Assembler für ARM Cortex M-Prozessoren (STM32F411RE)

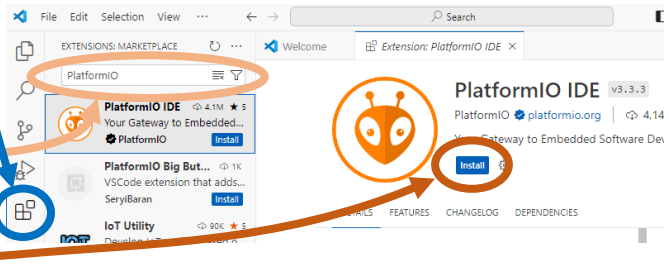
Aufgabe 1: Installation von Visual Studio Code mit Arm Tools (für zu Hause)

Als Entwicklungsumgebung (= **I**ntegrated **D**evelopment **E**nvironment = IDE) werden wir das Programm *Visual Studio Code* mit Plugins für die GNU Open Source Entwicklungswerkzeuge der ARM-Prozessoren verwenden. Diese Installationsanleitung ist für zu Hause, in der Schule ist alles bereits installiert.

1.1. Laden Sie sich den Installer für Visual Studio 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“.

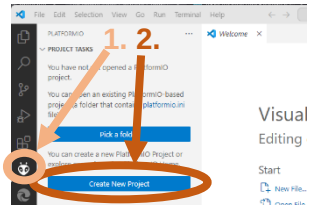
1.3. Installieren Sie das „PlatformIO“-Plugin[2].



Aufgabe 2: Anlegen eines Assembler-Projektes

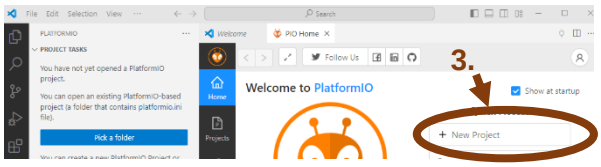
2.1. Klicken Sie links in der Modusleiste das PlatformIO-Symbol an und wählen Sie „Create New Project“ in der PlatformIO-Spalte.

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.

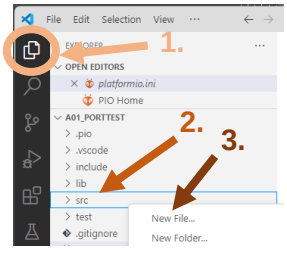
- Name: `A01_Porttest`
- Board: `ST Nucleo F411RE`
- Framework[3]: `CMSIS`
- Use default location: kein Häkchen. Wählen bzw. erzeugen Sie Ihr Verzeichnis für µController-Projekte, z.B. *H:\TGI11_IFTP\02_uController*



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 der Dateien in dem Projektverzeichnis vertrauen, setzen Sie das Häkchen, dass sie den Autoren aller Dateien im übergeordneten Hauptprojektverzeichnis vertrauen und bestätigen Sie das „Vertrauen“.

2.5. Erzeugen Sie eine Assembler-Datei *porttest.s* im Unterordner *src*[4] des Projektes. Klicken Sie dazu mit der rechten Maustaste auf das Verzeichnis und wählen Sie „New File …“ aus.



2.6. Kopieren Sie den Quellcode von der nächsten Seite in die Assembler Datei und speichern Sie diese mit *Datei* > *Save* oder Strg-S.

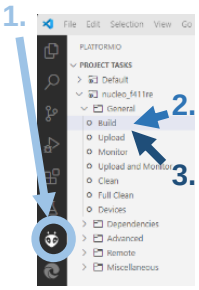
Bei einem anderen Projekt als den „Porttest“ können Sie auch den Quellcode der ausgeteilten Vorlagendatei in die Datei kopieren oder im Dateimanager des Betriebssystems die Vorlagendatei in das *src*-Verzeichnis des Projektes kopieren und umbenennen.

[1] extension = engl. für „Erweiterung“
[2] „PlatformIO Core“ (command line interface außerhalb VS Code) braucht nicht installiert zu werden.
[3] ab Klasse 12 werden wir das Betriebssystem MBED (und C++) verwenden, für Assembler nehmen wir nur die Basics.
[4] „src“ ist die Abkürzung für engl. „source“ = „Quelle“, gemeint ist der Quellcode des Programms.

Porttest-Programm, das von PortB[7:0] liest und auf PortC[7:0] und PortC[15:8] schreibt:

```
.syntax unified         // es wird die neueste Syntax (= Regelsystem für die Darstellung von Befehlen) verwendet

// ------------- Konstantendeklaration ----------------------------------------------------------

// Portdeklarationen - Jeder Port hat 12 Konfigurationsregister -> Offsetadressen
GPIOA = 0x40020000          // Basisadresse Port A (16 Bit)
GPIOB = 0x40020400          // Basisadresse Port B (16 Bit)
GPIOC = 0x40020800          // Basisadresse Port C (16 Bit)
GPIOD = 0x40020C00          // Basisadresse Port D (16 Bit)
GPIOE = 0x40021000          // Basisadresse Port E (16 Bit)
GPIOH = 0x40021C00          // Basisadresse Port H (16 Bit)

// Port-Offsetadressen - Adressen für je 4 Byte -> Konfigurationsregister
MODER   = 0x00   // Modifikationsregister - 2 Bit/Portpin {input  // output // alternate Funktion // analog mod}
OTYPER  = 0x04   // Out-Put-Type-Register nur untere 16 Bit (1 Bit/Portpin) {push-pull // open-drain}
OSPEEDR = 0x08   // Out-Put-Speed-Register - 2 Bit/Portpin {low // medium // fast // high } speed
PUPDR   = 0x0C   // Pull-Up-Down-Register - 2 Bit/Portpin {no pull-up/down // pull-up // pull-down // reserved}
IDR     = 0x10   // Input-Data-Register nur untere 16 Bit - read-only
ODR     = 0x14   // Output-Data-Register nur untere 16 Bit - read and write
BSRR    = 0x18   // Bit-Set/Reset-Register
LCKR    = 0x1C   // configuration Lock Register
AFRL    = 0x20   // Alternate Funktion Register Low
AFRH    = 0x24   // Alternate Funktion Register High

RCC = 0x40023800        // Reset and Clock-Control-Register
RCC_AHB1ENR = 0x30      // peripheral clock enable register -> Freigabe der Ports

// ------------- Code Section -------------------------------------------------------------

.section .text          // Bereich für Instruktionen (.data = Bereich für initialisierte Daten, .bss = uninitialisierte)
.global main            // main kann von anderem File verwendet werden (von startup_stm32f411xe.s)

// ------------- Hauptprogramm ------------------------------------------------------------
main:
    ldr     R1,=RCC                 // R1 = Adresse von rcc (siehe equ-Anweisung)
    ldr     R2,[r1,RCC_AHB1ENR]     // R2 = Inhalt der Adrese[rcc+offset AVB1ENR]
    orr     R2,0x7                  // setze die unteren 3 Bit von R2 auf ‚1‘
    str     R2,[r1,RCC_AHB1ENR]     // Speicherzelle[RCC+offset AVB1ENR] = R2

    // GPIOB ---- Input----------------------------
    ldr     R1,=GPIOB           // R1 <- Basisadresse von Port B
    mov     R2,0x0000           // MODER_von_B = 00_00_00_00_00_00_00_00
    strh    R2,[R1,MODER]       // Low Byte Port B als input (00)
    mov     R2,0x0000           // PUPDR_von_B = 00_00_00_00_00_00_00_00
    strh    R2,[R1,PUPDR]       // Low Byte Port B no Pullup-Pulldown

    // GPIOC ---- Output----------------------------
    ldr     R1,=GPIOC           // R1 = Basisadresse von Port C
    ldr     R2,=0x55555555      // MODER_von_C = 01_01_01_01_01_01_01_01_01_01_01_01_01_01_01_01
    str     R2,[R1,MODER]       // Port C ist Output (10)
    mov     R2,0x0000           // OTYPER_von_C = 0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0
    strh    R2,[R1,OTYPER]      // Port C ist push-pull-Ausgang
loop:
    ldr     R1,=GPIOB           // R1 = Basisadresse von Port B
    ldrb    R2,[R1,IDR]         // R2 = Input Byte von GPIOB

    mov     R3,0                // R3 = 0
    mov     R3,R2, LSL 8        // Lowbyte R2 in Highbyte R3
    eor     R3,0xFF00           // Highbyte R3 invertieren
    orr     R3,R2               // R3 = R3 | R2

    ldr     R1,=GPIOC           // R1 = Basisadresse von Port C
    strh    R3,[R1,ODR]         // Ausgabe Inhalt von R3 auf GPIOC

    b loop                      // Springe (“branch”) zum Label “loop” = Endlosschleife
```

2.7. *Kompilieren*

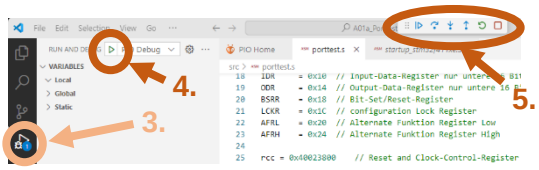


Wählen Sie im PlatformIO-Menü (1.) *Build* (2.) oder drücken Sie Strg-Alt-B.

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

2.8. *Download bzw. Upload in die Hardware*

Laden Sie mit *Upload* (3.) das Programm in den angeschlossenen µController.



Alternativ können Sie auch den *Debugger* verwenden. In seine Ansicht gelangen Sie durch Auswahl im PlatformIO-Menü. 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.

2.9. *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]

Schließen Sie dann die LEDs an PC[15:8] (P2-obere Reihe Mitte) an. Was stellen Sie fest?

2.10. *Analyse*

Analysieren Sie die verwendeten Instruktionen mit Hilfe der Assembler-Formelsammlung.

2.11. *Schließen des Projektes*

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