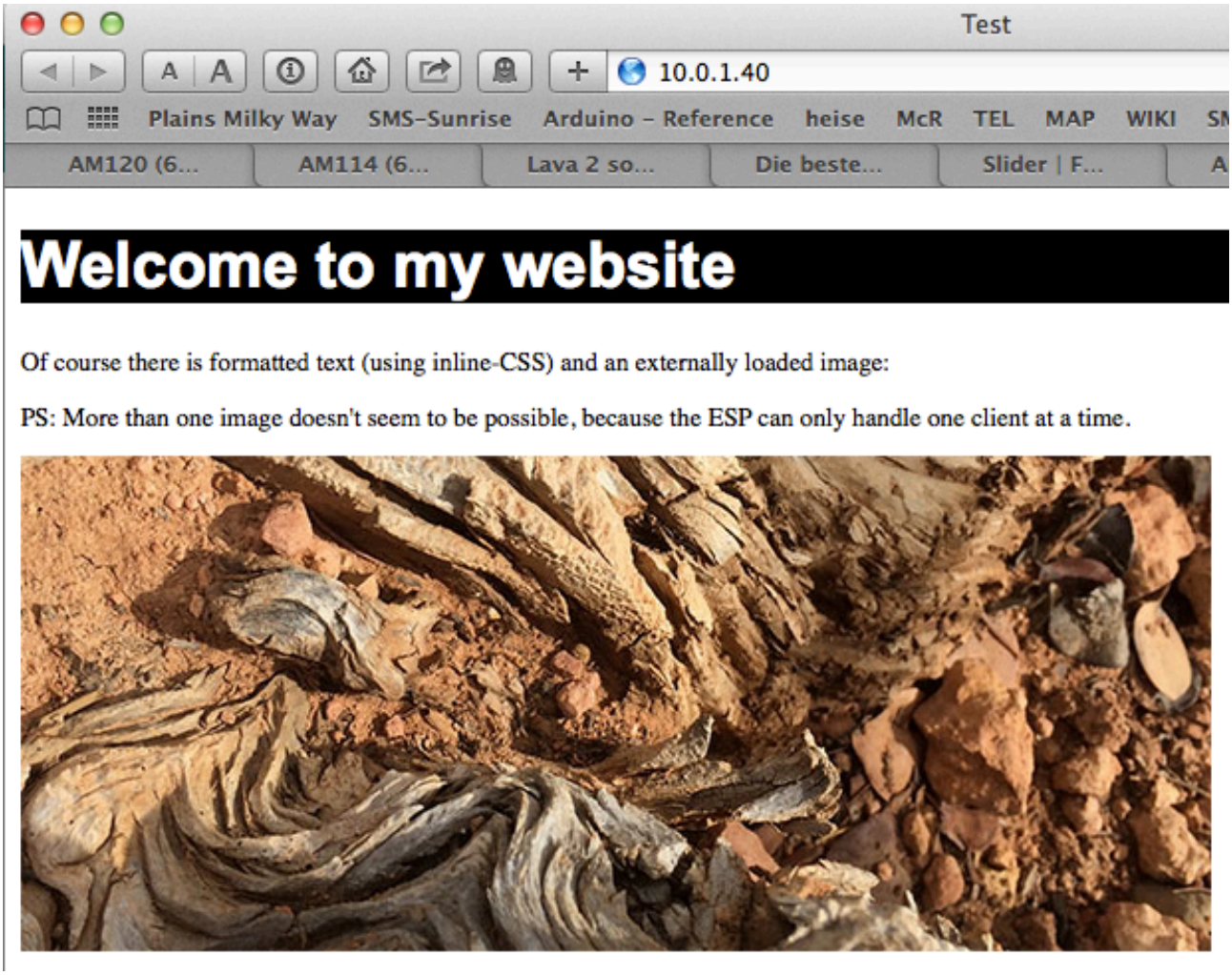
ESP als Webserver für richtige Webseiten



Probleme:

1. Der ESP8266-12E hat nur 64kB RAM, viele Webseiten enthalten aber Dateien, die viel grösser sind.
2. Der ESP kann immer nur eine Verbindung aufrecht erhalten. Braucht eine Webseite viele Dateien, kann das zu Problemen führen, weil die Browser mehrere Anfragen zeitnah stellen.

Lösungen:

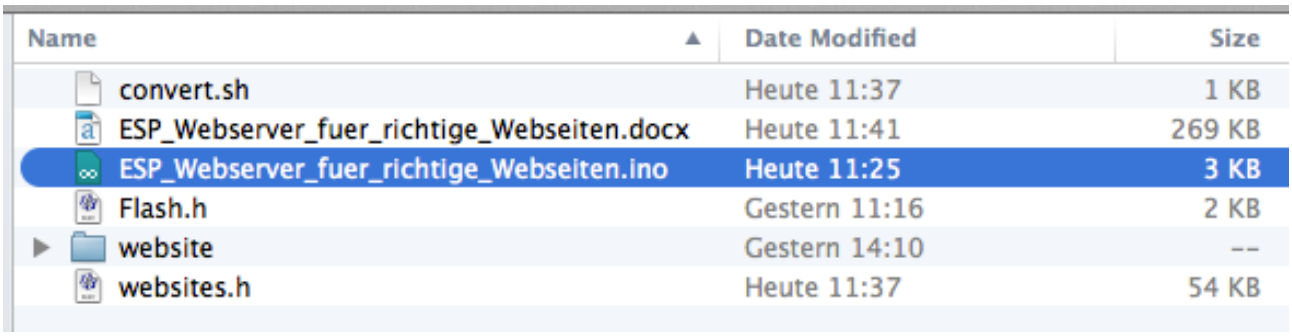
1. Webseitendateien werden im Flash (mindestens 1 MByte gross) gespeichert und von dort ausgegeben. Dazu gibt es ein Shell-Script (convert.sh), das die Dateien des Ordners «website» in die Datei «websites.h» übersetzt, welche schliesslich mittels «Flash.h» in den Flash geladen wird.

   Funktioniert gut auch mit Dateien, die einige Duzend kB gross sind.
2. Möglichst alle Dateien der Webseite in eine Datei verpacken (Skripte, CSS, auch Bilder, falls mehr als ein Bild eingefügt werden soll).

Code:

Übersicht über die nötigen Dateien:

1a. ESP_Webserver_Foundation.ino: Webserver-Code
1b. Flash.h: Handelt das Schreiben und Lesen in den Flash-Speicher
1c. convert.sh: Übersetzt die Webseite in websites.h (Vorbereitung für Überführung in Flash)
1d. website: Die Dateien der Webseite: index.html und bild.jpg
1e. websites.h: Die strukturierte Byte-Variante der Webseite-Dateien



1a. ESP_Webserver_fuer_richtige_Webseiten.ino: Webserver-Code

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include "websites.h"

// Source: http://www.esp8266.com/viewtopic.php?f=32&t=3780# - Tom torx@esp8266.com
// Thank you very much Tom
// Incorporated improvement suggested by User treii28@esp8266.com
// Used WiFi-setup as posted at
// https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/
// examples/WiFiWebServer/WiFiWebServer.ino

// Modyfied by Daniel Suesstrunk, local_dani_21@wunderwald.ch, 2017

const char *ssid = "w";
const char *password = "password";

ESP8266WebServer server(80);

void handleNotFound();
bool loadFromFlash(String path);

void setup()
{
  // Enable serial port for debugging purposes
  Serial.begin ( 115200 );
  Serial.println("ESP starting up...");

  WiFi.begin ( ssid, password );
  Serial.println ( "" );

  // Wait for connection
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }

  Serial.println ( "" );
  Serial.print ( "Connected to " );
  Serial.println ( ssid );
  Serial.print ( "IP address: " );
  Serial.println ( WiFi.localIP() );

  // in the notFound we search for files ourselves
  server.onNotFound(handleNotFound);
```

```
  server.begin();
}

bool loadFromFlash(String path) {
  if(path.endsWith("/")) path += "index.html";
  Serial.println(path);

  int NumFiles = sizeof(files)/sizeof(struct t_websitefiles);

  for(int i=0; i<NumFiles; i++) {
    if(path.endsWith(String(files[i].filename))) {
      _FLASH_ARRAY<uint8_t>* filecontent;
      String dataType = "text/plain";
      unsigned int len = 0;
      WiFiClient client = server.client();

      dataType = files[i].mime;
      len = files[i].len;

      server.setContentLength(len);
      server.send(200, files[i].mime, "");

      filecontent = (_FLASH_ARRAY<uint8_t>*)files[i].content;

      filecontent->open();

      client.write(*filecontent, 100);
      return true;
    }
  }

  return false;
}

void handleNotFound() {

  // try to find the file in the flash
  if(loadFromFlash(server.uri())) return;

  String message = "File Not Found\n\n";
  message += "URI..........: ";
  message += server.uri();
  message += "\nMethod.....: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments..: ";
  message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  message += "\n";
  message += "FreeHeap.....: " + String(ESP.getFreeHeap()) + "\n";
  message += "ChipID.......: " + String(ESP.getChipId()) + "\n";
  message += "FlashChipId..: " + String(ESP.getFlashChipId()) + "\n";
  message += "FlashChipSize: " + String(ESP.getFlashChipSize()) + " bytes\n";
  message += "getCycleCount: " + String(ESP.getCycleCount()) + " Cycles\n";
  message += "Milliseconds.: " + String(millis()) + " Milliseconds\n";
  server.send(404, "text/plain", message);
}

void loop()
{
  // process webserver
  server.handleClient();

}
```

1b. Flash.h: Handelt das Schreiben und Lesen in den Flash-Speicher

```
#ifndef __FLASH_H__
#define __FLASH_H__

#include <pgmspace.h>
#include "Arduino.h"

// Example: FLASH_ARRAY(float, temperatures, 98.1, 98.5, 99.1, 102.1);
#define FLASH_ARRAY(type, name, values...) \
  static const type name##_flash[] PROGMEM = { values }; \
  _FLASH_ARRAY<type> name(name##_flash, sizeof(name##_flash) / sizeof(type));


#ifndef ARDUINO_CORE_PRINTABLE_SUPPORT
class _Printable
{
public:
  virtual void print(Print &stream) const = 0;
};
#endif

/* _FLASH_ARRAY template class.  Use the FLASH_ARRAY() macro to create these. */
template<class T>
class _FLASH_ARRAY : public _Printable
{
  typedef T _DataType;

public:
  _FLASH_ARRAY(const _DataType *arr, size_t count) : _arr(arr), _size(count)
  { }

  size_t count() const
  { return _size; }

  size_t size() const
  { return _size; }

  size_t available()
  { return _size -_lastread; }

  void open()
  { _lastread=0; }

  void close()
  { _lastread=0; }

  size_t read(uint8_t *dst, size_t len)
  {
    size_t i = 0;

    for(i=0; i<len; i++) {

      if( _lastread >= _size ) {
        break;
      }

      dst[i] = (*this)[_lastread];
      _lastread++;
    }
    return i;
  }

  const _DataType *access() const
  { return _arr; }

  T operator[](int index) const
  {
    uint32_t val = 0;
    if (sizeof(T) == 1)
```

```
      val = pgm_read_byte(_arr + index);
    else if (sizeof(T) == 2)
      val = pgm_read_word(_arr + index);
    else if (sizeof(T) == 4)
      val = pgm_read_dword(_arr + index);
    return *reinterpret_cast<T *>(&val);
  }

  void print(Print &stream) const
  {
    for (size_t i=0; i<_size; ++i)
    {
      stream.print((*this)[i]);
      if (i < _size - 1)
        stream.print(",");
    }
  }

private:
  const _DataType *_arr;
  size_t _size;
  size_t _lastread;
};

#ifndef ARDUINO_STREAMING
#define ARDUINO_STREAMING

template<class T>
inline Print &operator <<(Print &stream, T arg)
{ stream.print(arg); return stream; }

#endif

inline Print &operator <<(Print &stream, const _Printable &printable)
{ printable.print(stream); return stream; }

template<class T>
inline Print &operator <<(Print &stream, const _FLASH_ARRAY<T> &printable)
{ printable.print(stream); return stream; }

#endif // def __FLASH_H
```

1c. convert.sh: Übersetzt die Webseite in websites.h (Vorbereitung für Überführung in Flash)

```
#!/bin/bash

#Dieses Skript übersetzt Dateien im Ordner websites (ohne Unterverzeichnisse) in eine
Flash-Datei,
#die dann auf den ESP ins Flash geladen werden kann.

echo "Converting files in folder \"websites\" to C Header file websites.h"

WEBSITE="website/"
CURRDIR="$(pwd)"
OUTFILE="$CURRDIR/websites.h"

#change into website folder
cd $WEBSITE

cat > $OUTFILE <<DELIMITER
//
// converted websites to mainly flash variables
//

#include "Flash.h"

DELIMITER

#convert contents into array of bytes
```

```
INDEX=0
for i in $(ls -1); do
  CONTENT=$(cat $i | xxd -i)
  printf "FLASH_ARRAY(uint8_t, file_$INDEX,\n$CONTENT\n);\n" >> $OUTFILE
  echo >> $OUTFILE
  INDEX=$((INDEX+1))
done

# write typedefinition
cat >> $OUTFILE <<DELIMITER
struct t_websitefiles {
  const char* filename;
  const char* mime;
  const unsigned int len;
  const _FLASH_ARRAY<uint8_t>* content;
} files[] = {
DELIMITER

# add other data and create array
INDEX=0
for i in $(ls -1); do
  CONTENT=$(cat $i | xxd -i)
  CONTENT_LEN=$(echo $CONTENT | grep -o '0x' | wc -l)
  MIMETYPE=$(file --mime-type -b $i)

  echo "  {" >> $OUTFILE
  echo "    .filename = \"$i\"," >> $OUTFILE
  echo "    .mime = \"$MIMETYPE\"," >> $OUTFILE
  echo "    .len = $CONTENT_LEN," >> $OUTFILE
  echo "    .content = &file_$INDEX" >> $OUTFILE
  echo "  }," >> $OUTFILE

  INDEX=$((INDEX+1))
done
echo "};" >> $OUTFILE

cd $CURRDIR
```

1d. website: Die Dateien der Webseite: index.html und bild.jpg

1e. websites.h: Die strukturierte Byte-Variante der Webseite-Dateien (gekürzt)

```
//
// converted websites to mainly flash variables
//

#include "Flash.h"

FLASH_ARRAY(uint8_t, file_0,
  0xff, 0xd8, 0xff, 0xe1, 0x00, 0x18, 0x45, 0x78, 0x69, 0x66, 0x00, 0x00,
  0x49, 0x49, 0x2a, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xff, 0xec, 0x00, 0x11, 0x44, 0x75, 0x63, 0x6b,
  0x65, 0x26, 0x90, 0x6d, 0xb1, 0xd7, 0xa6, 0x70, 0x3f, 0x65, 0x9f, 0xdb,
  0xfd, 0x8f, 0xff, 0xd9 [gekürzt]
);

FLASH_ARRAY(uint8_t, file_1,
  0x3c, 0x21, 0x44, 0x4f, 0x43, 0x54, 0x59, 0x50, 0x45, 0x20, 0x68, 0x74,
  0x6d, 0x6c, 0x3e, 0x0d, 0x3c, 0x68, 0x74, 0x6d, 0x6c, 0x20, 0x64, 0x69,
  0x72, 0x3d, 0x22, 0x6c, 0x74, 0x72, 0x22, 0x20, 0x6c, 0x61, 0x6e, 0x67,
  0x3d, 0x22, 0x64, 0x65, 0x2d, 0x63, 0x68, 0x22, 0x3e, 0x0d, 0x20, 0x20,
  0x6c, 0x3e, 0x0d [gekürzt]
);

struct t_websitefiles {
  const char* filename;
  const char* mime;
  const unsigned int len;
  const _FLASH_ARRAY<uint8_t>* content;
```

```
} files[] = {
  {
    .filename = "bild.jpg",
    .mime = "image/jpeg",
    .len =    71981,
    .content = &file_0
  },
  {
    .filename = "index.html",
    .mime = "text/html",
    .len =      596,
    .content = &file_1
  },
};
```