Jürgen Pauritsch und Stefan Thonhofer

# Embedded Webserver

## Einleitung

Ziel unseres Projekts war es, einen Webserver auf einer einzigen Platine („Embedded system“, „System on a chip“) aufzusetzen.

Der Vorteil derartiger Systeme liegt darin, dass sich der gesamte Computer platzsparend auf einer einzigen Platine befindet, wenig Strom verbraucht, im Normalfall nicht zusätzlich gekühlt werden muss und deshalb geräuscharm läuft.

Für die Umsetzung haben wir uns für das sogenannte **DevKit 8000** der chinesischen Firma Embest entschieden – und nicht für das teurere und weiter verbreitete **beagleboard** – weil hier schon bei der Standardkonfiguration ein Netzwerkanschluss dabei war.


*Abbildung 1: Größenvergleich DevKit zu Scheckkarte*

## Technische Daten des Embest DevKit8000 Evaluation Kit (Standard Configuration) :

| | |
|---|---|
| Prozessor | TI OMAP 3530 @ 600MHz |
| RAM | 128MB DDR SDRAM, 166MHz |
| Flashspeicher | 128MB NAND Flash, 16bit |
| Wichtigste Schnittstellen | SD-Card-Slot, 1 USB-Port, Ethernet-Anschluss, HDMI-Ausgang, serielle Schnittstelle |
| Energieversorgung | 5V |
| Abmessungen | 110mm x 950mm x 20mm |
| Inkludiertes Zubehör | 512MB SD-Karte, Netzgerät mit Adapter für mitteleuropäische Steckdosen, serielles Kabel, Software-CD mit Windows CE und Angstrom Linux |

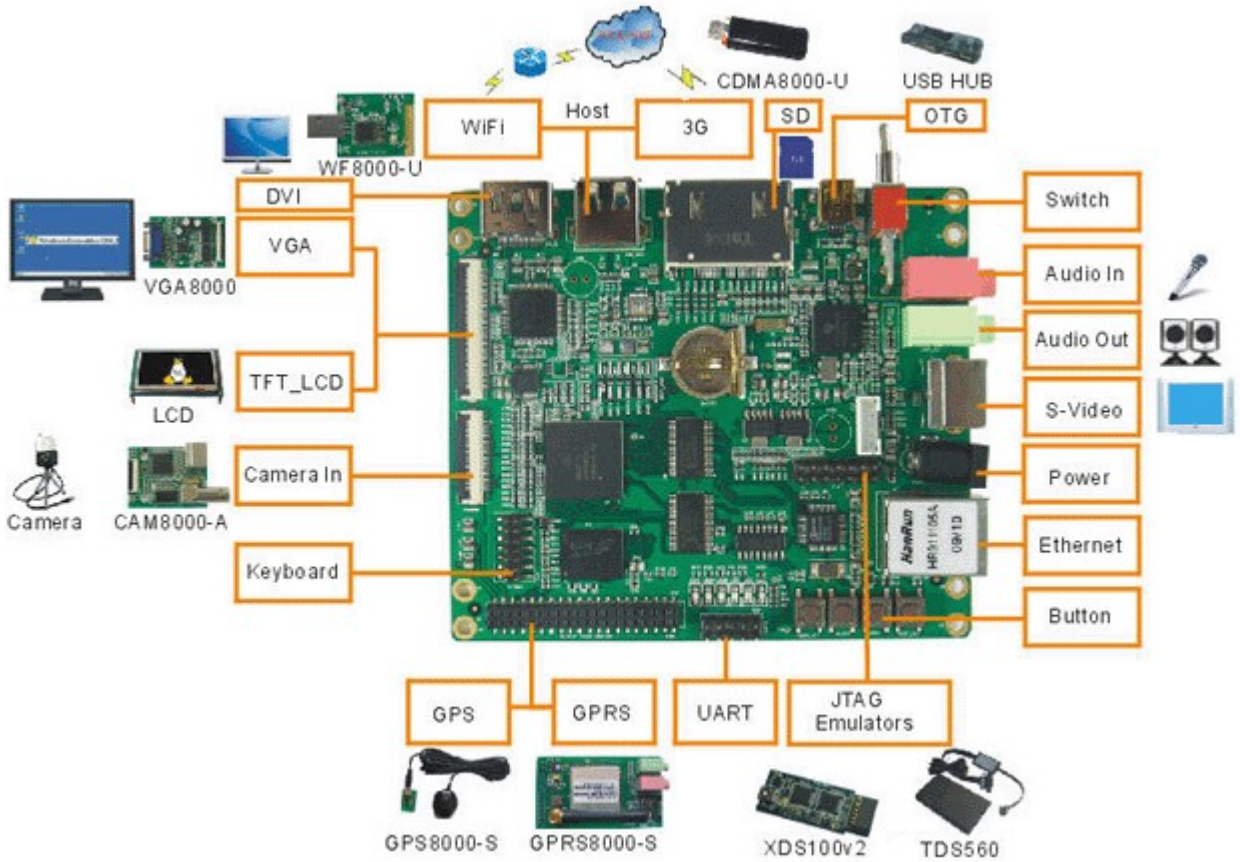



*Abbildung 2: Beschreibung der Buttons und Anschlüsse*

## Zusätzliche Hardware

- Adapter: serielle Schnittstelle zu USB

- USB-Hub mit zusätzlicher Stromversorgung (zum Anschließen von eigener Maus und Tastatur)
- HDMI-Kabel für Bildschirmanschluss

## Benötigte Software

- Gparted
- Minicom
- Angstrom-Linux

## Aufsetzen

### *Vorbereitung*

Bevor wir uns dem **DevKit** selbst widmen, sollte einmal getestet werden, ob der serielle Anschluss des Computers über den wir unser Embedded System aufsetzen, überhaupt funktioniert. Dazu schließt man das serielle Kabel an (in unserem Fall über den USB-Adapter) und testet mit dem Befehl `dmesg | grep tty` ob der Anschluss erkannt wurde. Wenn ja, startet man `minicom`, welches eine Konsole über die serielle Schnittstelle herstellt. Zuerst testeten wir, ob das Senden und Empfangen generell möglich ist. Dazu schlossen wir die Pins TX und RX mit einem Kupferdrahtstück zusammen. In `minicom` eingegebener Text wurde dann sofort wieder ausgegeben, d.h. die serielle Schnittstelle funktioniert.

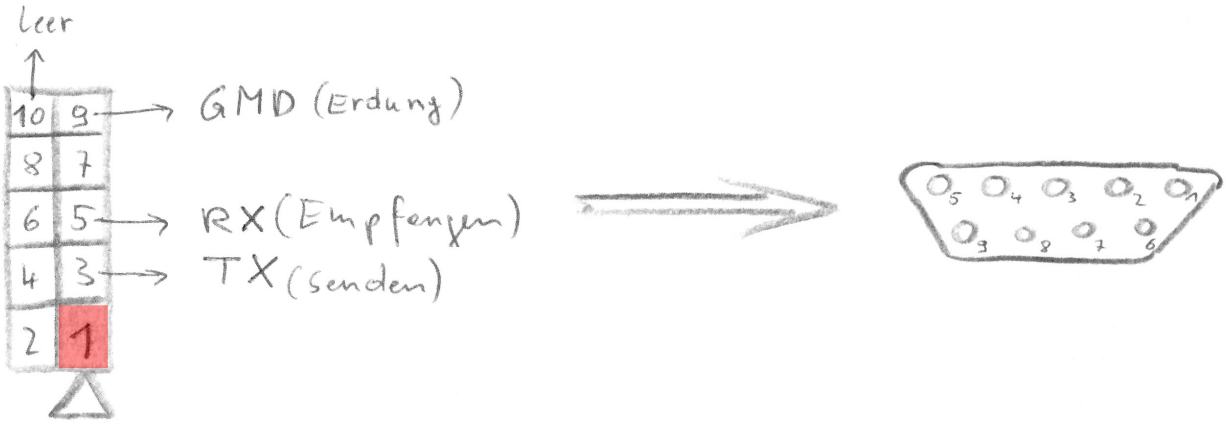
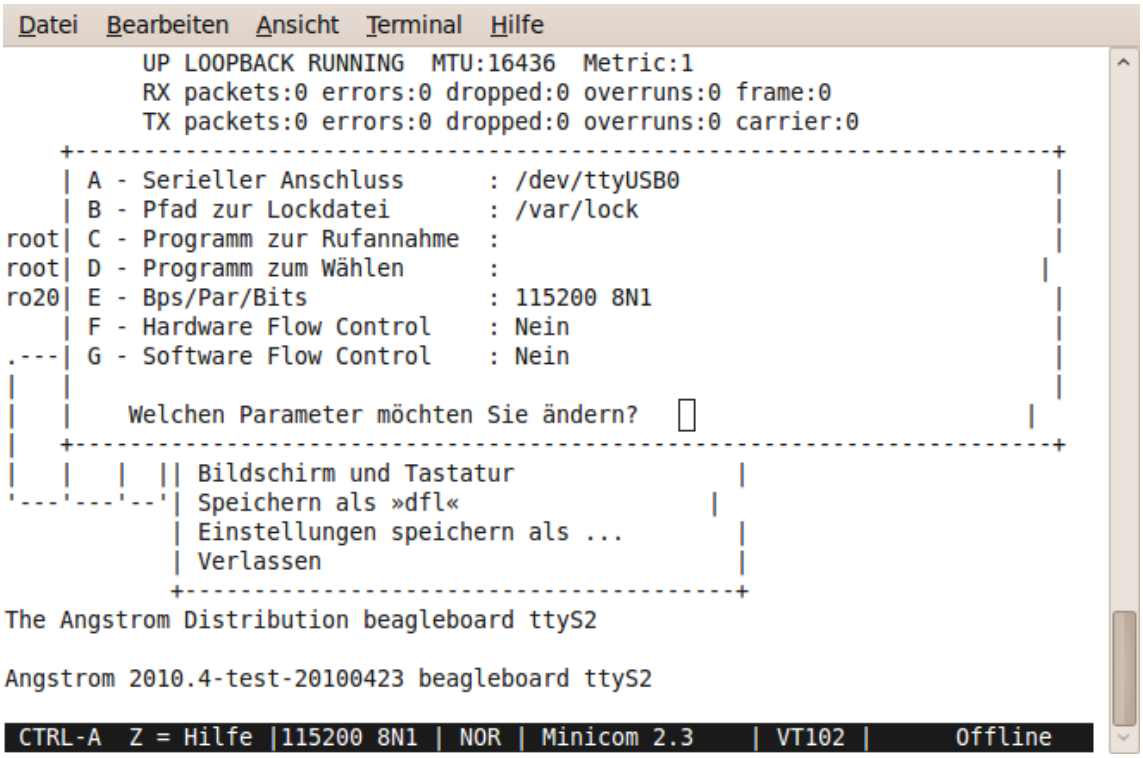



*Abbildung 3: Zuordnung von Pins serieller Schnittstellen*

Um nun über `minicom` eine serielle Verbindung zum **DevKit** herzustellen, mussten wir die `minicom`-Konfiguration abändern. In der Einstellungsseite (Abbildung 4) veränderten wir die Punkte A und F, da wir einen USB-seriell-Adapter verwendeten.




Danach schlossen wir unser **DevKit** an und schalteten es ein, um zu überprüfen, ob das vorinstallierte **Angstrom-Linux** aus dem NAND-Speicher bootet. Das funktionierte soweit, allerdings bietet dieses vorinstallierte **Angstrom-Linux** bei weitem nicht alle Pakete an, die für einen vernünftigen Betrieb nötig sind.
Im folgenden Punkt wird darauf eingegangen, wie man sich selbst ein geeignetes **Angstrom-Linux**-Image erstellt und installiert.

## *SD-Karte formatieren, partitionieren und OS installieren*

Der Speicher der SD-Karte wird auf eine 50MB große FAT32- und eine EXT2-Partition aufgeteilt. Hierfür verwendeten wir das Tool `gparted`.

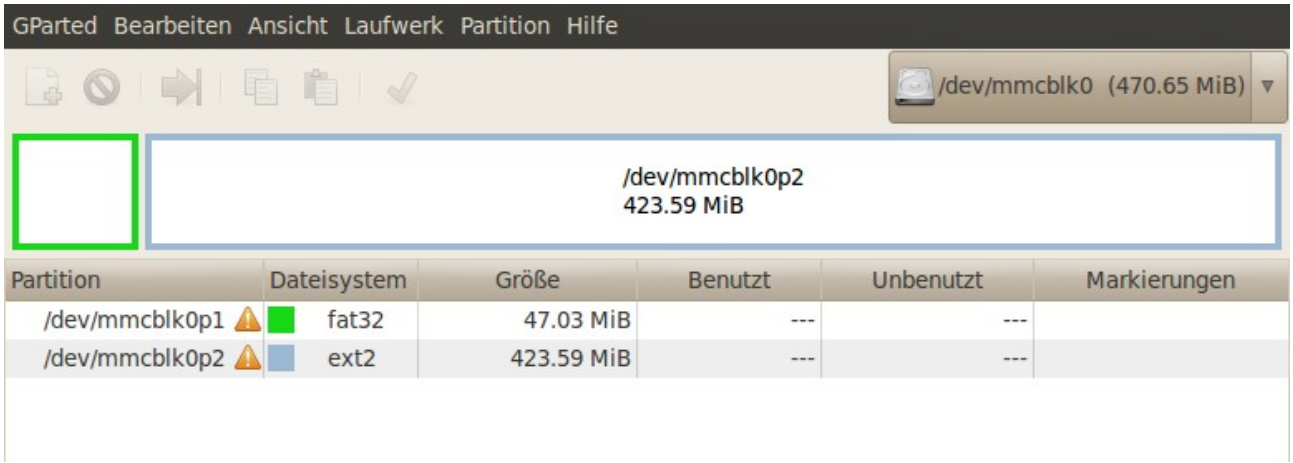



*Abbildung 5: Partitionierung mit gparted*

Auf die „frisch“ formatierte FAT32-Partition kopiert man nun zuerst einmal die Datei **MLO**

(muss am ersten Sektor beginnen) und in weiterer Folge das **uImage**, was als Bootloader fungiert. Beide sind auf der enthaltenen CD-ROM zu finden, die „neuesten“ Versionen, die frei online verfügbar sind, funktionierten bei uns nicht.

Auf der **Angstrom**-Homepage erstellten wir letztendlich eine Distribution für das Beagleboard (hat die gleiche Architektur wie unser **DevKit**) mit den von uns erwünschten Paketen.

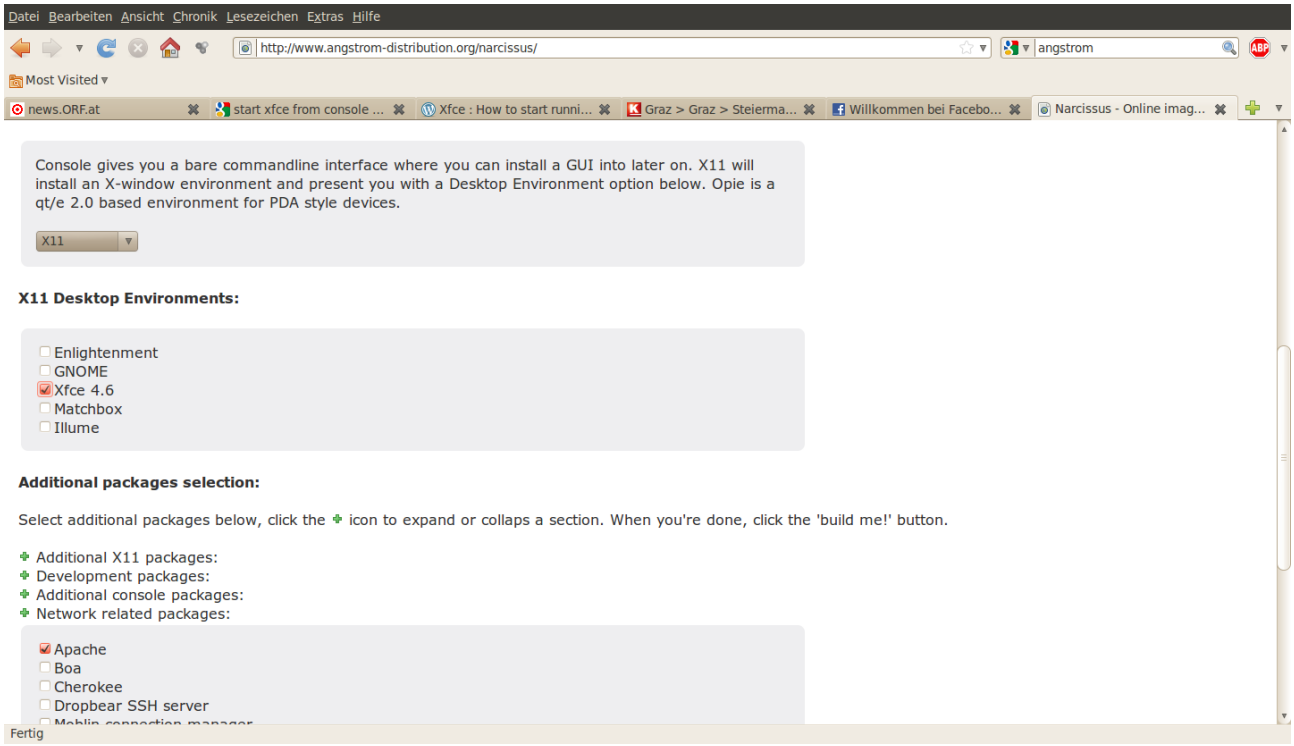



*Abbildung 6: Image erstellen auf http://www.angstrom-distribution.org/narcissus/*

## Inbetriebnahme

Damit standardmäßig unser Image von der SD-Karte bootet, muss man noch 2 Bootoptionen setzen. Dazu muss während des Bootvorganges ~~der ANY key~~ eine beliebige Taste gedrückt werden. Gibt man nun `printenv` ein, werden die einzelnen Optionen angezeigt. Mit `setenv` kann man eine Änderung vornehmen.

In unserem Fall:

- `setenv bootcmd 'mmcinit; fatload mmc 0:1 0x80300000 uImage; bootm 0x80300000`
- `setenv bootargs 'console=ttyS2, 115200n8 console=tty0 root=/dev/mmcblk0p2 rootdelay=2 rootfstype=ext2 rw'`

Ist in der Eingabekonsole das Zeilenende erreicht, werden die getippten Zeichen nicht mehr angezeigt, aber trotzdem übernommen. Deshalb besonders auf Tippfehler achten!

Nach dem Speichern dieser Einstellungen (Befehl: `saveenv`) wird permanent mit dieser

Konfiguration gestartet.

Damit hatten wir ein funktionierendes **Linux**-Betriebssystem bei dem standardmäßig der **DHCP-Client** und der **Apache-Webserver** bereits liefen. Dies konnten wir einfach in einem privaten Netzwerk testen:

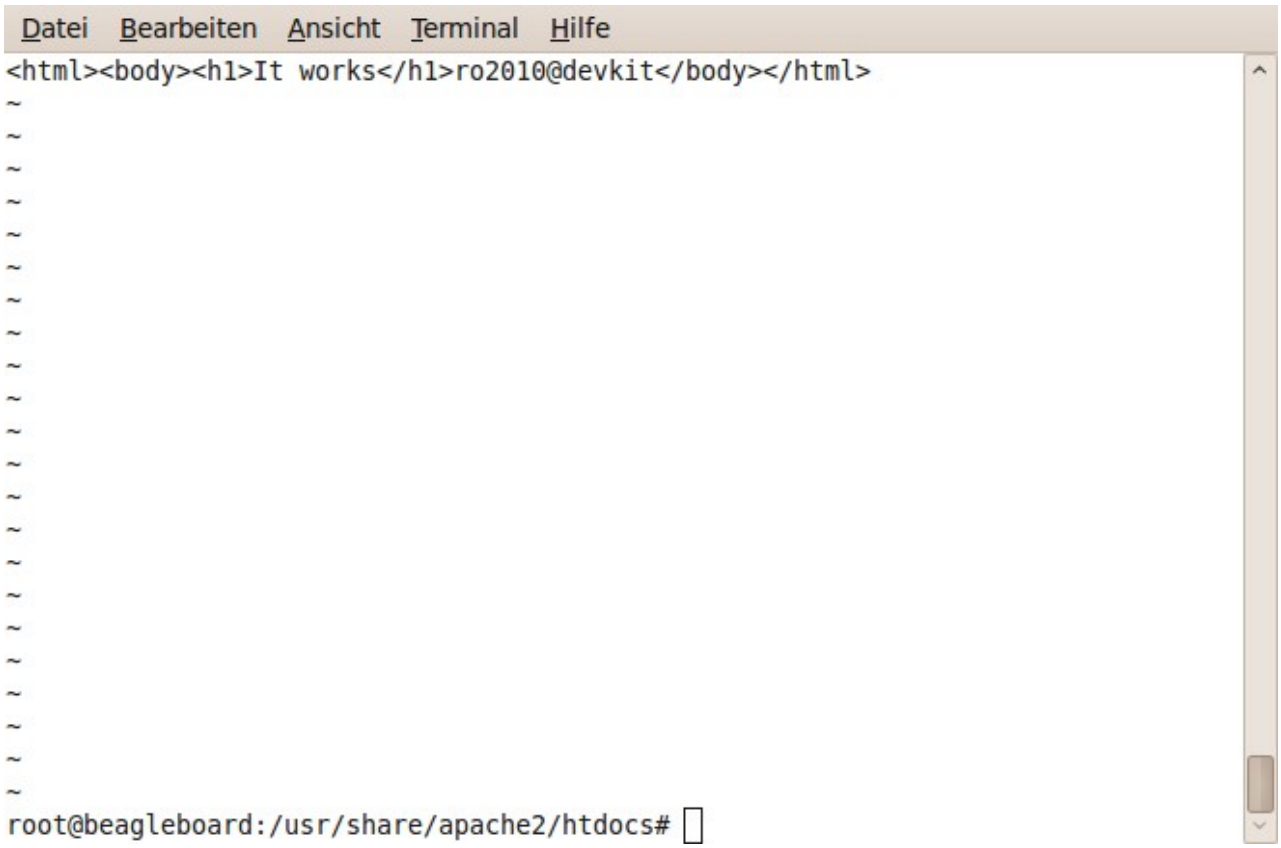



*Abbildung 7: HTML-Code auf dem Webserver*

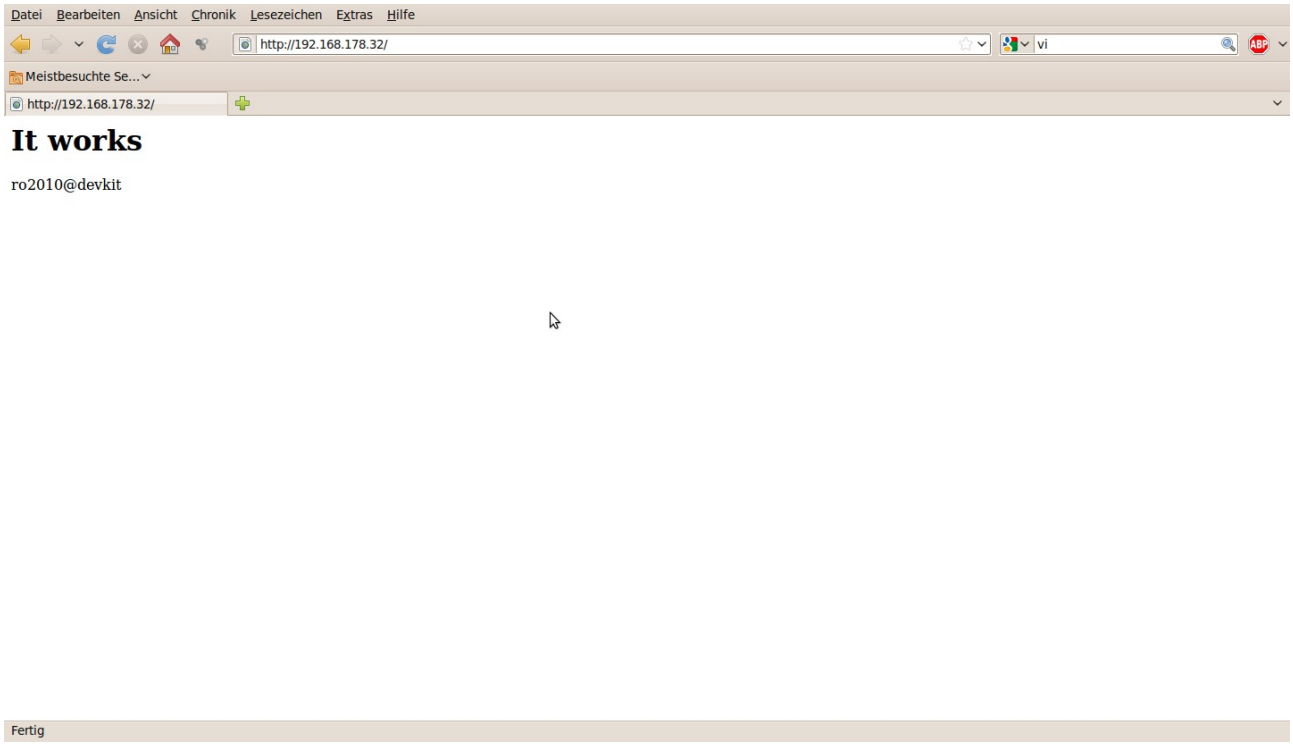



*Abbildung 8: HTML-Ansicht auf einem Client*

Weiters funktionierte problemlos der Login über `ssh`.

Damit kann man den Embedded Webserver in ein bestehendes Netzwerk einbinden und über ssh warten.