

Übung zum Webserver

Verteilte Systeme

Mouaad Gssair

Andreas Baur

Daniel Grotz

Andreas Roth

30. Mai 2019

Table of Contents

[1. Einleitung 3](#_Toc10129933)

[2. Einrichten/Vorbereitung 3](#_Toc10129934)

[2.1. Projektordner herunterladen 3](#_Toc10129935)

[3. Implementierung 3](#_Toc10129936)

[3.1. Interrupt Vektor-Tabelle anpassen 3](#_Toc10129937)

[3.2. FatFs Port-Anpassungen 3](#_Toc10129938)

[3.3. Lange Filenamen aktivieren 3](#_Toc10129939)

[3.4. File-APIs einfügen 3](#_Toc10129940)

[3.5. HTML Dateien ergänzen und speichern 3](#_Toc10129941)

[3.6. LwIP Optionen anpassen 3](#_Toc10129942)

[3.6.1. DHCP 3](#_Toc10129943)

[3.6.2. AutoIP 4](#_Toc10129944)

[3.6.3. Weitere Optionen 4](#_Toc10129945)

[4. Test & Lösung 4](#_Toc10129946)

[4.1. Schritte 4](#_Toc10129947)

[5. Optionale Features 4](#_Toc10129948)

[5.1. GPIO Port konfig 4](#_Toc10129949)

[5.2. GPIO Handler Implementierung 5](#_Toc10129950)

[5.3. HTTP-GET Request (javascript) 5](#_Toc10129951)

[5.4. HTTP-GET Handler (fs-open) 5](#_Toc10129952)

# Einleitung

# Einrichten/Vorbereitung

## Projektordner herunterladen

Link zur Git Repository hinzufügen

# Implementierung

## Interrupt Vektor-Tabelle anpassen

In diesem Schritt müssen in der Interrupt Vektor Tabelle folgende Interrupt-Handler angelegt werden.

**lwIPEthernetIntHandler:** Wird aufgerufen bei empfangen einer gültigen Nachricht über den Ethernetport.

**SysTickIntHandler:** Wird für den periodischen Aufruf des lwIP-Protokoll-Stacks benötigt.

**disk\_timerproc:** Wird von FatFS benötigt um Zeitüberschreitungen zu signalisieren und Pausen zu generieren.



Anschließend muss die Interrupt-Service-Routine für den FatFS-Timer in der Datei **mmc-ek-tm4c1294xl.c** (im **Projekt-Ordner** unter \third\_party\fatfs\port zu finden) implementiert werden.

**void** **disk\_timerproc** (**void**)

{

// BYTE n, s;

BYTE n;

n = Timer1; /\* 100Hz decrement timer \*/

**if** (n) Timer1 = --n;

n = Timer2;

**if** (n) Timer2 = --n;

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

}

## Timer Initialisierung

Die zuvor beschriebene Interrupt-Service-Routine disk\_timerproc soll alle 10ms aufegrufen werden. Hierfür wird ein Timer benötigt. Er wird so eingestellt, dass alle 10ms ein Timer-Overflow-Interrupt generiert wird. Die folgende Initialisierung des Timers erfolgt in der Datei **webserv\_appl\_fcns.c.**

**void** **InitTimer0Int**(**void**)

{

uint32\_t ui32\_Period;

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER0);

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

ui32\_Period = (g\_ui32SysClock/100); //120MHz/100 => 1.2Mticks -> Period of 0.01s

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, ui32\_Period - 1); //-1 zero-count

**TimerIntEnable**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

**IntEnable**(INT\_TIMER0A);

}

## FatFs Port-Anpassungen

Um die SD-Karte nutzen zu können, müssen die Ports für die SPI-Schnittstelle an das verwendete Boosterpack angepasst werden. Folgend sind die Port-Definitionen aufgelistet, welche in die Datei **mmc-ek-tm4c1294xl.c** (im **Projekt-Ordner** unter \third\_party\fatfs\port zu finden) eingefügt werden müssen.

/\* Definition of the Boosterpack is used \*/

**#define** VS\_Webserver\_Boost

**#elif** defined(VS\_Webserver\_Boost)

/\* Peripheral definitions for EK-TM4C1294XL and EK-TM4C129EXL boards \*/

/\* SSI port \*/

**#define** SDC\_SSI\_BASE SSI3\_BASE

**#define** SDC\_SSI\_SYSCTL\_PERIPH SYSCTL\_PERIPH\_SSI3

/\* GPIO for SSI pins \*/

/\* CLK pin \*/

**#define** SDC\_SSI\_CLK\_SYSCTL\_PERIPH SYSCTL\_PERIPH\_GPIOQ

**#define** SDC\_SSI\_CLK\_GPIO\_PORT\_BASE GPIO\_PORTQ\_BASE

**#define** SDC\_SSI\_CLK GPIO\_PIN\_0

/\* TX pin \*/

**#define** SDC\_SSI\_TX\_SYSCTL\_PERIPH SYSCTL\_PERIPH\_GPIOQ

**#define** SDC\_SSI\_TX\_GPIO\_PORT\_BASE GPIO\_PORTQ\_BASE

**#define** SDC\_SSI\_TX GPIO\_PIN\_2

/\* RX pin \*/

**#define** SDC\_SSI\_RX\_SYSCTL\_PERIPH SYSCTL\_PERIPH\_GPIOQ

**#define** SDC\_SSI\_RX\_GPIO\_PORT\_BASE GPIO\_PORTQ\_BASE

**#define** SDC\_SSI\_RX GPIO\_PIN\_3

/\* CS pin \*/

**#define** SDC\_SSI\_FSS\_SYSCTL\_PERIPH SYSCTL\_PERIPH\_GPIOA

**#define** SDC\_SSI\_FSS\_GPIO\_PORT\_BASE GPIO\_PORTA\_BASE

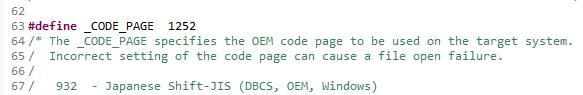
**#define** SDC\_SSI\_FSS GPIO\_PIN\_7

## Lange Filenamen aktivieren

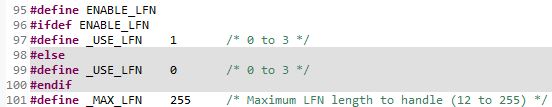
Standardmäßig lassen sich beim FatFs-Modul nur kurze Filenamen im 8.3 Zeichen Namensformat verwenden. Hierbei bestehen Dateinamen nur aus acht Buchstaben oder Ziffern gefolgt von einem Punkt und der aus drei Zeichen bestehenden Namenserweiterung. Um lange File und Verzeichnisnamen bis zu einer Zeichenlänge von 255 Zeichen verwenden zu können müssen folgende Anpassungen gemacht werden:

1. **Anpassungen in der Datei ffconfig.h** (im **SW\_ROOT-Ordner** unter \third\_party\fatfs\src zu finden)

Zunächst muss der richtige Zeichensatz eingestellt werden. In unserem Fall ist das der Latin1 Zeichensatz mit der CODE\_PAGE 1252



Anschließend muss die Unterstützung für lange Filenamen durch die Definition von ENABLE\_LFN freigeschaltet werden.



Festlegen der Sektorgröße mit 512 Byte.



1. **Anpassungen in der Datei mmc-ek-tm4c1294xl.c** (im **Projekt-Ordner** unter \third\_party\fatfs\port zu finden)

Hier müssen die folgenden zwei Funktionen implementiert werden. Sie sind für die Konvertierung der langen Filenamen zuständig.

/\*-----------------------------------------------------------------------\*/

/\* Needed no convert 8x3 filenames in long filenames \*/

/\*-----------------------------------------------------------------------\*/

WCHAR **ff\_convert** (WCHAR wch, UINT dir)

{

**if** (wch < 0x80) {

/\* ASCII Char \*/

**return** wch;

}

/\* I don't support unicode it is too big! \*/

**return** 0;

}

/\*-----------------------------------------------------------------------\*/

/\* Needed no convert 8x3 filenames in long filenames \*/

/\*-----------------------------------------------------------------------\*/

WCHAR **ff\_wtoupper** (WCHAR wch)

{

**if** (wch < 0x80) {

/\* ASCII Char \*/

**if** (wch >= 'a' && wch <= 'z') {

wch &= ~0x20;

}

**return** wch;

}

/\* I don't support unicode it is too big! \*/

**return** 0;

}

## File-APIs einfügen

Das Lesen gewünschter Files aus der SD-Karte erfolgt über die Funktionen f\_open, f\_read und f\_close. Diese müssen wie folgt in die Datei **enet\_fs.c** implementiert werden.

1. **f\_open**

Die Funktion f\_open öffnet eine Datei und erstellt ein Dateiobjekt. Das Dateiobjekt wird für die Identifikation und für nachfolgende Lese und Schreibvorgänge in die Datei verwendet. Die geöffnete Datei sollte nach dem Dateizugriff mit der Funktion f\_close geschlossen werden.

**if** (**ustrncmp**(pcName, "/", 1) == 0)

{

fresult = **f\_open**(psFatFile, pcName + 1, FA\_READ);

}

**else**

{

fresult = **f\_open**(psFatFile, pcName, FA\_READ);

}

1. **f\_read**

Die Funktion beginnt an der Position, auf die der Lese-/ Schreibzeiger zeigt, Daten aus der Datei zu lesen.

fresult = **f\_read**(psFile->pextension, pcBuffer, iCount, &uiBytesRead);

1. **f\_close**

Die Funktion schließt ein geöffnetes Dateiobjekt. Nachdem die Funktion erfolgreich ausgeführt wurde, ist das Dateiobjekt nicht mehr gültig und kann verworfen werden.

**f\_close**(psFile->pextension);

## HTML Dateien ergänzen und speichern

Es muss ein Index.htm oder Index.html Datei vorhanden sein.

Beispiele Seiten sind im Ordner „fs“ zu finden.

Speichern Sie die Dateien in der SD-Karte, und stecken Sie sie in dem Adapter ein.

## LwIP Optionen anpassen

### DHCP

Um automatisch eine IP-Adresse zugewiesen zu bekommen, braucht man ein DHCP Server. Diese Option ist in LwIP vorhanden, man muss es allerdings aktivieren.

Setzen Sie dafür den Wert von LWIP\_DHCP in lwipopts.h auf 1.

**#define** LWIP\_DHCP 1 // default is 0

### AutoIP

Eine weitere Option, die in LwIP vorhanden ist, ist AutoIP. Diese ermöglicht die Zuweisung von einer statische IP-Adresse, die aus der MAC-Adresse berechnet wird.

Setzen Sie den Wert von LWIP\_AUTOIP auf 1, um die Option zu aktivieren.

**#define** LWIP\_AUTOIP 1 // default is 0

### Weitere Optionen

Die Applikation wartet eine Zuweisung der IP-Adresse, entweder vom DHCP Server oder von AutoIP. Um diese Wartezeit zu reduzieren, kann man die Wiederholungen der DHCP Server Proben verkleinern, indem man den Wert von LWIP\_DHCP\_AUTOIP\_COOP\_TRIES in lwipopts.h ändert.

**#define** LWIP\_DHCP\_AUTOIP\_COOP\_TRIES 1 // default is 9

# Test & Lösung

## Schritte

* Firmware auf dem MCU programmieren
* Ethernet Kabel und SD-Karte Adapter einstecken.
* Auf die vom DHCP oder AutoIP zugewiesene IP-Adresse warten. Die IP-Adresse wird nach einiger Sekunden auf dem Display gezeigt werden.
* IP-adresse im Browser eingeben.

# Optionale Features

Um zusätzliche Funktionalitäten in dem Projekt hinzuzufügen, Beispielsweise eine LED Steuerung, folgen Sie folgende Schritte.

## GPIO Port konfig

Konfigurieren Sie die Ports, wo die LED angeschlossen ist. Dies sollte am Anfang des Programablaufs sein.

## GPIO Handler Implementierung

Implementieren Sie einen Handler, den Sie später aufrufen können, um die LED anzusteuern.

## HTTP-GET Request (javascript)

Implementieren Sie eine Javascript Funktion, die ein HTTP-GET Request generieren kann. Die Funktion muss in einer verbundenen Javascript-Datei implementiert werden und würde wie folgt aussiehen:

**function** getSensorData() {

**var** req = false;

**function** callbackFunction() {

**if** (req.readyState == 4) {

**if** (req.status == 200) {

// TODO: Handel Response Here

}

}

}

**if** (window.XMLHttpRequest) {

req = **new** XMLHttpRequest();

}

**else** **if** (window.ActiveXObject) {

req = **new** ActiveXObject("Microsoft.XMLHTTP");

}

**if** (req) {

// TODO: Set the appropriate URI for the GET Request

req.open("GET", "/get\_smth?id=" + Math.**random**(), true);

req.onreadystatechange = callbackFunction;

req.send(null);

}

}

## HTTP-GET Handler (fs-open)

Fügen Sie eine weitere Option zur Verarbeitung Ihrer generierten HTTP-GET Request, indem Sie eine weitere IF-Anweisung in der Funktion fs\_open(), die in der Datei enet\_fs.c

// TODO: Change the URI accordingly to what you wrote in the GET Request

**if** (ustrncmp(pcName, "/get\_smth", 9) == 0)

{

**static** **char** pcBuf[5];

// TODO: Write the appropriate Response in the Buffer pcBuf using

usnprintf(pcBuf, 5, "%d", someValue)

// Or this

convertINTtoCHAR(someValue, pcBuf);

psFile->data = pcBuf;

psFile->len = strlen(pcBuf);

psFile->index = psFile->len;

psFile->pextension = NULL;

**return** (psFile);

}