C/C++ mit Mbed-OS für ARM Cortex M-Prozessoren (STM32F411RE)

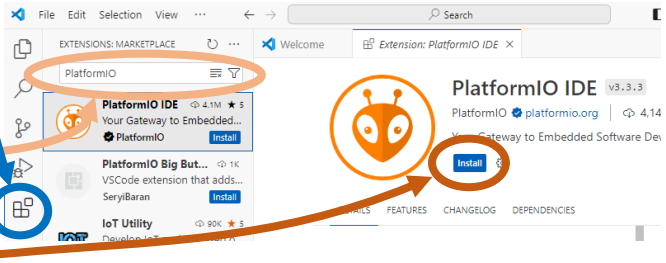
Aufgabe 1: Installation von Visual Studio Code mit Arm Tools (für zu Hause)

Als Entwicklungsumgebung (= **I**ntegrated **D**evelopment **E**nvironment = IDE) werden wir das Programm *Visual Studio Code* mit dem *PlatformIO*-Plugin für Mikrocontroller verwenden. Diese Installationsanleitung ist für zu Hause, in der Schule ist alles bereits installiert.

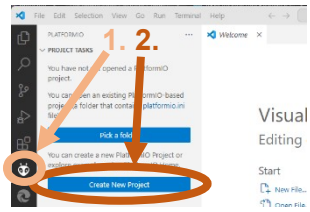
1.1. Laden Sie sich den Installer für VS Code von der Webseite https://code.visualstudio.com/ herunter und installieren Sie das Programm.

1.2. Wählen Sie mit einem Mausklick die Extension[1]-Liste ganz unten in der linken Menüspalte aus und suchen Sie nach „PlatformIO“.

1.3. Installieren Sie das „PlatformIO“-Plug-in[2].



Aufgabe 2: Anlegen eines Projektes (Alternative 1)



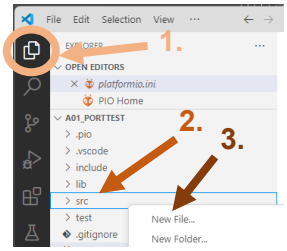
2.1. Klicken Sie links in der Modusleiste das PlatformIO-Symbol an und wählen Sie „Create New Project“ in der PlatformIO-Spalte.

2.2. Wählen Sie im neuen Fenster „PIO Home“ rechts „New Project“.

2.3. Geben Sie im folgenden „Project Wizard“ die Projektdaten ein, z.B.

- Name: `C01_Porttest`
- Board: `ST Nucleo F411RE`
- Framework: `Mbed`
- Use default location: kein Häkchen. Wählen bzw. erzeugen Sie Ihr Verzeichnis für µController-Projekte, z.B. *H:\TGI12_IFTP\uController_Projects*

und übernehmen Sie ihre Auswahl mit „Finish“. Beim allerersten Projekt kann das Herunterladen des SDKs (Software Development Kit) für den Arm-Prozessor ziemlich lange dauern.



2.4. Sollte ein Dialog erscheinen, ob Sie den Autoren aller Dateien in dem Projektverzeichnis übergeordneten Verzeichnis vertrauen, setzen Sie das Häkchen und bestätigen sie es.

2.5. Erzeugen Sie eine C++-Datei *porttest_main.cpp* im Unterordner *src*[3] des Projektes wie in der Abbildung. Klicken Sie dazu mit der rechten Maustaste auf das Verzeichnis und wählen Sie „*New File …*“ aus.

2.6. Geben Sie die nebenstehende Grundstruktur eines Mbed-Programms in die Datei ein. Das *#include* bindet die Deklarationen des Betriebssystems ein, die 1 in der *while*-Scheile bedeutet „true“, da alle Werte ungleich 0 als „true“ interpretiert werden. Speichern Sie die Datei mit *Datei* > *Save* oder *Strg-S*.

Alternativ können Sie auch den Quellcode aus dem Anhang oder der ausgeteilten Vorlagendatei in die Datei kopieren.

```
#include "mbed.h"

int main() {
    // Initialisierungen

    // Endlosschleife
    while (1) {

    }
}
```

1 extension = engl. für „Erweiterung“
2 „PlatformIO Core“ (command line interface außerhalb VS Code) braucht nicht installiert zu werden.
3 „src“ ist die Abkürzung für engl. „source“ = „Quelle“, gemeint ist der Quellcode des Programms.

Aufgabe 3: Anlegen eines neuen Projekts durch Kopieren (Alternative 2) bzw. Öffnen

Alternativ zum Anlegen eines neuen Projektes kann auch ein bestehendes, z.B. das Vorlagenprojekt, kopieren werden. Überspringen Sie diese Aufgabe, wenn sie die vorherige gemacht haben.

3.1. Kopieren Sie das gesamte Projektverzeichnis und benennen Sie das gesamte Verzeichnis um.
Beispiel: `Volage_cpp` → `Vorlage_cpp-Kopie` → `C01_Porttest`

3.2. Öffnen Sie das Verzeichnis in Visual Studio Code mit *File* > *Open Folder*.
Ändern Sie den Namen des Programms im `src`-Verzeichnis des Projektes. Entweder im Dateimanager ihres Betriebssystems oder in VS Code mit *Rechtsklick* > *Rename…* bzw. *Linksklick* und *F2*. *Beispiel*: `vorlage.cpp` → `porttest_main.cpp`

3.3. Dateien aus anderen Projekten können Sie durch Kopieren in das *src*-Verzeichnis dem Projekt hinzufügen. Dies kann auch durch Ziehen von Dateien auf die Ordner in VS Code erledigt werden. Bibliotheksdateien sollten in Unterordner im *lib*-Verzeichnis hinzugefügt werden.

Aufgabe 4: Erstellen eines ersten Mbed-Programms „Porttest“

In der *porttest_main.cpp*-Datei soll ein Port-Test-Programm erstellt werden, das von PortB[7:0] (=Port 3) Schalter einliest und auf PortC[7:0] (=Port 1) herausschreibt.

4.1. Deklarieren Sie einen Ausgangsport *port1* zwischen dem *#include* und der *while*-Schleife:

```
PortOut port1(PortC,0x00FF);  // PortOut: Dattentyp/Klassenname,
                              // port1   : Variablen-/Objektname
                              // PortC   : in mbed.h deklarierte Konstante
                              // 0x00FF  : „Maske“, nur die unteren 8 Bit werden
                              //           verändert. Maskenangabe kann weggelassen
                              //           werden, dann werden alle Bits verändert.
```

Hinweis: alles hinter // sind Kommentare und ist für die Funktion nicht relevant.

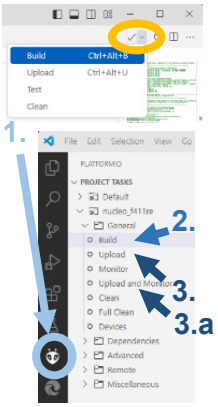
4.2. Deklarieren Sie einen Eingangsport (*PortIn*) für die Schalter auf Port B, von dem nur die unteren 8 Bit gelesen werden.

4.3. Kopieren Sie in der Endlosschleife der *main()*-Routine *port3* auf *port1*.

Hinweis: *PortOut* besitzt eine *void write(int value)*-Methode, *PortIn* eine *int read()*. Diese können wie folgt verwendet werden:

```
port1.write(42);          // port1 = 42;          ginge auch (Kurzschreibweise)
int value = port3.read(); // int value = port3;  ginge auch (Kurzschreibweise)
```

Aufgabe 5: Ausführen und Testen eines Programms



5.1. *Kompilieren*

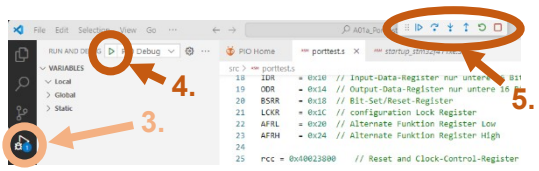
Wählen Sie im Menü rechts oben mit dem Haken *Build*. Alterantiv geht dies auch im PlatformIO-Menü (1.) mit *Build* (2.) oder Sie drücken Strg-Alt-B.

Korrigieren Sie etwaige Syntaxfehler. Durch Strg-Klick auf den Dateinamen mit Zeilenangabe in der Fehlermeldung gelangen Sie zu der Zeile mit dem Fehler.

5.2. *Download bzw. Upload in die Hardware*

Laden Sie mit *Upload* (3.) bzw. Strg-Alt-U das Programm in den angeschlossenen µController. Mit *Upload and Monitor* (3.a) starten Sie gleich einen Terminal um *printf*-Statements[4] aus ihrem Programm anzeigen zu lassen.

4 siehe Formelsammlung. Beispiel: `printf("Ich heiße %s und bin %i Jahre alt. ", name, alter);`



Alternativ können Sie auch den *Debugger* verwenden. Mit dem Run-Button (Dreieck) starten Sie das Programm. Anschließend können Sie mit den Debugger-Control-Buttons schrittweise durch das Programm gehen oder es laufen lassen. Wenn Sie vor die Zeilennummer einer Zeile mit einer Instruktion klicken, können Sie auch sogenannte „Breakpunkte“ setzen, an denen bei Erreichen das Programm anhält.

5.3. *Hinweis*: sollte es Probleme mit Standard-Upload-Protocol (ST-Link) geben, können Sie in der *platformio.ini*-Datei folgende Zeile einfügen: `upload_protocol = mbed`
Damit wird die erzeugte Binärdatei *firmware.bin* im .pio-build-Ordner auf den als USB-Speicher angemeldeten Controller kopiert.

5.4. *Programmtest*

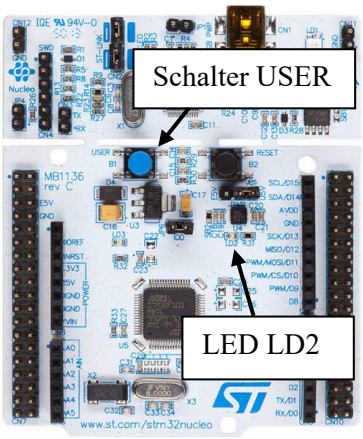
Schließen Sie die **Schalter** an PB[7:0] (**P3**- obere Reihe rechts) an und die **LEDs** an PC[7:0] (**P1**-obere Reihe links) an. Testen Sie das Verhalten der Schaltung.

Ports der oberen Reihe:
PC[7:0]-PC[15:8]-PB[7:0]

5.5. *Schließen des Projektes*

Wenn Sie ein neues Projekt anlegen möchten, sollten Sie vorher das aktuelle Projekt mit dem Menüeintrag *File > Close Folder* schließen. Um das Projekt wieder zu öffnen, können Sie *File > Open Folder* verwenden.

Aufgabe 6: Erstellen eines „Blinklicht“-Programms



In dieser Aufgabe soll ein Blinklicht-Programm erstellt werden, das mit einer Periodendauer von einer Sekunde blinkt.

6.1. Erstellen Sie gemäß den vorherigen Aufgaben ein neues Projekt *C02_Blinky*.

6.2. Suchen Sie aus der Vorlage die Deklaration für einen einzelnen Pin heraus und kopieren Sie sie hinter den Kommentar mit den Beispieldeklarationen. Verwenden Sie als Parameter angegebenen Wert LED1, der dem Pin PA_5 entspricht.

6.3. Mit dem Befehl `wait_us(int us)` kann man eine in µs angegebene Zeitspanne warten. Erstellen Sie damit ein Programm, das mit einer Periodendauer von einer Sekunde die LED LD2 auf dem µControllerboard blinken lässt.

6.4. Halten Sie bei gedrücktem, blauen USER-Button (als BUTTON1 (=PC_13) vordefiniert) das Blinken der LED LD2 an. Nach dem Loslassen soll die LED weiterhin blinken.

Aufgabe 7: Erweiterung des Porttest-Programms (Zusatzaufgabe)

7.1. Geben Sie auf den Bits [15:8] (=Port 2) den von den Schaltern eingelesenen Wert invertiert aus.

Hinweis: der ‚-‘-Operator berechnet das 2-er Komplement, der ‚~‘-Operator das 1er.

Hinweis: Auch wenn Sie bei der *PortOut*-Deklaration von *port2* mit der Maske 0xFF00 angegeben haben, dass sie nur die oberen 8 Bit des Port beschreiben wollen, müssen die zu schreibenden Bits an der richtigen Position (15:8) sein, d.h. die zu schreibende Zahl muss um 8 Bit nach oben geschoben werden. Verwenden Sie einen *BusOut* müssen Sie nicht schieben.

Anhang: "C++ mit Mbed"-Vorlagen-Programm:

```
/******************** C-Programm ********************************************************
** Programmname:  Vorlage_C_mbed                                                       **
** Beschreibung:  Projektvorlage C++ mit mbed auf einem ARM Cortex-M µController      **
**                                                                                     **
** ___________________________________________________________________________________ **
**                                                                                     **
** Autor:                              Datum:  .  .2025                                **
*************************************************************************************/
#include "mbed.h"

/**** globale Konstantendeklaration ***********************************************/


/**** globale Variablendeklaration ************************************************/
/* Pinnnamen sind in mbed.h deklariert
DigitalOut  led(LED1);        // LED1 in PinNames.h  vordeklariert  (=  PA_5)
DigitalIn   button(BUTTON1);  // BUTTON ebenso wie LED  (=  PC_13)
InterruptIn taster(PB_0);   // Portpin PB_0 als Interrupteingang taster

// Konstantendeklaration der Portnamen wie "PortC" sind in mbed.h definiert.
PortInOut   port1(PortC,0x00FF);    // GPIOC_L, die "Maske" definiert die 8 unteren Bits
PortInOut   port2(PortC,0xFF00);    // GPIOC_H (port2 = 42 << 8; // schreibt eine 42)
PortInOut   port3(PortB,0x00FF);    // GPIOB_L

BusInOut  p1( PC_0,  PC_1, PC_2,  PC_3,  PC_4,  PC_5,  PC_6,  PC_7  );  // GPIOC_L
//                        |--- SPI 3 -------|
//                        |SCK    MISO  MOSI|
BusInOut  p2( PC_8,  PC_9, PC_10, PC_11, PC_12, PC_13, PC_14, PC_15 );  // GPIOC_H
//                              |--- SPI 3 -------|
//                              |SCK    MISO  MOSI|
BusInOut  p3( PB_0,  PB_1, PB_2,  PB_3,  PB_4,  PB_5,  PB_6,  PB_7  );  // GPIOB_L

//            | Serial 1|  |--I2C 1--|   | --------- SPI 1 -------|
//            | RxD  TxD|  |SDA  SCL |   | CS   |MISO   SCK   MOSI|
BusInOut  p4( PA_10, PA_9, PB_9,  PB_8,  PA_15, PA_6,  PA_5,  PA_7  );  // GPIO-Mixed
BusInOut  p5( PB_7,  PB_6, PB_9,  PB_8,  PA_15, PA_6,  PA_5,  PA_7  );  // GPIO-Mixed
//            | Serial 6|                 | --------- SPI 2 -------|
//            | TxD  RxD|                 | CS   |MISO   SCK   MOSI|
BusInOut  p6( PC_6,  PC_7, PB_9,  PB_8,  PB_12, PB_14, PB_13, PB_15 );  // GPIO-Mixed
*/

/**** Funktionen ******************************************************************/


/**** Hauptprogramm ***************************************************************/

int main() {
    // Initialisierungen


    // Endlosschleife
    while (1) {               //  alles ungleich 0 ist "true"

    }
}
```

Anhang: portable Installation von VS Code z.B. auf einem USB-Memory-Stick

Wie auf der Seite https://code.visualstudio.com/docs/editor/portable beschrieben, kann Visual Studio Code kann auch portabel installiert werden, so dass alle Plugins und Einstellungen unterhalb des Installationsordners gespeichert werden. Der Installationsordner darf damit keine Zugriffsbeschränkungen haben, d.h. der normale Programme-Ordner geht nicht.

Anleitung: Erstellen Sie einen neuen Ordner auf ihrem USB-Stick oder Computer, z.B. *VS_Code_portable*. Laden Sie anschließend von der Download-Seite https://code.visualstudio.com/download die zip-Version für x86 herunter und entpacken Sie in Ihrem eben erstellten Installationsverzeichnis. Damit diese Installation portabel wird, müssen Sie in ihrem Installationsordner ein Verzeichnis mit dem Namen *data* anlegen. Zum Ausführen des Programms klicken Sie doppelt auf *Code.exe*.