M 3 - Arduino-Code Implementierung und die SimuBox

Teilthemen:

SimuBox Inbetriebnahme	**M 3.1**
Variablen Deklaration	**M 3.2**
Setup()	**M 3.3**
Loop()	**M 3.4**

Arbeitsaufträge für die Vorgehensweise

- Angeleitete Inbetriebnahme der SimuBox
- Textanalyse
- Codeanalyse

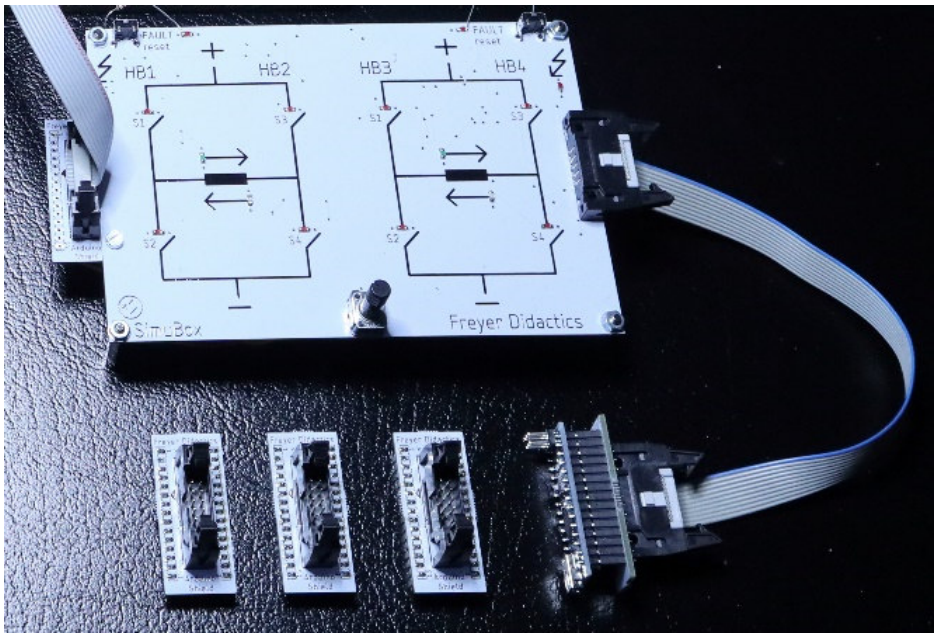
M 3.1 - SimuBox Inbetriebnahme

Es wird ein Arduino Nano verwendet, um die Steuerung der SimuBox zu programmieren. Dazu muss erst die IDE (Integrated Development Environment / Programmierungssoftware) von Arduino installiert werden. Diese findest du unter dem Link: https://www.arduino.cc/en/software

Achte darauf, dass Version 1.8.19 installiert wird!

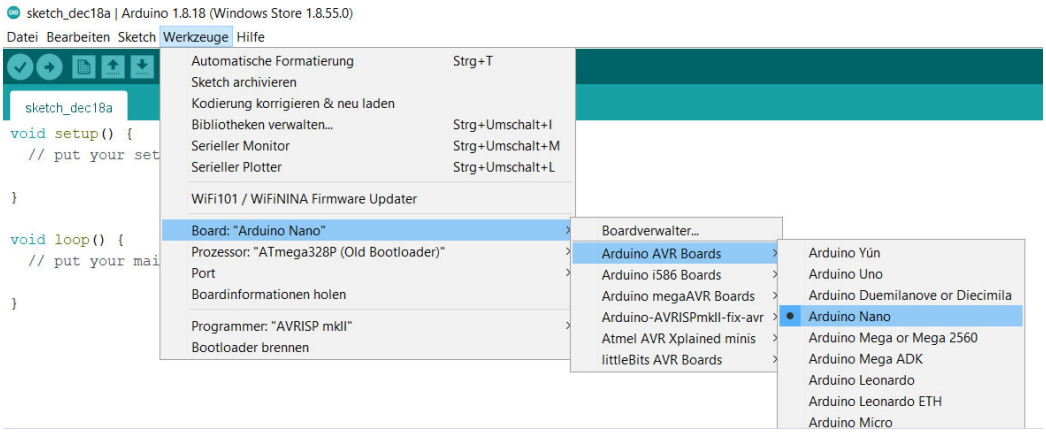
Aufgaben

1. **Schließe** den Arduino Nano, mit aufgesteckter Adapterplatine, über USB an deinen Laptop / PC an.
2. **Verbinde** den Arduino Nano mit der SimuBox mithilfe des Flachbandkabels

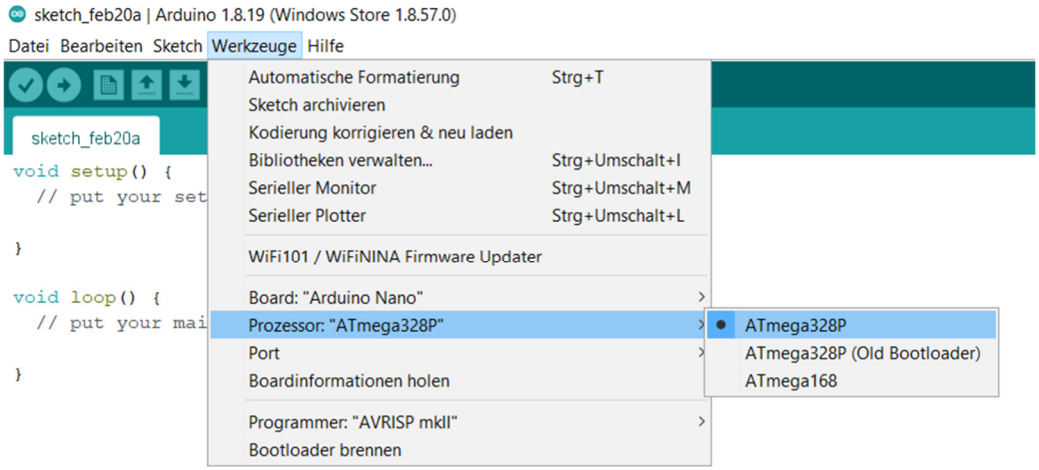


3. Richte die Arduino IDE ein. Dafür müssen folgende Einstellungen getroffen werden:

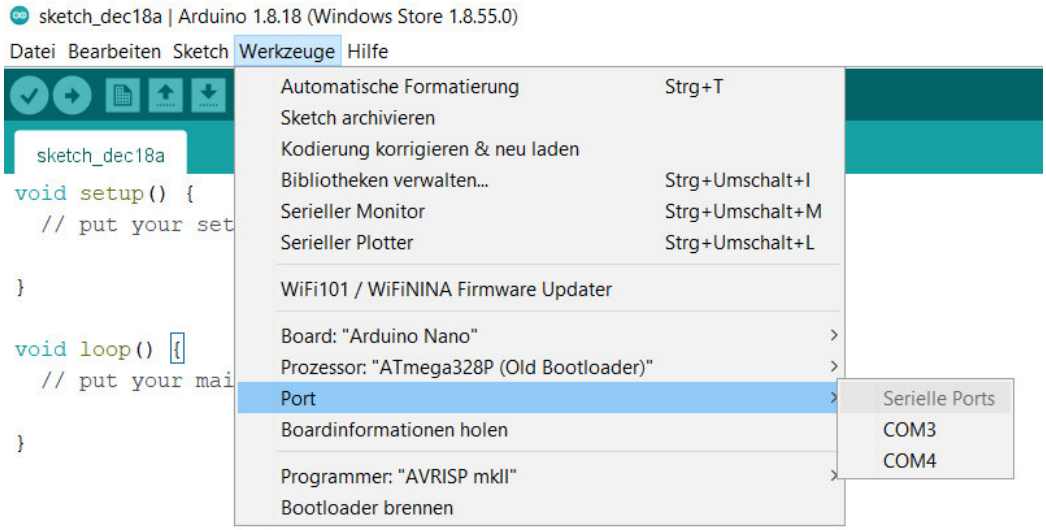
    - Als Board wird der Arduino Nano ausgewählt



- Als Prozessor wird der ATmega328 ausgewählt.



- Als Port wird der vom Laptop / PC verwendete Port ausgesucht, an dem der USB-Anschluss angeschlossen wurde. Dieser ist meist schon korrekt eingestellt, Im Falle das der nächste Schritt nicht funktioniert, kann dieser über den Gerätemanager des Rechners eingesehen werden.



M 3.2 - Variablen Deklarieren

Ein Arduino Programm (der gesamte Code) besteht aus drei Teilen:

1. **Variablen Deklaration**
2. **Setup()**
3. **Loop()**

Ein Beispiel der Syntax (die Schreibweise):

```
#define Led 5

Int poti = 0;
```

#define wird genutzt, um einem konstanten Wert einen Namen zu geben, bevor das Programm kompiliert wird.

Definiert die Variable „poti" als „integer", d. h. als ganze Zahl, mit dem (momentanen) Wert 0. Nach jeder Zeile folgt als Abschluss ein Semikolon ;

Innerhalb der Variablen Deklaration werden Werte hinterlegt, die das Programm im weiteren Verlauf verarbeitet.

Aufgabe 1: Öffne die Arduino IDE, speichere das Projekt an einem beliebigen Ort und kopiere die folgenden Deklarationen in dein Arduino Programm über das `void setup()`.

```
#define S1_S4_LEFT_BRIDGE_PIN 6
#define S2_S3_LEFT_BRIDGE_PIN 5
#define S1_S4_RIGHT_BRIDGE_PIN 9
#define S2_S3_RIGHT_BRIDGE_PIN 10
#define ADC_PIN A0
#define ADC_MAX (1<<10) - 1 // 2^n <=> 1<<n | n = ADC bits
#define F_MIN 1     // Minimum output frequency [Hz]
#define F_MAX 200   // Maximum output frequency [Hz]
const float scale = ((1000.0/F_MIN) - (1000.0/F_MAX)) / ((float) ADC_MAX);
unsigned long loop_time = 0;           // Contains last timestamp [ms]
unsigned int T_period = 1000 / F_MIN;  // Period duration of rect-signal [ms]
unsigned int T_quarter = T_period / 4; // Quarter of period duration [ms]
unsigned int quarter_step_count = 0;
// Gets increased every quarter period duration (to handle 90° phase-shifted second strand)
```

Aufgabe 2: Im folgenden Text sind Eigenschaften der Begriffe `int`; `long` und `bool` beschrieben. Die Größen `int` und `bool` sind abhängig von der Rechnerarchitektur, in diesem Fall für den Arduino. Markiere die Eigenschaften zu den einzelnen Ausdrücken und liste zu jedem Ausdruck zwei Eigenschaften auf:

Mit Integer wird in der Informatik ein Datentyp bezeichnet, der ganzzahlige Werte speichert. Auf dem Arduino Uno (und anderen ATmega-basierten Boards, z.B. NANO) speichert ein `int` einen 16-Bit-Wert (2 Byte). Dies ergibt einen Bereich von -32,768 bis 32,767 (Minimalwert -2^15 und Maximalwert (2^15) - 1). Long-Variablen sind Variablen mit erweiterter Größe für die Nummernspeicherung und speichern 32 Bit (4 Byte), von -2,147,483,648 bis 2,147,483,647. Ein `bool` enthält einen von zwei Werten, `true` oder `false`. Jede `bool`-Variable belegt 1 Byte Speicher.

`int`:

1) _____

2) _____

`long`:

1) _____

2) _____

`bool`:

1) _____

2) _____

M 3.3 - Setup()

Nach der Variablen Deklaration wird die `setup()` Funktion aufgerufen. Die `setup()` Funktion wird jedes Mal aufgerufen, wenn der Sketch startet. Sie soll benutzt werden, um Variablen, Pinmodi, Bibliotheken, usw. zu initialisieren. Die Funktion wird nur ein einziges Mal aufgerufen, jedes Mal, wenn das Board gestartet oder resettet wird.

Aufgabe 1: Kopiere die `setup()` Funktion unter die Variablen Deklaration und ersetzte damit die automatisch erzeugte `setup()` Funktion in deinen Arduino Code.

```
void setup() {
  pinMode(S1_S4_LEFT_BRIDGE_PIN, OUTPUT);   // Set pin as output
  pinMode(S2_S3_LEFT_BRIDGE_PIN, OUTPUT);
  pinMode(S1_S4_RIGHT_BRIDGE_PIN, OUTPUT);
  pinMode(S2_S3_RIGHT_BRIDGE_PIN, OUTPUT);
}
```

Aufgabe 2: Im folgenden Text sind Eigenschaften der Begriffe `pinMode(), digitalWrite()` und `analogRead()` beschrieben. Markiere die Eigenschaften zu den einzelnen Ausdrücken im Text und liste zu jedem Ausdruck zwei Eigenschaften auf:

Der Befehl `pinMode(Pin, Modus)` deklariert einen digitalen Kanal auf dem Arduino-Board entweder als Eingang (INPUT) oder Ausgang (OUTPUT). Als Informationen werden Pin (Kanal) und die Funktion (INPUT oder OUTPUT) in die Klammer gesetzt.

```
pinMode(3,OUTPUT); // setzt den digitalen Kanal 3 als Ausgang
```

Digitale Pins geben nur zwei Werte aus, entweder eine 1, also ein HIGH (5V) oder eine 0, ein Low (0V). Der Befehl `digitalWrite(Pin, Wert)` schreibt einen HIGH- oder LOW-Wert an einen digitalen Pin. Wenn der Pin mit `pinMode()` als OUTPUT konfiguriert wurde, wird seine Spannung mit HIGH auf die Betriebsspannung des verwendeten Boards (5V) oder mit LOW auf 0V (bzw. GND) gesetzt.

```
digitalWrite(13, HIGH); // setzt den digitalen Pin 13 auf high
```

Analoge Pins verarbeiten den Spannungsbereich von 0 bis 5V. Der Befehl `analogRead(Pin)` liest den Wert vom angegebenen analogen Pin ein. Die Arduino-Boards enthalten einen 10-Bit-Analog-zu-Digital-Konverter. D.h. das Board mappt Eingangsspannungen zwischen 0 und 5 V auf Integer-Werte zwischen 0 und 1023. Die erreichte Auflösung ist damit z.B. auf einem Arduino NANO 5 V / 1024 Teile oder 0,0049 V (4,9 mV) per Teil.

```
poti = analogRead(potiPin); // Lesen des Eingangspins potiPin und Zuordnung auf die Variablen poti
```

`pinMode`:

1) _____

2) _____

`digitalWrite`:

1) _____

2) _____

`analogRead`:

1) _____

2) _____

M 3.4 - Loop()

Nach dem Erstellen einer `setup()` Funktion, die die Anfangswerte (Variablen, Pins und Bibliotheken) initialisiert, macht die Funktion `loop()` genau das, was der Name andeutet. Sie ist eine Endlosschleife, die nach jedem Durchlauf erneut aufgerufen wird. Dadurch kann dein Programm Variablen verändern, Daten lesen oder darauf reagieren. Verwende diese Option, um das Arduino-Board aktiv zu steuern.

Aufgabe 1: Kopiere die folgende `loop()` Funktion unter die `setup()` Funktion im Arduino Programm und ersetzte damit die automatisch erzeugte `loop()` Funktion.

```
void loop() {
unsigned long delta_T = millis() - loop_time;


 if (delta_T >= T_quarter) // One quarter of a whole period has passed
  {
    switch (quarter_step_count)
    {
      case 0: // New period started
        T_period = (1000/F_MAX) + scale * analogRead(ADC_PIN);
        T_quarter = T_period / 4; // Update quarter period duration
 // Set outputs:
        digitalWrite(S2_S3_LEFT_BRIDGE_PIN, LOW);
        digitalWrite(S1_S4_LEFT_BRIDGE_PIN, HIGH);
        break;
      case 1: // First quarter passed
        digitalWrite(S2_S3_RIGHT_BRIDGE_PIN, LOW);
        digitalWrite(S1_S4_RIGHT_BRIDGE_PIN, HIGH);
        break;
      case 2: // Second quarter passed
        digitalWrite(S1_S4_LEFT_BRIDGE_PIN, LOW);
        digitalWrite(S2_S3_LEFT_BRIDGE_PIN, HIGH);
        break;
      case 3: // Third quarter passed
        digitalWrite(S1_S4_RIGHT_BRIDGE_PIN, LOW);
        digitalWrite(S2_S3_RIGHT_BRIDGE_PIN, HIGH);
        break;
    }
    loop_time = millis(); // Update looptime
    quarter_step_count ++; // Increase period counter
    if (quarter_step_count > 3) {
    quarter_step_count = 0;
    }
  }
}
```

Das `if`-Statement überprüft, ob eine Bedingung `true` ist. Ist dies der Fall, wird der Code innerhalb der geschweiften Klammern ausgeführt. Ist die Bedingung `false` wird das `if`-Statement übersprungen. Die Bedingung wird innerhalb der Klammer () nach dem `If` abgefragt. Der auszuführende Code befindet sich innerhalb der geschweiften Klammern `{}`.

Syntax:

```
if (Bedingung) {
 //statement(s)
}
```

Aufgabe 2: Aus wie vielen `if`-Statements besteht die `loop()` Funktion aus Aufgabe 1?

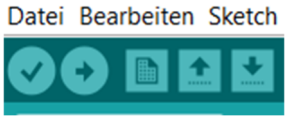
_____

Aufgaben 3: Markiere und kennzeichne im folgenden `If`-Statement den Bereich der Bedingung und den des Statements (ausführender Code).

```
if (quarter_step_count > 3) {
quarter_step_count = 0;
}
```

Aufgaben 4: Überprüfe den gesamten Code in der Ardunio IDE und lade ihn danach auf den Arduino NANO.

Zum Überprüfen klicke auf das Häkchen, erscheint kein Fehler ist der Code in Ordnung. Um den Code auf das Board zu laden, klicke auf den Pfeil nach rechts.



Nach dem erfolgreichen Upload geben die LEDs der SimuBox den Steuerungszyklus der H-Brückenschaltung wieder. Dieser ist mithilfe des Potentiometers in der Frequenz manipulierbar.