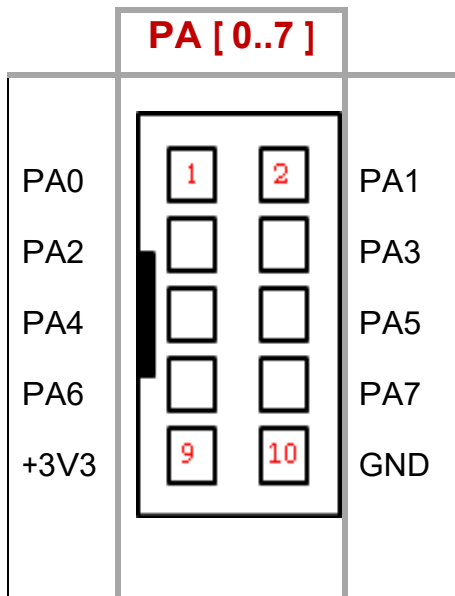
(Änderungen jederzeit möglich. Siehe Dokumentation.)

11 Anhang Anschlüsse am μ C-Board MCB32

11.1 Stecker Belegung am Beispiel Port A

Der Port A kann als Eingang (parallel zu P0) benutzt werden. Grundsätzlich sind alle anderen Ports gleich. Pin9 ist 3V3 und Pin10 ist GND.

1. GrSteckerbelegung für 10pol. Stecker PA[0..7]



Die Belegung des 10poligen Steckers sieht wie oben abgebildet aus. Die roten Zahlen definieren die Adern des Flachbandkabels mit roter Ader = Pin1.

2. Original Belegung PA[0..7]

Die Original Pin-Belegung von Stecker PA[0...7] ist wie in der nebenstehenden Tabelle. Diese Einstellung wird im Versuch aber nicht benötigt.

Pin	Funktion	Module/Device	
PA0	Wakeup	Switch Wakeup	
PA1	RMII_REF_CLK	Ethernet LAN	
PA2	RMII_MDIO	Ethernet LAN	
PA3	-	-	
PA4	-	-	
PA5	SPI1_SCK	SD Card CLK	
PA6	SPI1_MISO	SD Card DAT0	
PA7	SPI1_MOSI	SD Card CMD	

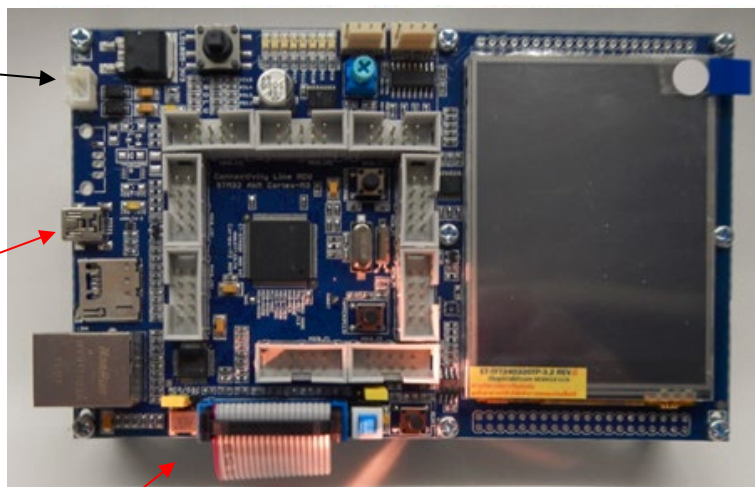
11.2 Entwicklungsanschlüsse

Entwicklungsanschlüsse

Fremdspeisung
genau 5V! oder ...

USB Speisung (**roter Stecker**)

USB - ST-Link (**schwar-
zer USB Stecker**)
Zielsystemdebugging,
Boardspeisung
wie oben



Bootloadschalter:

Ungedrückt lassen **LED OFF**

Reset-Taster:

Programmrestart

Digitale Ein- und Ausgaben am µC-Board MCB32. **ACHTUNG** mit Potentiometer (Pot)

P1: 8x onboard LEDs
parallel zu 8x extern LEDs



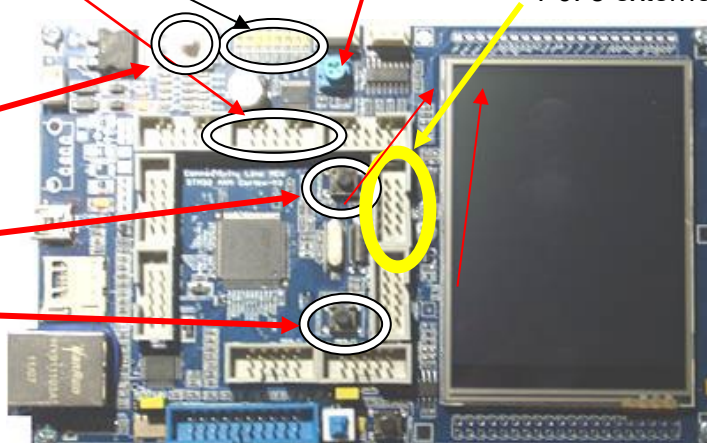
Pot auf Anschlag Uhrzeigersinn
drehen, da gleichzeitig P0_4

P0: 8 externe Schalter

ControlStick

Button 1

Button 0



// In TouchP0P1.h definierte Pin-Bezeichnungen PA_0 .. PD_11, ohne Bezeichner wie Button .. !

```
char Button0    = PA_0;
char Button1    = PC_13;
```

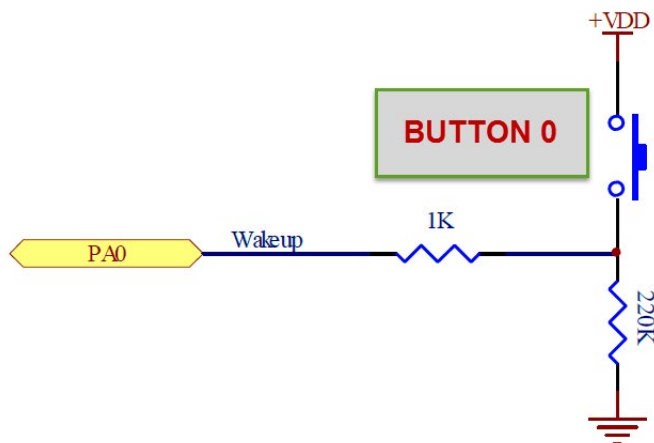
```
// Bitwert 1/0, aktiv low, prellt wenig
```

```
char Stick      = PD_High;
char StickSelect = PD_15;
char StickDown  = PD_14;
char StickLeft  = PD_13;
char StickUp    = PD_12;
char StickRight = PD_11;
```

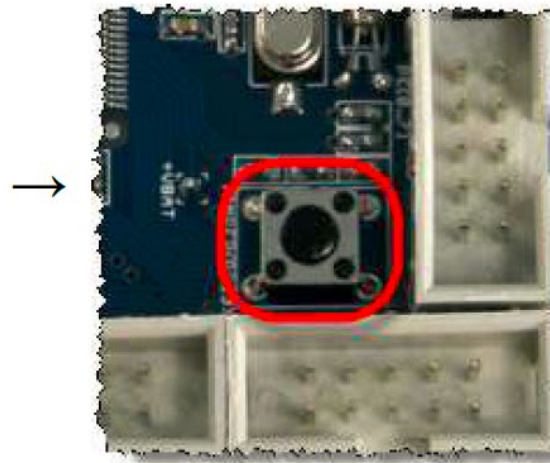
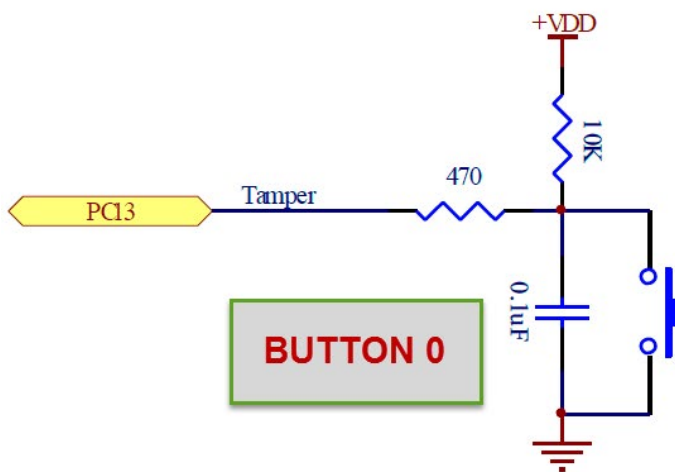
```
// als Byte 0xF8 open, aktiv low, alle entprellt
// Bitwert 1/0; Bytewert 0x80
//          1/0;          0x40
//          1/0;          0x20
//          1/0;          0x10
//          1/0;          0x08
```

Aufsetzen eines ersten Programmes

Button 0 (PA 0)

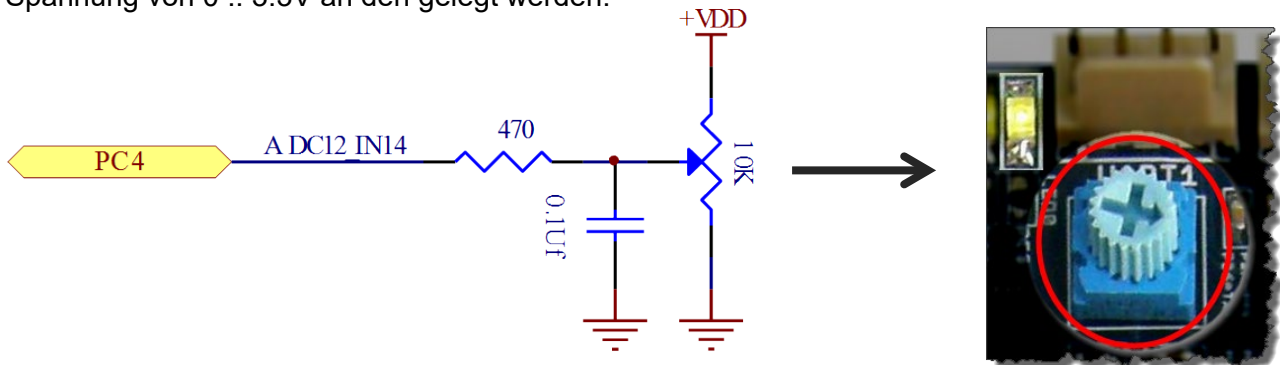


Button 1 (PC 13)



Potentiometer (PC 4) // resp. P0_4 (Library)

Wenn der Port PC4 als Analog-Input (AD-Wandler) geschaltet ist kann mit dem Potentiometer eine Spannung von 0 .. 3.3V an den gelegt werden.



11.3 LED auf Board aktivieren

Per Default sind die Ports auf dem MCB32 mit der Touch_Lib floatend um die IO's vor **Fehlmanipulationen** zu schützen.

Mit den folgenden Befehlen wird Port PE[8..15] so gesetzt, dass die LEDs auf dem Board **aktiv** angesteuert sind. Siehe Beispiel Code im Listing unten.

```
GPIOE->CRH    &= 0x00000000;    // Configure the GPIOE for
GPIOE->CRH    |= 0x22222222;    // General purpose output push-pull, 2MHz
```

```
int main(void)                // Hauptprogramm
{
    long t;                    // Verzögerungsvariable
    InitTouchPOP1("1");        // P0,P1 auf Touchscreen ON

    // GPIOE->CRH    &= 0x00000000; // Configure the GPIOE for
    // GPIOE->CRH    |= 0x22222222; // General purpose output push-pull, 2MHz

    P1=0x00;                    // Zaehlung nullen
    while(1)                    // Endlosschleife
    {
        if(P0_0)                // Zählung nur mit P0_0 = 1
        {
            if(!P0_1) P1++;      // Zählung aufwärts, wenn P0_1 = 0
            else P1--;           // Zählung abwärts, wenn P0_1 = 1
        }
        for(t=0;t<1200000;t++); // Zählverzögerung ca. 100msek
    }
}
```


12 Anhang: Referenzen

- [1 KEIL-Link1, «<http://www2.keil.com/mdk5/.....>,» [Online]. Available:
] http://www2.keil.com/mdk5?mkt_tok=3RkMMJWWfF9wsRonua7BZKXonjHpfsX77OkuXaaylMI%2F0ER3fOvrPUfGjl4ASsJkl%2BSLDwEYGJlv6SgFSbHHMbNhwrgJUxk%3D.
[Zugriff am 09 03 2016].
- [2 E. Malacarne, «MCB32_Flyer_KEIL_Installation_Vxxxx,» Cityline AG, Zürich, 2016.
]
- [3 R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
]
- [4 J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3 Hrsg., Bd. 1,
] Elsevier, Hrsg., Oxford: Elsevier, 2014.
- [5 R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1 Hrsg., www.mitp.de,
] Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.
- [6 Diller, «System Timer,» 06 07 2014. [Online]. Available: http://www.diller-technologies.de/stm32.html#system_timer. [Zugriff am 06 07 2014].
- [7 A. Limited, «DDI0337E_cortex_m3_r1p1_trm.pdf,» ARM Limited,
] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf, 2005, 2006 ARM Limited.
- [8 A. C. Group, «http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm,» 2007. [Online].
]
- [9 E. Malacarne, *Glossar Malacarne*, V11 Hrsg., Rüti: Cityline AG, 2014.
]
- [1 R. Weber, «General Purpos Input Output,» 2014.
0]
- [1 STM, «STM32F10x Standard Peripherals Firmware Library,» STM, 2010ff.
1]
- [1 E. Schellenberg und E. Frei, «Programmieren im Fach HST,» TBZ, Zürich, 2010ff.
2]