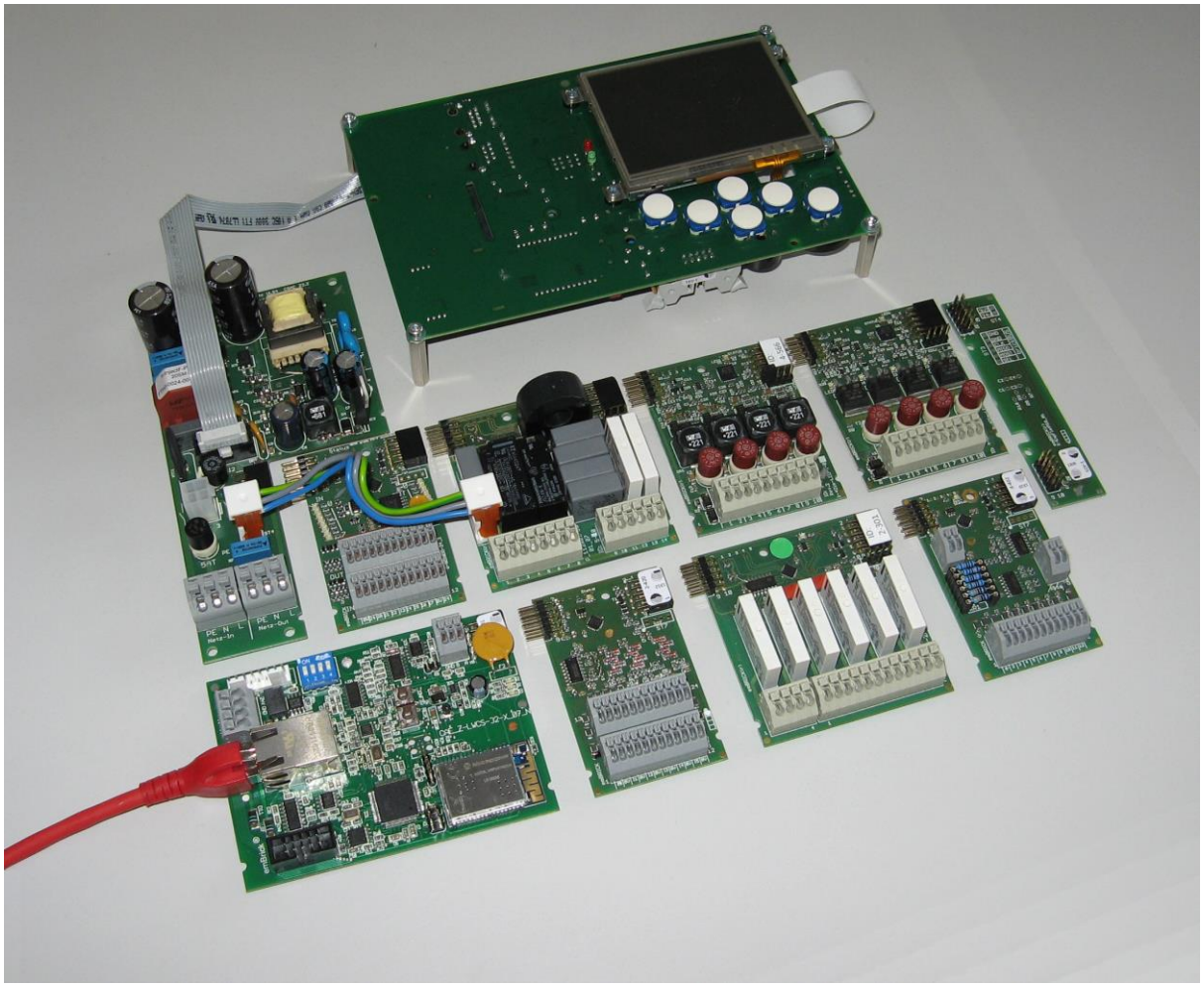
emBRICK

emBRICK® - EPC

Embedded Patch-board Controller



Benutzerhandbuch Ardu-inoBrickSystem Manual

emBRICK® is an open-source project, developed and supported by

IMACS GmbH
Alfred-Nobel-Straße 2
D – 55411 Bingen am Rhein
www.imacs-gmbh.com
www.embrick.de

support@embrick.de
Hotline: +49 (0) 7154 80 83 - 15

You can live for yourself today
or help build tomorrow - for everyone
(Ronan Harris)

Inhaltsverzeichnis

1. Starten mit dem Arduino

1.1 Entwicklungsumgebungen

Zur Softwareentwicklung des Arduinos werden von dem Hersteller folgende Möglichkeiten angeboten:

- Arduino IDE für folgende Betriebssysteme:
  - Windows, ab XP
  - Mac OS X, ab 10.7
  - Linux 32 und 64 Bit
  - Linux ARM
- Arduino Web IDE
- Arduino App für Win10

Die nachfolgenden Links führen direkt zu der Downloadseite bzw. zu dem Online Compiler

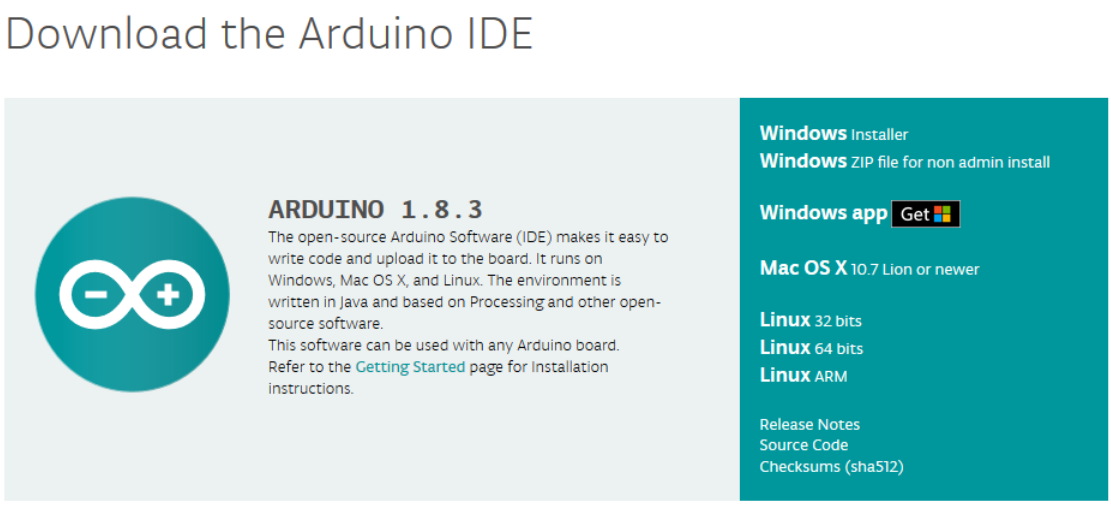
- Arduino IDE: https://www.arduino.cc/en/Main/Software/
- Arduino Web: https://create.arduino.cc/

Beide Versionen können kostenfrei genutzt und in einer beliebigen Sprache benutzt werden.
Da ständig an der IDE weiterentwickelt wird, ist auch abzusehen, dass der Editor auch auf weiteren Betriebssystemen zur Verfügung stehen wird.

1.2 Installation der Arduino IDE

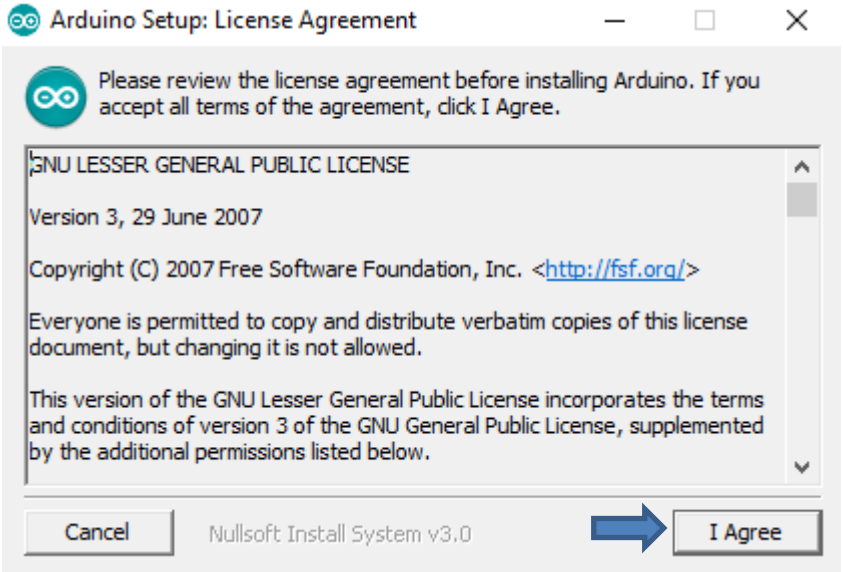
Um die Arduino IDE zu installieren, gehen Sie auf den oben genannten Link und laden sich dort den für Ihr System passenden Installer herunter. Im Folgenden wird der Weg für Windows 10 beschrieben:

Schritt 1:

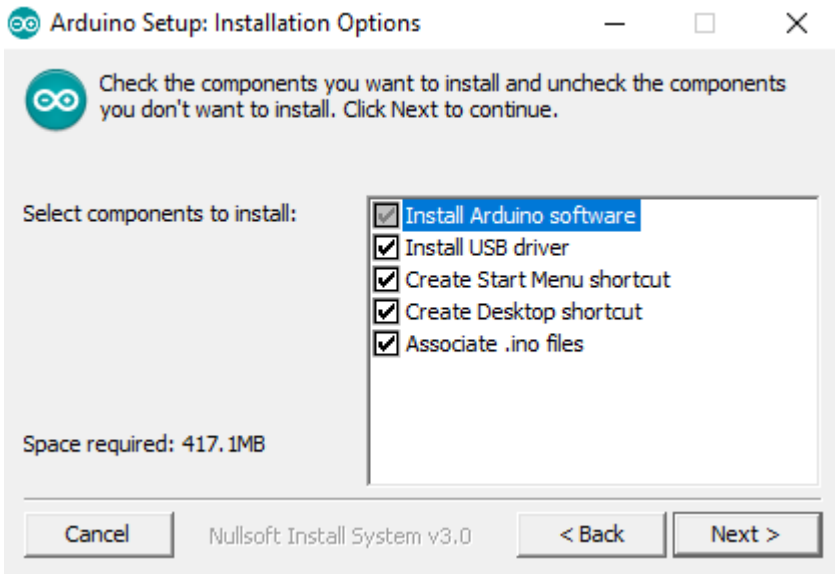


Schritt 2: Installation der Software

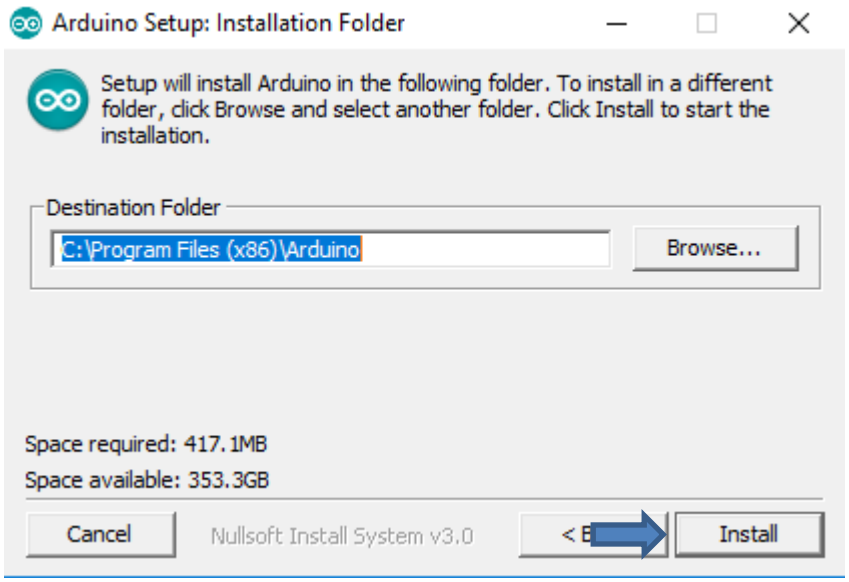
Ist der Download abgeschlossen kann die Installationsdatei aufgerufen werden.
Die Installationsdatei führt Sie durch die einfach aufgebaute Menüführung der Installation.



Nun müssen Sie die Nutzungsvereinbarung akzeptieren. Auch wenn die Nutzung der Software kostenfrei ist, wird ein genaues Durchlesen der Bedingungen nahegelegt.
Mit “I Agree” bestätigen Sie die Nutzungsbedingungen und gelangen anschließend zum nächsten Fenster:



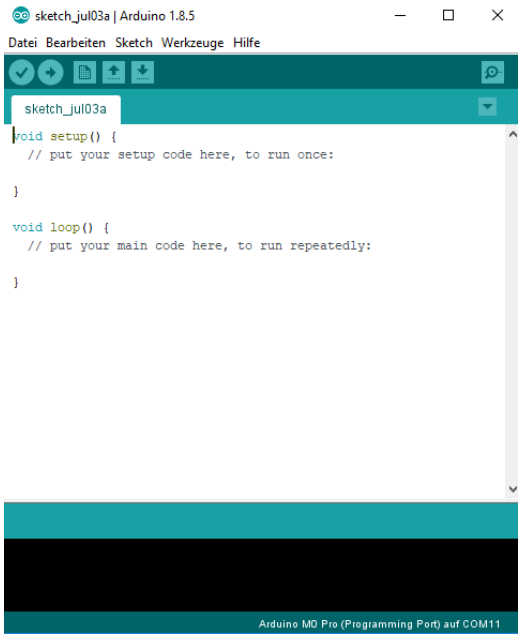
Hier entscheiden Sie, welche Komponenten der Software installiert werden. Wenn Sie bisher noch nichts mit einem Arduino gemacht haben, empfiehlt es sich, wie im obigen Bild, alles zu installieren, denn so bekommen Sie auch den passenden Treiber für den Arduino installiert.



Anschließend legen Sie noch den Pfad an, wo die Arduino IDE installiert werden soll. Standardmäßig wird der Pfad „C:\Program Files (x86)\Arduino“ unter Windows gewählt.
Wenn Sie den gewünschten Pfad gewählt haben, klicken Sie auf “Install” und starten somit die Installation der Software. Nach der Installation können Sie die Software über die Verlinkung auf dem Desktop oder durch das Startmenü öffnen.
Nun sollte auch der Arduino mithilfe eines USB-Kabels an den PC angeschlossen werden.

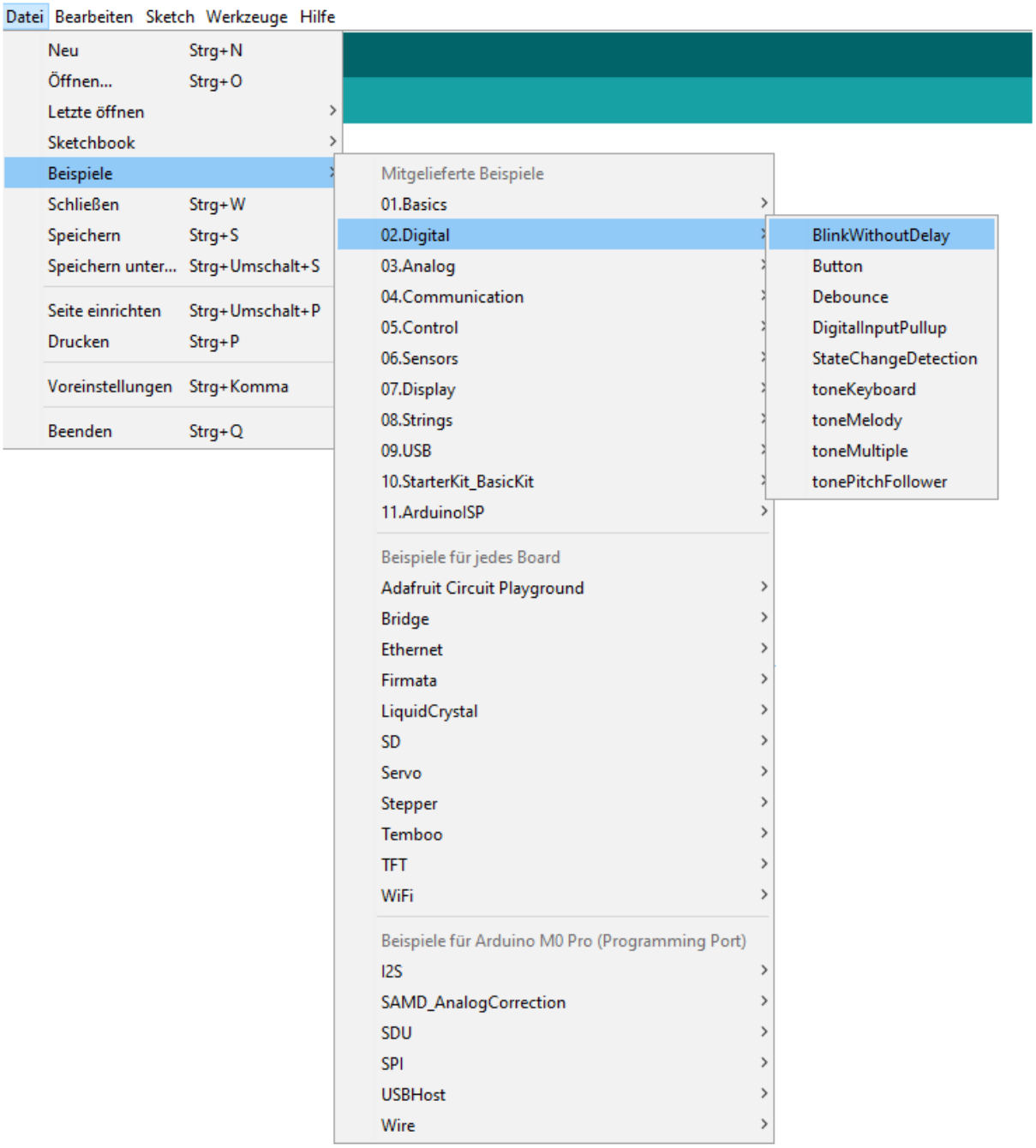
1.3 Starten der Arduino IDE und das erste Projekt

Wenn Sie nach der Installation die Arduino IDE starten, gelangen Sie zu folgendem Fenster:



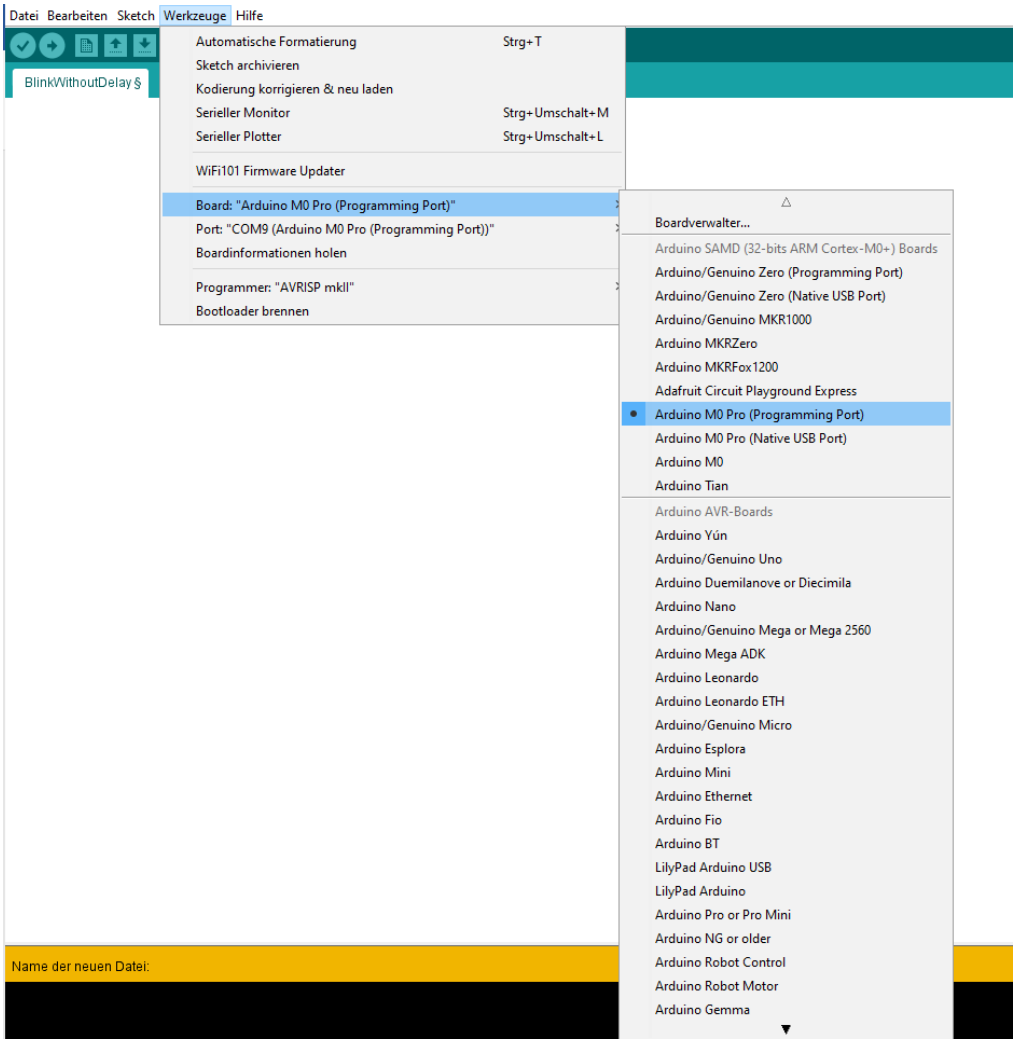
Dieses Fenster öffnet sich immer, sobald Sie ein neues Projekt starten.
Um nun ein erstes Programm auf den Arduino zu flashen, öffnen Sie aus den mitgelieferten Beispielen, den „Blink“ Sketch, welcher die LED auf dem Arduino in einem definierten Takt blinken lässt.

Um das Beispiel zu öffnen, klicken Sie auf “Datei”. Dann “Beispiele“ auswählen und im neuen Fenster die Kategorie „02.Digital“ öffnen. In dieser „BlinkWhithoutDelay” anklicken.
Eine bildliche Darstellung des Vorgangs finden Sie auf der nächsten Seite.



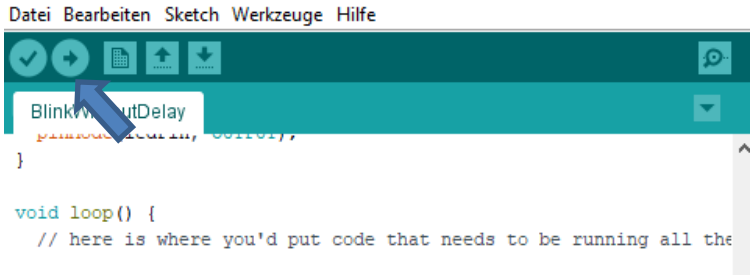
Anschließend öffnet sich ein neuer Editor, mit dem fertigen Code, welchen Sie nun auf das Gerät flashen können.

Als nächstes müssen Sie den über USB angeschlossenen Arduino in der Software auswählen und den zugewiesenen COM Port einstellen. Dies wird in dem Menüpunkt “Werkzeuge“ und dann „Board“ und „Port“ erledigt.
Folgend ein Beispiel Anhand eines Arduino M0 Pro:



Wir nutzen für unsere Projekte den Arduino M0 Pro, dementsprechend wurde im obigen Bild als Board der „Arduino M0 Pro (Programming Port)“ ausgewählt.
Der Arduino M0 Pro verfügt über zwei USB Anschlüsse und der Programming Port wird zum flashen benutz.
Da nun das Board festgelegt wurde muss nur noch der COM Port des angeschlossenen Arduinos ausgewählt werden. Bei uns wurde dem Arduino M0 Pro der COM Port „COM9“ vom System zugewiesen. Daher wurde hier COM9 gewählt. Dies kann aber von System zu System abweichen.

Nachdem Sie Ihren Arduino ausgewählt haben, können Sie den Sketch auf das Gerät hochladen. Dazu nutzt man folgendes Symbol in der IDE:



Nun wird der Sketch erst von dem Compiler geprüft und anschließend auf den Arduino hochgeladen. Nach erfolgreichem Hochladen, sollte die LED auf dem Arduino anfangen zu blinken.

2. Menüführung der Arduino IDE

In diesem Kapitel wird etwas näher auf die vorliegenden Dropdownmenüs eingegangen und die wichtigsten Funktionen erklärt.

2.1 “Datei”

Unter diesem Menüpunkt befinden sich allgemeine Werkzeuge, die fast bei jedem anderen Programm ebenfalls zur Verfügung stehen.
Hier können Sie

- Neue Projekte / Dateien erstellen
- Vorhandene Projekte / Dateien öffnen
- Beispiele öffnen
- Speichern
- Drucken
- Und den aktuellen Sketch bzw. Das komplette Programm beenden

Ein Projekt erkennt man bei der Arduino IDE durch die Endung .ino. Weitere Dateitypen wie .c, .cpp und .h können über includes in jedes Projekt eingebunden werden.

2.2 “Bearbeiten“

Unter Bearbeiten finden Sie, wie der Name auch schon sagt, nützliche Kommandos zur Bearbeitung des Sketches. Sei es Ausschneiden oder Kopieren, die dort aufgelisteten Befehle sind selbsterklärend.

2.3 “Sketch”

Interessanter wird es in diesem Menüpunkt. Hier finden Sie alle nützlichen Werkzeuge rund um den Sketch.

- Überprüfen / Kompilieren: Der Sketch wird auf Fehler geprüft, aber nicht auf das Gerät hochgeladen
- Hochladen: Der Sketch wird, wenn keine Fehler vorhanden sind, auf das Gerät hochgeladen
- Hochladen mit Programmer: wird ein externer Programmer verwendet, kann man über diesen Befehl den Sketch auf das Gerät laden
- Binärdatei exportieren: exportiert den Code in eine Binärdatei
- Sketch Ordner anzeigen: ruft den Odner auf, in der das Projekt gespeichert wurde
- Bibliothek einbinden: bindet eine aus der Liste ausgewählte Bibliothek in das Projekt ein
- Datei hinzufügen: fügt eine Datei in das aktuelle Projekt ein

2.4 “Werkzeuge”

Folgende Werkzeuge stehen zur Verfügung:

- Automatische Formatierung: Formatiert den Code automatisch
- Sketch archivieren: archiviert das Projekt
- Serieller Monitor: Öffnet ein Fenster, mit dem man mit dem Arduino kommunizieren kann. Hier werden Serielle Plots dargestellt
- Serieller Plotter: gibt die Ausgabe des Arduinos als fortlaufenden Graf aus
- WiFi101 Firmware Updater: hier kann man die Firmware des Arduinos updaten
- Board: hier wird das Arduino Board ausgewählt. Wenn es nicht auf der Liste zu finden ist, gehen Sie auf Boardverwalter und suchen Sich das passende Paket für Ihre Hardware.
- COM-Port: hier wird der COM-Port des angeschlossenen Arduinos ausgewählt
- Boardinformationen holen: gibt Ihnen Infos zu dem angeschlossenen Board zurück
- Programmer: stellt den Programmer des Boards ein
- Bootloader Brennen: spielt einen Bootloader auf neue Controller

2.5 “Hilfe”

Unter dem Menüpunkt “Hilfe” finden Sie sämtliche Infos zum Arduino bzw. zu der Arduino IDE.

3. Starten mit emBRICK

3.1 Hardware

Um den Arduino mit dem emBRICK System verbinden zu können, benötigen Sie

- den ArduinoBRICK
- verschiedene Bricks (bis zu 32)
- die Arduino IDE oder einen beliebigen Editor, der die Programmierung des Arduinos unterstützt
- das ArduinoBRICK Software Starter Kit
- eine konstante 24V Spannungsversorgung
- ein passendes USB-Kabel zum Programmieren des Arduinos

In den nachfolgenden Darstellungen sehen Sie, wie das Starterkit aufgebaut ist, wo Sie den Arduino aufstecken, weitere Bricks anschließen, wie die Anschlüsse für das Beispielprogramm angeschlossen werden und wo Sie die Versongungsspannung anschließen müssen.

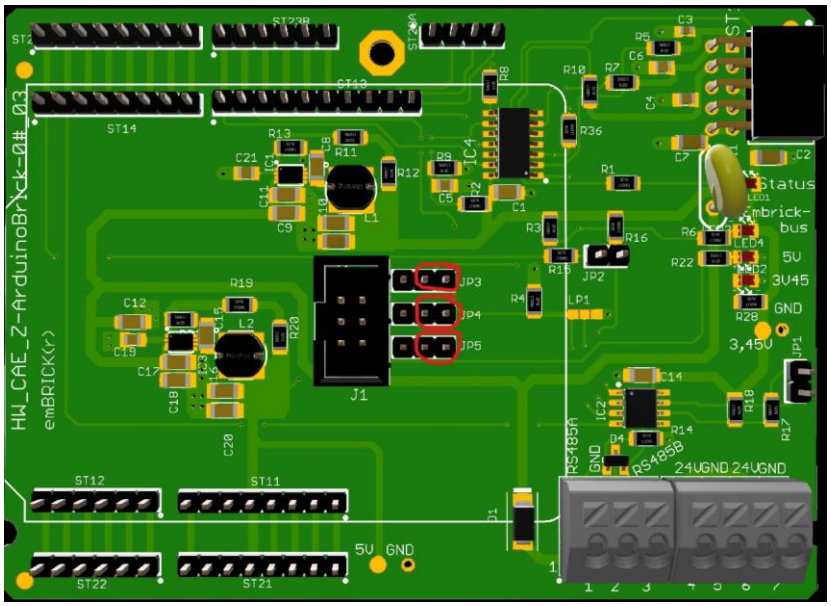
Bilder des Starterkits kommen, sobald der Brick produziert wird

3.2 Konfiguration des ArduinoBRICKs

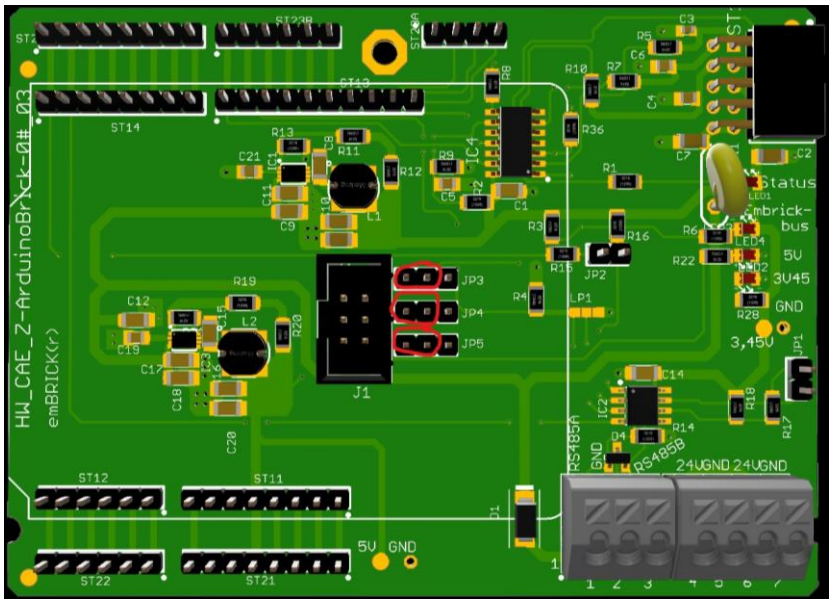
Haben Sie die oben genannten Teile, kann es weitergehen mit dem Setup des ArduinoBRICKs. Hierbei kommt es drauf an, welchen Arduino Sie verwenden möchten.
Nutzen Sie einen Arduino, bei dem die SPI Kommunikation über folgende Pins

- CS: 10
- MOSI: 11
- MISO: 12
- SCK: 13

stattfindet, dies ist zum Beispiel beim Arduino Uno der Fall, müssen Sie die Jumper JP3, JP4 und JP5, wie im folgenden Bild markiert, brücken.



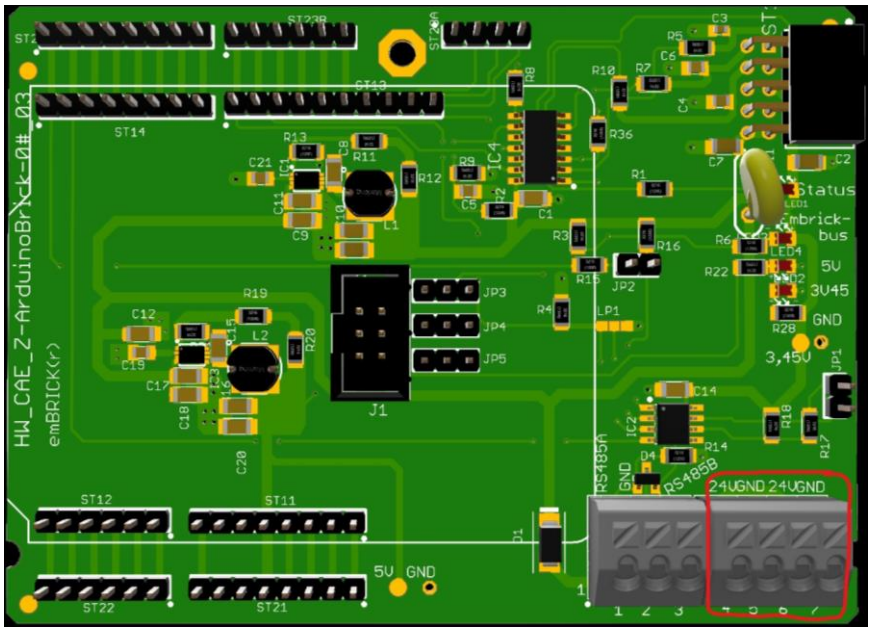
Haben Sie einen Arduino, bei dem die SPI Kommunikation über den ICSP Header läuft, werden die Jumper wie folgt gesetzt:



Zudem muss das mitgelieferte Flachbandkabel, von der ICSP Schnittstelle des Arduinos, mit dem J1 des ArduinoBRICKs verbunden werden.

Bei Verwendung eines solchen Arduinos können die Pins 11, 12 und 13 noch beliebig verwendet werden, welches bei dem Arduino Uno durch die notwendige Belegung für die SPI Schnittstelle nicht mehr möglich ist.

Nachdem Sie die Jumper passend gesetzt haben und bei Arduinos mit ICSP Nutzung das Flachbandkabel eingebaut haben, können Sie den Arduino auf den ArduinoBRICK stecken.
Als nächstes benötigen Sie eine 24V Spannungsversorgung, welche in folgende Klemmen eingesteckt wird:



Nun können auch die Bricks an die Buchsenleiste über der Status-LED eingesteckt werden.
Sind alle vorherigen Schritte abgearbeitet, kann es nun endlich mit dem Programmieren losgehen.

3.3 Starten des Beispielprogramms

Als nächstes kann das mitgelieferte Beispielprogramm mit der Arduino IDE geöffnet werden. Dazu entpacken Sie die .zip Datei und schauen Sie als erstes nach, ob folgende Dateien vorhanden sind:

- Appl.ino
- bBConfigs.h
- bBDefines.h
- bB_EasyAPI.cpp
- bB_EasyAPI.h
- bB_master.cpp
- brickbus.cpp
- brickbus.h

Wie man an den Symbolen der einzelnen Dateien im Explorer erkennen kann, ist die Appl.ino das Hauptprogramm und lässt sich direkt mit der Arduino IDE öffnen. Nun sollten in dem Editor zu der Appl.ino auch alle weiteren Dateien geöffnet sein. In der Appl.ino werden Sie Ihr Programm programmieren. Alle weiteren Dateien dienen zum einstellen der brickBUS Kommunikation und sollten auf keinen Fall geändert werden, da sonst die sehr speziellen Timings der Kommunikation nicht mehr stimmen könnten und somit keine Kommunikation mehr aufgebaut werden kann!

Um den Einstieg so einfach wie möglich zu gestalten, haben wir bereits ein laufendes Beispielprogramm in der Appl.ino aufgebaut. In diesem Beispiel werden vier verschiedene Bricks mit dem Arduino gesteuert und ausgelesen.
In der Funktion application() werden hier beispielhaft einige Befehle ausgeführt, um Informationen zu den einzelnen Bricks zu senden bzw. von den Bricks zu empfangen. Die Struktur sollte auch weitestgehend in Ihrem Projekt so übernommen werden, um keine Probleme mit der Kommunikation zu bekommen.
Der wichtigste Teil, den Sie auf jeden Fall beibehalten sollten, ist alles, was unter dem Kommentar „Aufruf der Wartezeiten abhängig von der Initialisierung des brickBUSes“, zu finden ist. Dort geht es um das zyklische Aufrufen der bB_Processing(), die alle 10ms alle angeschlossenen Bricks updaten soll. Wenn Sie Daten von Bricks auswerten möchten, sollten Sie auch die seriellen Ausgaben nur in dem vorgesehenen Bereich eintragen, da sonst die Zykluszeit stark beeinflusst wird.

Um mit den angeschlossenen Bricks zu kommunizieren, werden folgende Hauptfunktionen benutzt:

- zum setzen der Bricks gibt es drei Standartbefehle
  - bB_putBit()
  - bB_putByte()
  - bB_putWord()
- zum auswerten der Bricks gibt’s es folgende drei Standardbefehle
  - bB_getBit()
  - bB_getByte()
  - bB_getWord()

Welche Funktionen es noch gibt und wie die genannten Funktionen genau definiert sind bzw. welche Parameter übergeben werden müssen, werden im nachfolgenden Kapitel genauer beschrieben.

4. Nutzbare Funktionen des brickBUSes

4.1 bB_Start()

In dieser Funktion wird das SPI initialisiert und mit allen benötigten Parameter gestartet. Diese Funktion muss in der setup() einmalig aufgerufen werden. Ohne diesen Aufruf wird keine Kommunikation aufgebaut werden können.

4.2 bB_processing()

In der bB_processing() wird die Initialisierung des brickBUSes eingeleitet und alle angeschlossenen Bricks automatisch initialisiert. Sobald alle Bricks erkannt wurden, startet die normale Kommunikation zwischen Master (Arduino) und den Slaves (Bricks).
Wenn die Initialisierung erfolgreich ist, sollten die Status-LEDs der Bricks gleichmäßig in einem 0,8Hz Takt blinken.
Die bB_processing() muss auch zwingend alle 10ms aufgerufen werden. Ist dies nicht der Fall, geben die Bricks einen, wie unten beschriebenen Morse Code als Fehlermeldung zurück.

Sollten die Status-LEDs einen anderen Takt aufweisen, werden folgende Infos zum Status der Kommunikation übermittelt:

- 5Hz Takt: ..... nicht initialisiert
- Morse Code *-*: ..... keine laufende Kommunikation / Kommunikation ist zusam-
  ..... mengebrochen / >2s keine Nachricht gesendet
- Morse Code ***: ..... Slave ist initialisiert und die Kommunikation läuft, aber nicht
  ..... stabil (<20 Nachrichten / s)
- Morse Code **-: ..... Slave ist initialisiert und die Kommunikation läuft, aber es sind
  ..... zu viele Fehler / falsche Nachricht an Slave gesendet
- 0,8Hz Takt: ..... funktionierende Kommunikation -> Betriebsmodus
- Dauerhaft an / aus: ..... Defekt des Moduls oder viel zu viele Nachrichten werden
  ..... gesendet

4.3 bB_getNumModules(node)

Wird diese Funktion mit einem Knoten aufgerufen, wird die Anzahl der an diesem Knoten angeschlossenen Bricks zurückgegeben.
Als Knoten muss derzeit eine 1 angegeben werden.

4.4 bB_getModulID(node, slaveNo)

Diese Funktion gibt für einen bestimmten Knoten und der gewünschten Brick Position die ModulID des Bricks zurück.

4.5 bB_getModulSwVers(node, slaveNo)

Gibt die Modul Software Version eines Bricks an dem gewünschten Knoten zurück.

4.6 bB_getModulbBVers(node, slaveNo)

Gibt die Protokoll Version des gewünschten Bricks zurück.

4.7 bB_getModulStatus(node, slaveNo)

Gibt den Status des gewünschten Bricks zurück.

4.8 bB_getWord(node, slaveNo, bytePos)

Wird diese Funktion aufgerufen, wird von dem gewünschten Brick von einem beliebigen Byte ein Word zurückgegeben.
Hier ein Beispiel: am Knoten 1 wollen wir vom 2. Brick die Daten des 1. Bytes auslesen, so muss die Funktion folgendermaßen aufgerufen werden:

Input = bB_getWord(1,1,0);

Allgemein ist zu erwähnen, dass die Knoten (node) immer bei 1, die Slaves (slaveNo) ebenfalls bei 1 beginnen und die Position des Bytes (bytePos) und bei der put / getBit() ebenfalls die Position des Bits beide bei 0 beginnen.
Somit würde ein Aufruf der bB_getBit() Funktion mit den Parametern 1. Knoten, 3. Slave, 1. Byte, 5. Bit wie folgt aussehen:

Input = bB_getBit(1,3,0,4);

4.9 bB_getByte(node, slaveNo, bytePos)

Mit dieser Funktion wird ein Byte des gewünschten Bricks zurückgegeben.
Der Aufruf wird wie in Kapitel 4.8 beschrieben aufgebaut.

4.10 bB_getBit(node, slaveNo, bytePos, bitPos)

Mit dieser Funktion wird ein Bit des gewünschten Bricks zurückgegeben.
Der Aufruf wird wie in Kapitel 4.8 beschrieben aufgebaut.

4.11 bB_putWord(node, slaveNo, bytePos, value)

Mit dieser Funktion wird ein Word an den gewünschten Brick gesendet.

Ähnlich wie bei den get Funktionen sind auch die put Funktionen aufgebaut:
Node und slaveNo starteten immer bei 1, bytePos und bitPos beginnen bei 0 und value wird von Ihnen im passenden Format (word, byte oder bit) gesetzt.

Als Beispiel nehmen wir den 2. Brick des 1. Knoten und möchten von dem 1. Byte das 1. Und 3. Bit setzen. Hierzu bietet sich die Funktion bB_putByte() an und wird wie folgt aufgebaut:

bB_putByte(1,2,0,0x05);

4.12 bB_putByte(node, slaveNo, bytePos, value)

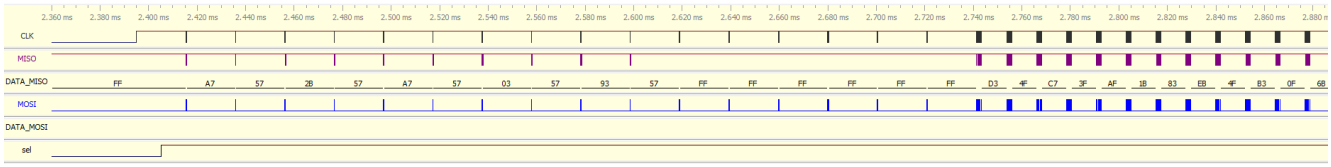
Mit dieser Funktion wird ein Byte des gewünschten Bricks gesetzt.
Der Aufruf wird wie in Kapitel 4.11 beschrieben aufgebaut.

4.13 bB_putBit(node, slaveNo, bytePos, bitPos, value)

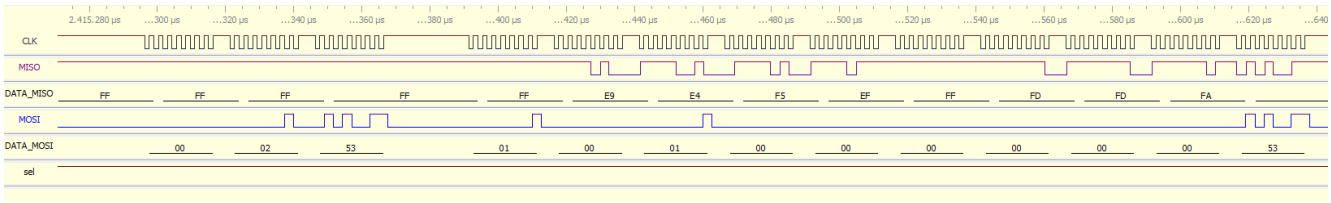
Mit dieser Funktion wird ein Bit des gewünschten Bricks gesetzt.
Der Aufruf wird wie in Kapitel 4.11 beschrieben aufgebaut.

5. Wie sieht eine korrekte Kommunikation aus?

In diesem Kapitel wird Ihnen gezeigt wie eine korrekte Kommunikation zwischen dem Arduino und den Slaves auszusehen hat.
So können Sie bei Ihrem Projekt leicht kontrollieren können, ob Sie alles richtig programmiert haben.



Hier sieht man den Startprozess des Arduinos. Als erstes wird der Chip Select aktiviert, um den ersten Brick anzusprechen. Anschließend wird die Initialisierung solange durchgeführt, bis alle Bricks erkannt wurden (hier bis 2.740ms). Ist die Initialisierung erfolgreich, geht es automatisch in die normale Brick Kommunikation über, welche alle 10ms geupdatet werden sollte.



So sollte ein Initialisierungsblock aussehen. Ganz wichtig sind die Pausen zwischen den einzelnen Blöcken. Die kleinen Pausen müssen exakt 5µs und die größere Pause 25µs betragen. Ist dies nicht der Fall, kann keine Kommunikation zwischen Master und Slaves aufgebaut werden.

Hier noch ein Beispiel, wie die fortlaufende Kommunikation aussehen sollte. Dargestellt ist ein Updateblock mit fünf Bricks. Man kann gut erkennen, dass es verschiedene Bricks sind, da die Länge der einzelnen Blöcke variiert. Wie groß die einzelnen Blöcke sein müssen, wird automatisch in der bB_processing() geregelt, somit müssen Sie lediglich die gewünschten Bricks an den Bus anschließen und die entsprechenden Bits / Bytes, die Sie benötigen, setzen oder abfragen.