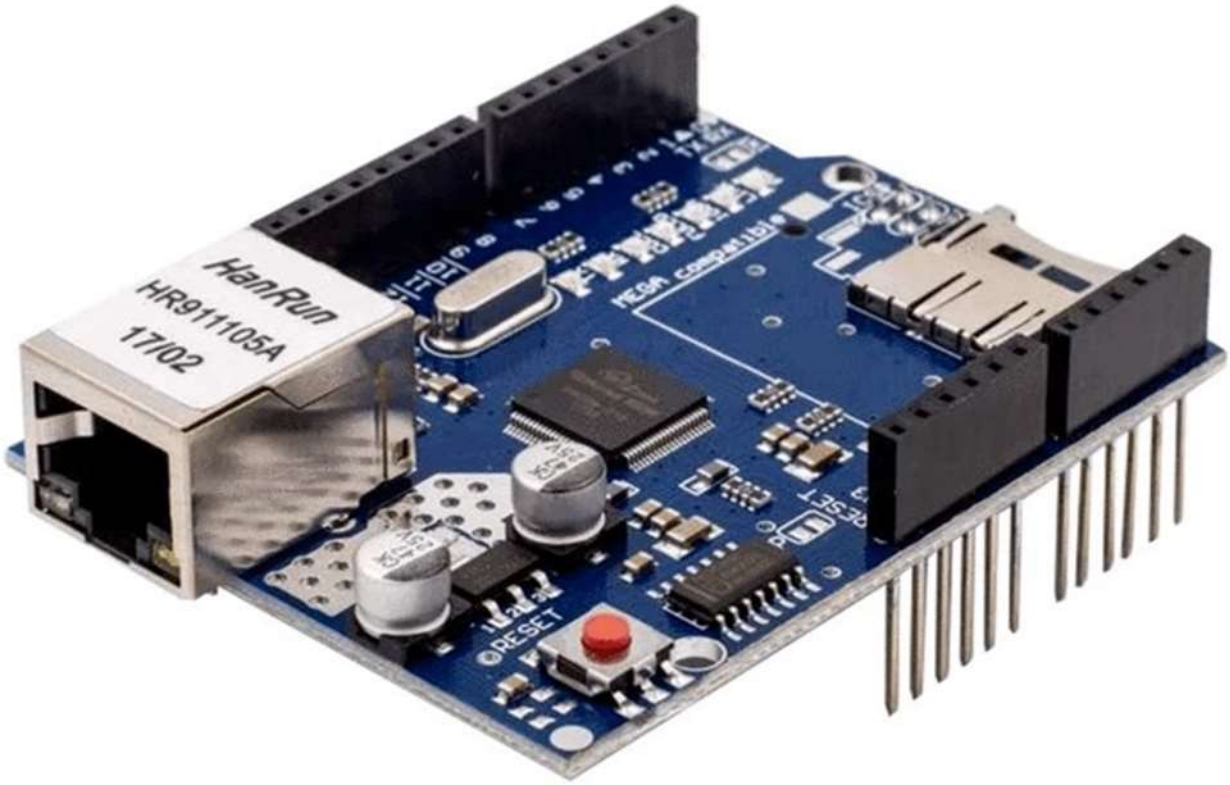
Herzlich willkommen!

Vielen Dank, dass Sie sich für unser AZ-Delivery Ethernet Shield W5100 für Arduino Uno und Mega entschieden haben. Auf den nachfolgenden Seiten geben wir Ihnen eine Einführung in das Programmieren und die Nutzung dieses praktischen Gerätes.

Viel Spaß!



Das Arduino Ethernet Shield verbindet schnell und einfach Ihr Arduino Board mit dem Internet. Sie müssen nur das Modul auf Ihr Arduino Board stecken, mit einem LAN-Kabel mit Ihrem Netzwerk verbinden (nicht enthalten) und ein paar einfachen Anweisungen folgen, um Ihr Arduino basiertes Projekt über das Internet umzusetzen.

Benötigte Komponenten:

- ein Arduino Board (nicht enthalten)
- Betriebsspannung 5V (wird vom Arduino Board bereitgestellt)
- Übertragungsgeschwindigkeit: 10/100Mb (via LAN-Kabel, nicht enthalten)
- Verbindung mit Arduino durch ein SPI Port

Das Arduino Ethernet Shield ermöglicht einem Arduino Board sich mit dem Internet zu verbinden. Es basiert auf einem Wiznet W5100 Ethernet Chip. Der Wiznet W5100 unterstützt sowohl TCP als auch UDP Netzwerk-Verbindungen. Es unterstützt bis zu vier gleichzeitige Netzwerkverbindungen. Nutzen Sie die in der Arduino IDE vorinstallierte Ethernet-Bibliothek, um Sketches zu schreiben, die sich über das Shield mit dem Internet verbinden. Das Ethernet Shield wird über ein Kabel mit dem Arduino Board verbunden, damit ein weiteres Shield aufgesteckt werden kann. Das Ethernet Shield hat eine Standard RJ-45 Verbindung, mit einem integrierten Leitungstransformator und einem aktivierten PoE.

Auf dem Board befindet sich ein Micro SD-Kartenslot, der benutzt werden kann, um Dateien über das Netzwerk zu speichern. Er ist mit allen Arduino/Genuino Boards kompatibel.

Der auf dem Board verbaute SD-Kartenleser ist durch die SD-Bibliothek erreichbar, die ebenfalls auf dem Arduino IDE vorinstalliert ist. In der Bibliothek liegt SS auf Pin 4.

Das Shield beinhaltet auch einen Reset-Controller, um sicherzustellen, dass sich das W5100 Ethernet Modul ordnungsgemäß nach dem Einschalten zurücksetzt. Ältere Modelle des Shields waren nicht mit dem Mega kompatibel und mussten manuell nach dem Einschalten zurückgesetzt werden.

Eigenschaften:

- Mit dem Ethernet Shield kann sich Ihr Arduino Board mit dem Internet verbinden
- Kann sowohl als Client als auch als Server benutzt werden
- Direkte Steckverbindung, kein Löten notwendig
- Aufgrund des Wiznet W5100 Ethernet Chips können Sie Ihren Arduino leicht online nutzen
- Direkte Unterstützung durch eine offizielle Arduino Ethernet Bibliothek
- Beinhaltet einen Micro-SD Kartenslot, um Dateien über das Netzwerk speichern zu können
- Mit Arduino Duemilanove, Uno und Mega verwendbar,
- Der Wiznet W5100 unterstützt (IP) TCP und UDP Netzwerkverbindungen.
- unterstützt bis zu vier simultane Netzverkverbindungen
- Das Shield verbindet sich mit dem Internet über Sketches, die die Ethernet-Bibliothek einbinden

Der Arduino kommuniziert sowohl mit dem W5100 als auch mit der SD-Karte über den SPI Bus (durch die ICSP Buchse). Dies entspricht den digitalen Pins 10, 11, 12, und 13 auf dem Uno und den Pins 50, 51, und 52 auf dem Mega. Auf beiden Boards wird Pin 10 benutzt, um den W5100 auszuwählen und Pin 4 für die SD-Karte. Diese Pins können generell nicht für I/O genutzt werden. Auf dem Mega wird der Hardware SS Pin 53 nicht benutzt, um den W5100 oder die SD-Karte auszuwählen, muss allerdings als Ausgabe geschaltet sein, ansonsten funktioniert das SPI Interface nicht.

Bitte beachten Sie, dass sich der W5100 und die SD-Karte den SPI Bus teilen. Aus diesem Grund können beide nicht gleichzeitig aktiv sein. Sollten Sie beide Peripheriegeräte in Ihrem Programm nutzen wollen, sollte dieses Problem von den entsprechenden Bibliotheken gelöst werden. Sollten Sie jedoch eines der beiden Peripheriegeräte nicht benutzen, müssen Sie es explizit deaktivieren. Dafür müssen Sie bei der SD-Karte Pin 4 als Ausgabe deklarieren und auf dem Pin ein High schreiben. Für den W5100 setzen Sie den digitalen Pin 10 als Ausgabe und auf High.

Das Shield stellt eine Standard RJ-45 Ethernet Buchse zur Verfügung.

Der Reset-Button auf dem Shield setzt sowohl den W5100 als auch das Arduino Board zurück.

Das Shield beinhaltet einige Status- LEDs:

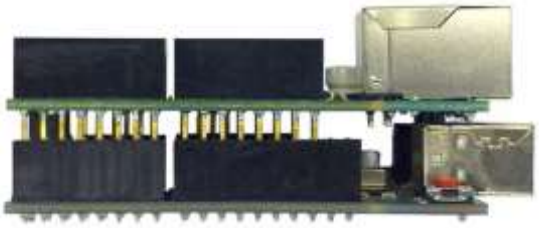
- **PWR:** Zeigt an, dass Board und Shield an sind
- **LINK:** Zeigt eine stehende Netzwerkverbindung an und blinkt bei Senden oder Empfangen von Daten auf dem Shield
- **FULLD:** Zeigt eine Vollduplex-Netzwerkverbindung an
- **100M:** Zeigt die Verfügbarkeit einer 100 Mb/s Netzwerkverbindung (im Gegensatz zu10 Mb/s)
- **RX:** Blinkt, wenn das Shield Daten empfängt
- **TX:** Blinkt, wenn das Shield Daten sendet
- **COLL:** Blinkt, wenn Netzwerkkollisionen registriert werden

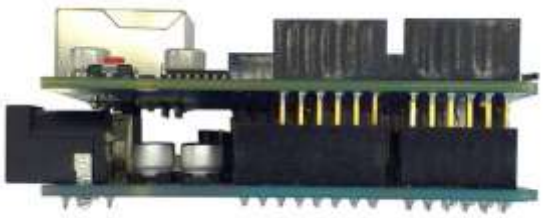
Die mit “INT” markierte Lötstelle kann mit dem Arduino Board verbunden werden, um Mitteilungen über Zwischenfälle von Programmunterbrechungen auf dem W5100 zu erhalten. Dies wird allerdings nicht von der Ethernet-Bibliothek unterstützt. Die Leitung verbindet den INT-Pin des W5100 mit dem digitalen Pin 2 auf dem Arduino.

Verbinden des Shields mit dem Arduino Board

Sie müssen das Shield nur auf das Arduino Board stecken. Auf den nachfolgenden Bildern wird dies mit einem Arduino UNO gezeigt:

Hier befindet sich eine Micro SD-Karte (nicht enthalten)!





Die Bibliothek

Die Bibliothek für dieses Shield ist bereits mit Arduino IDE vorinstalliert. Dadurch haben Sie viele Sketch-Beispiele, die Sie ausprobieren können. Sie müssen nur Ihr Shield mit dem Arduino Board verbinden, ein LAN-Kabel mit dem Shield und ein USB-Kabel mit dem Arduino.

Falls Sie die Arduino IDE nicht installiert haben, können Sie die Software für Ihr Betriebssystem unter folgendem Link herunterladen:

https://www.arduino.cc/en/main/software

Folgenden Sie den Anweisungen in der Installationssoftware.

Nach der Installation nutzen Sie die Menüpunkte File > Examples > Ethernet > Webserver und laden den Webserver Sketch herunter. Der Sketch erstellt einen Webserver und eine Webseite, liest alle analogen Eingaben und schreibt die Werte auf die Webseite.

Zuerst müssen Sie die IP-Adresse aus dem IP-Adressen-Pool für Ihr lokales Netzwerk suchen. Hier können wir eine Adresse des Adresspools „192.168.0.0“ (Netzwerkmaske 255.255.255.0) nutzen. Dies beinhaltet Adressen von 192.168.0.1 bis 192.168.0.255.
Zuerst sollten Sie überprüfen welche Adresspools in Ihrem lokalen Netzwerk verfügbar sind. In Linux können Sie die IP-Adresse und die Netzwerkmaske Ihres Computers mit dem Befehl „ifconfig“ im Terminal überprüfen. Für uns ergab sich:

IP-Adresse:	192.168.0.101
Subnetzmaske:	255.255.255.0
Broadcast:	192.168.0.255

Für Windows OS öffnen Sie die Eingabeaufforderung und geben Sie den Befehl „ipconfig“ ein.

In unserem Fall können wir die Adresse des PCs 192.168.0.101 und die Broadcast-Adresse 192.168.0.255 nicht verwenden. Alle anderen Adressen in dem zuvor genannten Bereich können benutzt werden. Beispielsweise ist die Adresse 192.168.0.254 verwendbar.

In diesem Sketch ändern wir zuerst diese Codezeile:

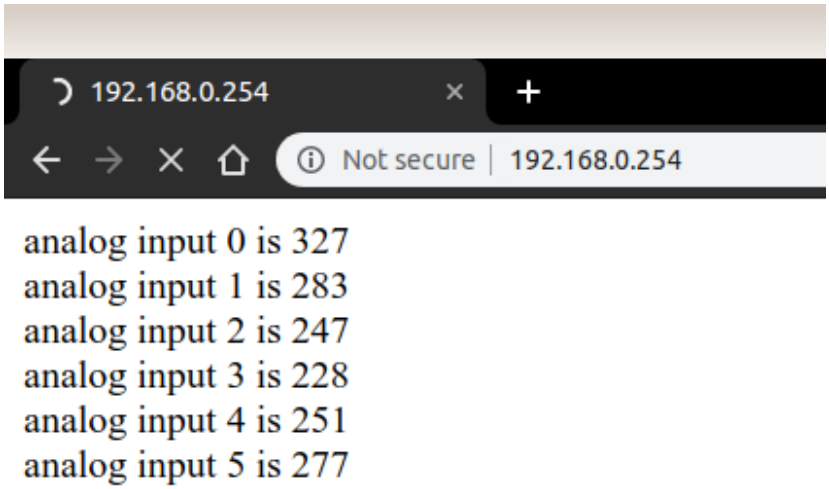
```
IPAddress ip (192, 168, 1, 177);
```

zu

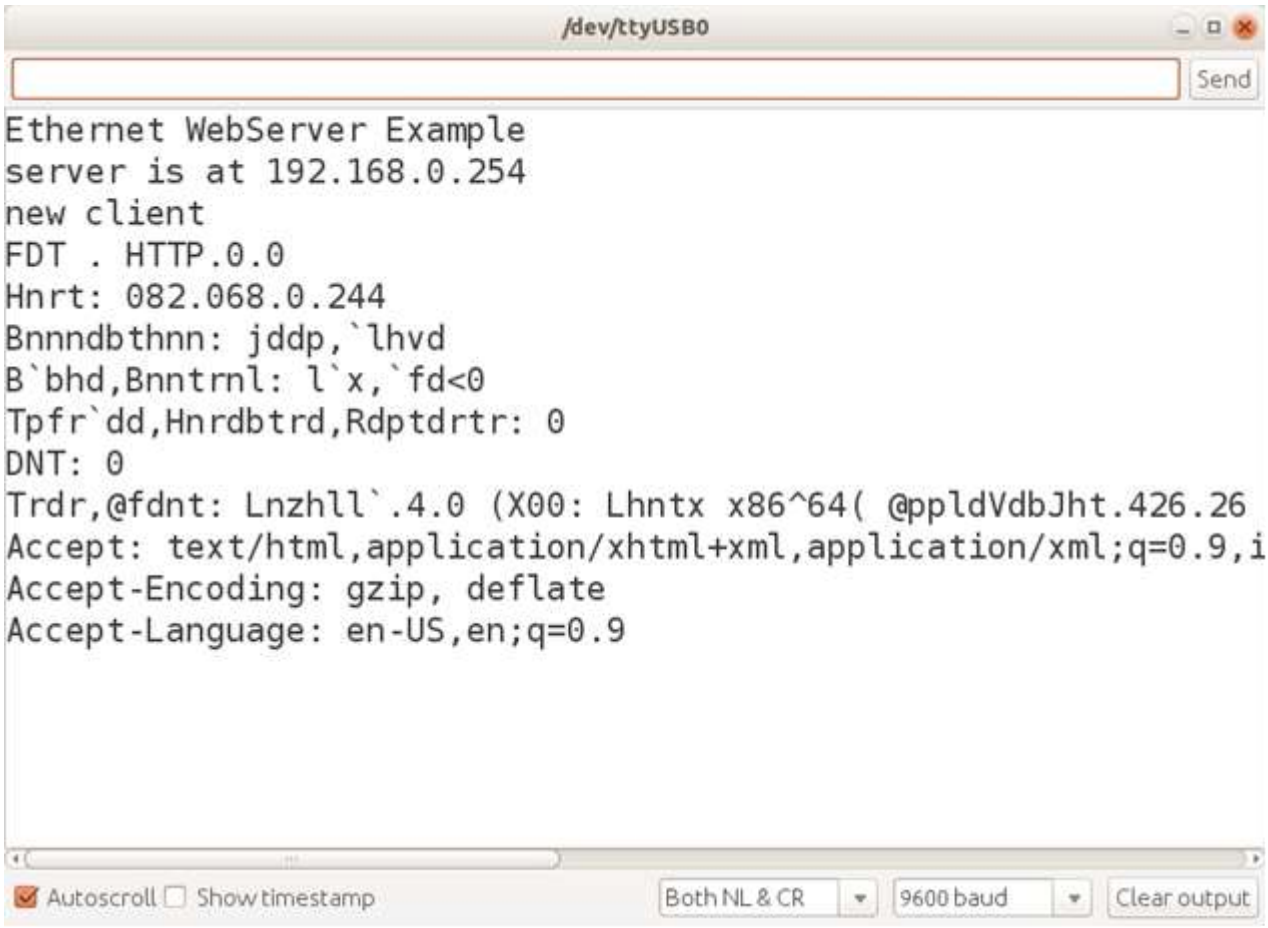
```
IPAddress ip (192, 168, 0, 254);
```

Oder zu einer IP-Adresse, die zu Ihrem IP-Adressen-Pool gehört.

Wenn Sie den Sketch auf Ihren Arduino geladen haben, öffnen Sie Ihren Internetbrowser und tippen Sie `192.168.0.254` ein. Folgende Webseite sollte geladen werden:



Wenn Sie Ihren Serial Monitor öffnen, `(Tools > Serial Monitor)` sollte die Ausgabe folgendermaßen aussehen:



Die Webseite aktualisiert sich alle 5 Sekunden. Dies kann durch Ändern der Zahl 5 in dieser Code-Zeile behoben werden:

```
client.println("Refresh: 5");//refresh the page automatically
every 5
```

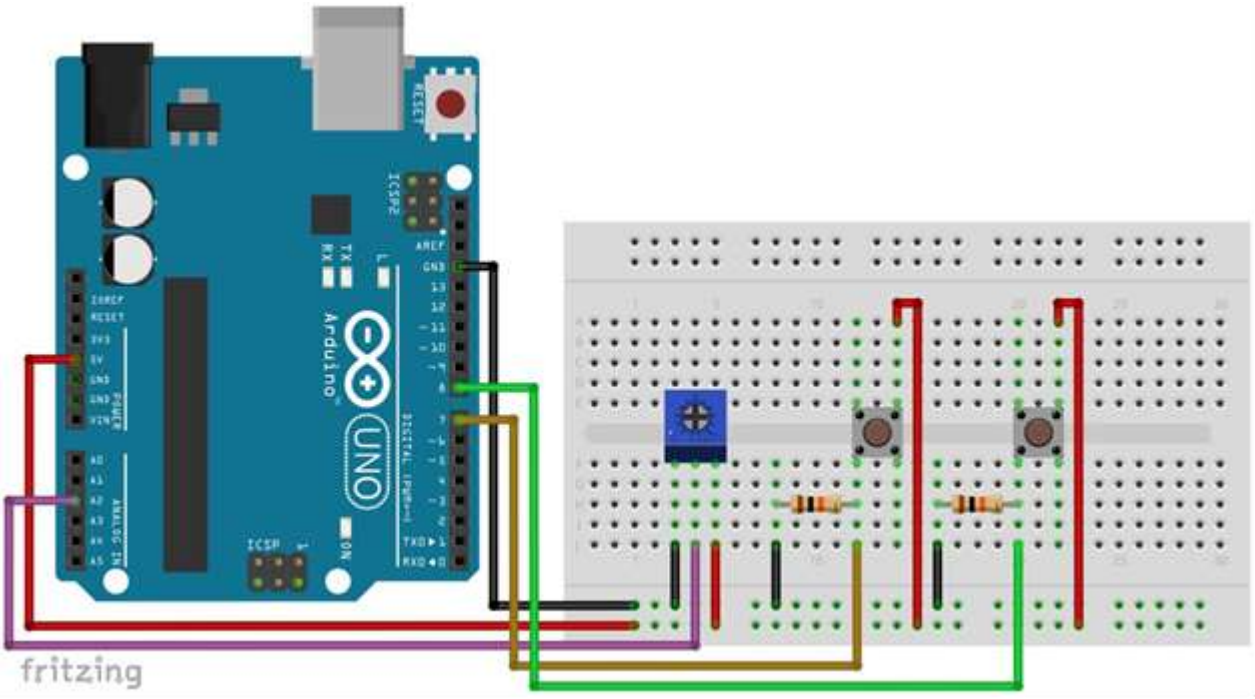
Senden der Webseite von der SD-Karte mit AJAX

Mit diesem Programmbeispiel versuchen wir einen Webserver einzurichten, der eine Webseite ausgibt, die in der Lage ist den Status von zwei Druckknöpfen und die Spannung eines Potentiometers zu lesen.

Die Knöpfe sind mit den digitalen I/O-Pin 7 und 8 und das Potentiometer mit dem analogen Pin 2 auf dem Arduino UNO verbunden.

In dem Sketch wird AJAX verwendet, um die Werte auf der Webseite zu aktualisieren ohne die ganze Webseite neu zu laden.

Verbinden Sie alles so, wie auf dem Verbindungsdiagramm unten:



Wie Sie sehen können benutzen wir für jeden Knopf einen Pull-Down-Widerstand von 10kΩ, was bedeutet, dass wenn der Knopf nicht gedrückt wird ein Low-Level-Signal (oder 0 V) an den digitalen Input-Pin ausgegeben wird. Sobald der Knopf gedrückt wird, liest der digitale Input-Pin ein High-Level-Signal.

Wir verbinden das Ethernet Shield mit dem Arduino UNO genau wie in den vorangegangenen Kapiteln gezeigt und benutzen die Micro SD-Karte, die als einzige Datei "*index.htm*" gespeichert haben sollte. Sie können einen beliebigen Texteditor benutzen, um die Datei zu erstellen und mit folgendem html und javascript Code zu füllen:

```
<!DOCTYPE html>
<html>
     <head>
          <title>Arduino SD Card Web Page using Ajax with
     XML</title>
          <script>
          function GetArduinoInputs() {
          nocache = "&nocache=" + Math.random() * 1000000; var re-
          quest = new XMLHttpRequest(); request.onreadystatechange
          = function() {
           if (this.readyState == 4) {
                if (this.status == 200) {
                     if (this.responseXML != null) {
// extract XML data from XML file (containing switch states and
analog value)
                     document.getElementById("input1").innerHTML =
this.responseXML.getElementsByTagName('button1')[0].childNodes[0].n
odeValue;
                     document.getElementById("input2").innerHTML =
this.responseXML.getElementsByTagName('button2')[0].childNodes[0].n
odeValue;
                     document.getElementById("input3").innerHTML =
this.responseXML.getElementsByTagName('analog1')[0].childNodes[0].n
odeValue;
                }
            }
      }
 }
```

```
(3 tabs)          request.open("GET", "ajax_inputs" +  nocache, true);
                  request.send(null);
                  setTimeout('GetArduinoInputs()',  1000);
            }
      </script>
      </head>
      <body  onload="GetArduinoInputs()">
            <h1>Arduino  Inputs  from  SD Card  Web Page using  Ajax
            with  XML</h1>
            <p>Button 1 (pin 7): <span id="input1">...</span></p>
            <p>Button 2 (pin 8): <span id="input2">...</span></p>
            <p>Analog (A2): <span id="input3">...</span></p>
      </body>
</html>
```

Nachdem Sie die Datei erstellt haben, müssen Sie diese auf die Micro SD-Karte und stecken Sie die Karte nachfolgend in das Shield.

Danach müssen Sie noch den Sketch in Ihre Arduino IDE kopieren. Der Code ist auf der nächsten Seite. Sie müssen jedoch sicherstellen, dass Sie die IP-Adresse wie vorhergehend besprochen selbst einstellen. In unserem Beispiel haben wir die Adresse 192.168.0.254 gewählt.

Wir erklären Ihnen nun das Programm.

Zu Beginn fügen wir drei Bibliotheken hinzu. Die SPI-Bibliothek ist für die Kommunikation zwischen Shield und SD-Karte mit dem Arduino verantwortlich. Die Zweite, die Ethernet-Bibliothek, ist für das Ethernet-Shield und die Dritte für die SD-Karte auf dem Shield.

Nun stellen wir die MAC-Adresse des Shields ein, da diese für eine erfolgreiche Verbindung zum Internet benötigt wird. Danach wird die IP-Adresse wie vorher besprochen festgelegt. Nachfolgend wird der Server als Objekt eingerichtet und der Serverport auf den Port 80 gelegt.

Im Setup-Bereich wird der Ethernetchip zuerst deaktiviert, da über das SPI-Interface immer nur ein Gerät betrieben werden kann. Das Arduino-Board ist hierbei der Master, der Ethernetchip slave Nummer 1 und die SD-Karte slave Nummer 2. Um nun die SD-Karte einzurichten, muss zuerst der Ethernetchip deaktiviert werden. Danach wird das serielle Interface mit einer Baudrate von 9600 eingerichtet. Dieses benutzen wir für das Debugging.

Es wird überprüft, ob im SD-Kartenslot eine Karte steckt und falls ja, ob die Datei „index.htm“ existiert. Danach erfolgt eine Ausgabe auf das serielle Interface.

Danach werden die digitalen I/O-Pins 7 und 8 als Eingabepins deklariert und der Webserver gestartet.

```
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
// size of buffer used to capture HTTP requests
#define REQ_BUF_SZ 50
// MAC address from Ethernet shield sticker under board
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 0, 254); // IP address, may need to change
                                // depending on network
EthernetServer server(80); // create a server at port 80
File webFile; // the web page file on the SD card
char HTTP_req[REQ_BUF_SZ] = {0}; // buffered HTTP request stored
                                // as null terminated string
char req_index = 0;             // index into HTTP_req buffer
void setup() {
      // disable Ethernet chip
      pinMode(10, OUTPUT);
      digitalWrite(10, HIGH);
      Serial.begin(9600); // for debugging
      Serial.println("Initializing SD card..."); // initialize SD
      card
      if(!SD.begin(4)) {
            Serial.println("ERROR - SD card initialization failed!");
            return; // init failed
}
Serial.println("SUCCESS - SD card initialized.");
// check for index.htm file
if(!SD.exists("index.htm")) {
      Serial.println("ERROR - Can't find index.htm file!");
      return; // can't find index file
}
Serial.println("SUCCESS - Found index.htm file.");
pinMode(7, INPUT);          // switch is attached to Arduino pin 7
pinMode(8, INPUT);          // switch is attached to Arduino pin 8
Ethernet.begin(mac, ip);    // initialize Ethernet device
server.begin();             // start to listen for clients
}
```

```
#include <Ethernet.h>
#include <SD.h>
// size of buffer used to capture HTTP requests
#define REQ_BUF_SZ   50
// MAC address from Ethernet shield sticker under board
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 0, 254);     // IP address, may need to change
                                    // depending on network
EthernetServer server(80);  // create a server at port 80
File webFile;               // the web page file on the SD card
char HTTP_req[REQ_BUF_SZ] = {0};    // buffered HTTP request stored
                                    // as null terminated string
char req_index = 0;              // index into HTTP_req buffer
void setup() {
    // disable Ethernet chip
    pinMode(10, OUTPUT);
    digitalWrite(10, HIGH);
    Serial.begin(9600);       // for debugging
    Serial.println("Initializing SD card..."); // initialize SD card
    if(!SD.begin(4)) {
        Serial.println("ERROR - SD card initialization failed!");
        return;    // init failed
    }
    Serial.println("SUCCESS - SD card initialized.");
    // check for index.htm file
    if(!SD.exists("index.htm")) {
        Serial.println("ERROR - Can't find index.htm file!");
        return;  // can't find index file
    }
    Serial.println("SUCCESS - Found index.htm file.");
    pinMode(7, INPUT);        // switch is attached to Arduino pin 7
    pinMode(8, INPUT);        // switch is attached to Arduino pin 8
    Ethernet.begin(mac, ip);  // initialize Ethernet device
    server.begin();           // start to listen for clients
}
```

```
void loop() {
    EthernetClient client = server.available();  // try to get client
    if(client) {  // got client?
        boolean currentLineIsBlank = true;
        while(client.connected()) {
            if(client.available()) {   // client data available to read
                char c = client.read(); // read 1 byte (character) from client
// buffer first part of HTTP request in HTTP_req array (string)
// leave last element in array as 0 to null terminate string (REQ_BUF_SZ - 1)
                if(req_index < (REQ_BUF_SZ - 1)) {
                    HTTP_req[req_index] = c; // save HTTP request character
                    req_index++;
                }
                // last line of client request is blank and ends with \n
                // respond to client only after last line received
                if(c == '\n' && currentLineIsBlank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    // remainder of header follows below, depending on if
                    // web page or XML page is requested
                    // Ajax request - send XML file
                    if(StrContains(HTTP_req, "ajax_inputs")) {
                        // send rest of HTTP header
                        client.println("Content-Type: text/xml");
                        client.println("Connection: keep-alive");
                        client.println();
                        // send XML file containing input states
                        XML_response(client);
                    }
                    else {  // web page request
                        // send rest of HTTP header
                        client.println("Content-Type: text/html");
                        client.println("Connection: keep-alive");
                        client.println();
                        // send web page
                        webFile = SD.open("index.htm"); // open web page file
                        if(webFile) {
                            while(webFile.available()) {
                                // send web page to client
                                client.write(webFile.read());
                            }
                            webFile.close(); }
```

```
(5 tabs)                }
                        // display received HTTP request on serial port
                        Serial.print(HTTP_req);
                        // reset buffer index and all buffer elements to 0
                        req_index = 0;
                        StrClear(HTTP_req, REQ_BUF_SZ);
                        break;
                    }
                    // every line of text received from the client ends with \r\n
                    if(c == '\n') {
                        // last character on line of received text
                        // starting new line with next character read
                        currentLineIsBlank = true;
                    }
                    else if(c != '\r') {
                        // a text character was received from client
                        currentLineIsBlank = false;
                    }
                } // end if (client.available())
            } // end while (client.connected())
            delay(1);       // give the web browser time to receive the data
            client.stop(); // close the connection
        } // end if (client)
}

// send the XML file with switch statuses and analog value
void XML_response(EthernetClient cl) {
    int analog_val;
        cl.print("<?xml version = \"1.0\" ?>");
    cl.print("<inputs>");
    cl.print("<button1>");
    if(digitalRead(7)) {
        cl.print("ON");
    }
    else {
        cl.print("OFF");
    }
    cl.print("</button1>");
    cl.print("<button2>");
    if (digitalRead(8)) {
        cl.print("ON");
    }
```

```
(1 tab)
    else {
        cl.print("OFF");
    }
    cl.print("</button2>");
    // read analog pin A2
    analog_val = analogRead(2);
    cl.print("<analog1>");
    cl.print(analog_val);
    cl.print("</analog1>");
    cl.print("</inputs>");
}
// sets every element of str to 0 (clears array)
void StrClear(char *str, char length) {
    for(int i = 0; i < length; i++) {
        str[i] = 0;
    }
}
// searches for the string sfind in the string str
// returns 1 if string found
// returns 0 if string not found
char StrContains(char *str, char *sfind) {
    char found = 0;
    char index = 0;
    char len;
    len = strlen(str);
    if(strlen(sfind) > len) {
        return 0;
    }
    while(index < len) {
        if(str[index] == sfind[found]) {
            found++;
            if(strlen(sfind) == found) {
                return 1;
            }
        }
        else {
            found = 0;
        }
        index++;
    }
    return 0;
}
```

In der Schleifenfunktion wird darauf gewartet, dass sich ein Client mit dem Webserver verbindet. Sobald dies geschieht, stellt der Server die Webseite bereit.

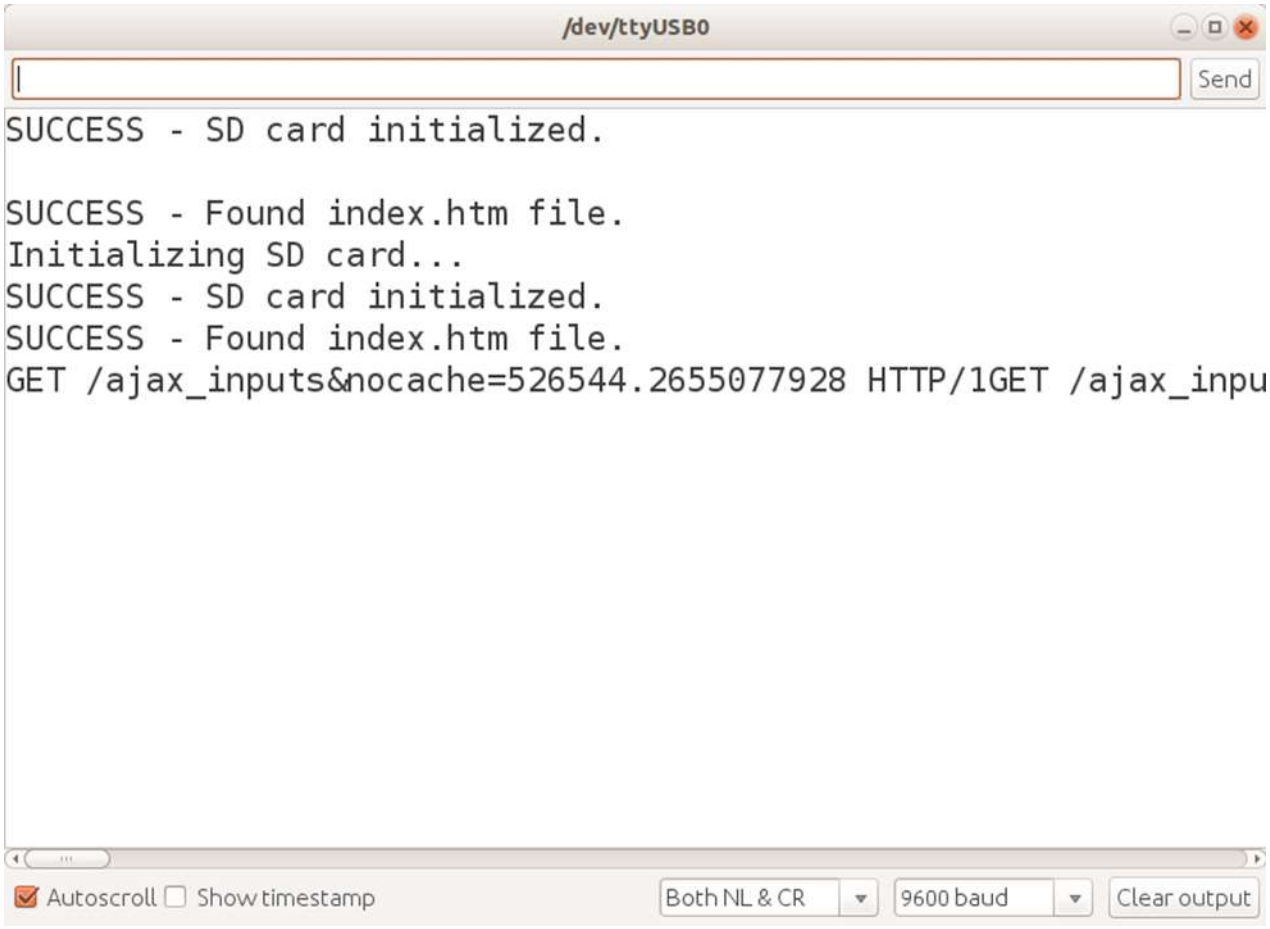
Wir gehen nachfolgend nicht ins Detail wie die Webseite übertragen wird, weil wir dafür HTML, Javascript und die Arduino IDE-Sprache verwenden. Zusätzlich erstellen wir eine XML-Datei, um Daten zwischen dem Webserver und der Webseite auszutauschen (Dadurch erfolgt dieser schneller.) Wir erstellen eine XML-Datei in diesem Sketch nur als Beispiel. Wir benutzen die XML_response-Funktion, um die Datei zu erstellen und um die Status der digitalen Input-Pins 7 und 8 und den Wert des analogen Input-Pins 2 zu lesen.

Das Programm ist ausführlich kommentiert und Sie sollten leicht jeden entsprechenden Programmteil finden und verstehen, was für was benötigt wird.

Sobald Sie den Sketch auf das Arduino-Board geladen haben, öffnen Sie einen Webbrowser und geben die von Ihnen im Sketch verwendete IP-Adresse in die Adresszeile ein. In unserem Beispiel ist das die 192.168.0.254. Drücken Sie Enter und die Webseite sollte in Ihrem Browser wie folgt angezeigt werden:

Sobald Sie einen der Knöpfe betätigen, ändert sich der OFF-Status zu einem ON-Status, ohne dass die Webseite neu geladen wird. Sobald Sie den Drehknopf des Potentiometers betätigen sollte sich der Wert der analogen Variable im Bereich von 0 bis 1023 verändern. Der Wert wird jede Sekunde einmal aktualisiert.

Wenn Sie den Serial Monitor (*Tools* > *Serial Monitor*) öffnen, sollte die Ausgabe wie folgt aussehen.



Sie haben es geschafft. Nun können Sie das Modul für Ihre Projekte verwenden.

Nun ist es an der Zeit neue Projekte selbstständig in Angriff zu nehmen. Dabei unterstützen Sie viele Beispiel-Sketches und Tutorials, die Sie im Internet finden.

Wenn Sie auf der Suche nach hochwertigen Produkten für Arduino und Raspberry Pi sind, sind wir von AZ-Delivery Vertriebs GmbH der richtige Ansprechpartner. Wir unterstützen Sie mit vielen Anwendungsbeispielen, Einrichtungshilfen, eBooks, Bibliotheken und natürlich unseren Technik-Experten!

https://az-delivery.de

Viel Spaß!

Impressum

https://az-delivery.de/pages/about-us