Contiki Tutorial - praktischer Teil 1

Sven Zehl, Thomas Scheffler

Beuth Hochschule für Technik Berlin

26. September 2013

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Contiki

The Open Source OS for the Internet of Things

-Einführung-

Contiki

The Open Source OS for the Internet of Things

- **AVR Studio** zum Upload auf den Mikrocontroller mithilfe des Programmers JTGiceMKII.
- **Cygwin**, eine Software die es ermöglicht Unix Programme unter Windows zu Nutzen. Mit Cygwin und dem Programm **make** sowie dem Compiler + Toolchain **avr-gcc** (unter Windows WinAVR) wird der Quellcode übersetzt.

Cygwin

1. Cygwin starten und in den Contiki Unterordner `/examples/` wechseln (**cd [Ordnername]** öffnet Verzeichnis, **cd ..** springt zurück)
2. neuen Projektordner erstellen mit **mkdir projekt1**, anschließend in den neuen Ordner wechseln (**cd projekt1**)
3. benötigte Dateien anlegen mit **touch projekt1.c Makefile Makefile.projekt1**
4. jetzt kann mit dem Windows Explorer zum Pfad navigiert werden (C:/Cygwin/home/[Benutzername]/contiki-2.6/examples/projekt1/) und die Dateien können mit einem Editor der Wahl bearbeitet werden.
5. Alternative: direkt unter Cygwin mit dem Kommandozeileneditor **vim** arbeiten, dazu einfach unter Cygwin den Befehl **vi [Dateiname]** ausführen.

Contiki Prozesse - benötigte Bibliotheken

1. die neu erstellte Datei **projekt1.c** mit dem Editor der Wahl öffnen.
2. benötigte Bibliotheken einbinden.

```
1 #include "contiki.h"
2 /*enthält alle Contiki spezifischen Makros und Funktionen*/
3 #include <stdio.h>
4 /*Standard Input Output z.B. für printf()*/
5 #include <avr/io.h>
6 /*Input Output speziell für AVR Microcontroller, z.B. die
    Makros für die Ports und Register */
```

Contiki Prozesse - Prozessdefinition & Autostart

1. Mithilfe der Anweisung **PROCESS([prozess_name], "[Prozess Beschreibung]");** wird der neue Prozess dem Betriebssystem bekannt gemacht.
2. Es soll ein Prozess generiert werden, welcher beim Bootvorgang eine LED des Entwicklungsboards einschaltet.
3. Dieser soll automatisch nach dem Bootvorgang gestartet werden. Dazu muss der neue Prozess in die Autostartliste eingetragen werden. Dies geschieht über die Anweisung **AUTOSTART_PROCESSES(&[prozess_name]);**. Wichtig ist hierbei, dass die Adresses des Prozesses eingetragen werden muss.

```
1 PROCESS(switch_led_on, "LED einschalten nach Booten");
2 AUTOSTART_PROCESSES(&switch_led_on);
```

Contiki Prozesse - Prozessaufbau

1. Es soll nun der eigentlich Prozess programmiert werden.
2. Prozesse beginnen in Contiki immer mit der Kopfzeile **PROCESS_THREAD([prozess_name], ev, data)**
3. Der folgende Rumpf des Prozesses wird immer durch das Makro **PROCESS_BEGIN()**; begonnen.
4. Das Ende des Prozesses wird immer durch das Makro **PROCESS_END()**; bestimmt.
5. zwischen **PROCESS_BEGIN()** und **PROCESS_END()** wird der eigentlich Quellcode des Programms geschrieben.

```
1 PROCESS_THREAD(switch_led_on, ev, data)
2 {
3 PROCESS_BEGIN();
4
5 /*hier kommt der eigentlich Programmcode hin*/
6
7 PROCESS_END();
8 }
```

Contiki Prozesse - Prozessaufbau - Port PIN Aktivierung

1. unter **https://wiki.ipv6lab.beuth-hochschule.de/_media/contiki/a-n-solutions-module/brick_mcu_pinassignments_ref2.pdf** kann der Schaltplan und die Port Belegung des Entwicklungsboards heruntergeladen werden.
2. Der Programmcode soll beim Start die LED DS3 einschalten, diese ist im Schaltplan auf Seite 6 zu finden.
3. der GND von LED DS3 ist intern an Port GPI02 angeschlossen, GPI02 entspricht Port PB7 des Atmega1281 siehe Seite 3.
4. um nun herauszufinden wie Port PB7 des 1281 auf GND geschaltet werden kann, wird das Datenblatt des ATmega1281 benötigt. Es kann unter **http://www.atmel.com/Images/doc2549.pdf** herunterlgeladen werden.

1. In Kapitel 13.2.1 (Seite 71) ist zu lesen, dass der PIN zuerst als Ausgangspin geschaltet werden muss um überhaupt als Ausgang fungieren zu können. *If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.*
2. Da wir den PIN als Ausgang nutzen wollen muss Pin DDB7 im Register DDRB zu logisch Eins gesetzt werden. Da Register nur byteweise beschrieben werden können und die restlichen Bits nicht verändert werden sollen, wird eine **bitweise ODER** Anweisung verwendet.

```
1 DDRB |= (1 << PIN7);
2 //entspricht DDRB = DDRB | 0b10000000
3 //(1<< PIN7) = (0b00000001 7x links-shift) = 0b10000000
```

1. Um den PIN7 von PortB nun auf Null Pegel zu setzen muss wieder das Kapitel 13.2.1 (diesmal Seite 72) des Datenblatts herangezogen werden. *If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one)*
2. Demnach muss also Pin7 von Register PORTB auf Null gesetzt werden (Es soll ja ein GND Potential erzeugt werden, Anode der LED liegt immer auf VCC, siehe Schaltplan). Um auch hier wieder keine anderen Pins zu verändern wird diesmal das **bitweise UND** verwendet.

```
1 PORTB &= ~(1 << PIN7);
2 //entspricht PORTB = PORTB & 0b01111111
3 //(1<< PIN7) = (0b00000001 7x links-shift) = 0b10000000
4 //durch das ~ Zeichen werden alle Bits invertiert
5 //zu 0b01111111
```

Contiki Prozesse - Prozessaufbau

1. Die beiden Anweisungen werden nun in den Prozess integriert.
2. Der fertige Quellcode inklusive Bibliotheken und Prozessdefinition und Autostarteintrag sieht dann aus wie folgt:

```
1  #include "contiki.h"
2  #include <stdio.h>
3  #include <avr/io.h>
4
5  PROCESS(switch_led_on, "LED einschalten nach Booten");
6  AUTOSTART_PROCESSES(&switch_led_on);
7
8  PROCESS_THREAD(switch_led_on, ev, data)
9  {
10 PROCESS_BEGIN();
11 /*Pin als Ausgang schalten*/
12 DDRB |= (1 << PIN7);
13 /*Pin auf Low Pegel schalten*/
14 PORTB &= ~(1 << PIN7);
15
16 PROCESS_END();
17 }
```

Contiki Prozesse - Makefile

1. Das erste Programm ist damit fertig geschrieben, nun die Datei **projekt1.c** abspeichern und die Datei **Makefile** im Editor der Wahl öffnen.
2. Makefiles werden vom Programm **make** gelesen, Contiki besitzt mehrere Makefiles, welche alle miteinander verknüpft sind und durch bestimmte vom Benutzer gesetzen Variablen die passenden Quellcodedateien von Contiki zur Kompilierung an den Compiler weiterleitet (z.B. passende Dateien für den verwendeten Mikrocontroller, passende Dateien für den passenden Funkchip, passende Dateien für IPv6 statt IPv4 usw...)

Contiki Prozesse - Makefile

1. Um **make** mitzuteilen was es zu tun hat, wird zuerst der Fall **all:** festgelegt, dieser wird immer ausgeführt, wenn make ohne weitere Anweisungen gestartet wird (es können später weitere Fälle festgelegt werden).
2. In diesem Fall wird immer die darauffolgende Zeile ausgeführt, welche immer mit einem **Tabulator** Abstand eingeben werden muss.
3. über **TARGET=avr-zigbit** wird die Zielplattform festgelegt, und über **-f** ein weiteres Makefile eingebunden, das **Makefile.projekt1**, welches zu Beginn erstellt wurde.
4. mit **projekt1.elf** wird anschließend die Zieldatei festgelegt.
5. das **Makefile** kann nun abgespeichert und sogleich das zweite Makefile, **Makefile.projekt1** geöffnet werden.

```
1 all:
2  ${MAKE} TARGET=avr-zigbit -f Makefile.projekt1 projekt1.elf
```

Contiki Prozesse - Makefile.projekt1

1. Wir befinden uns nun in der Datei **Makefile.projekt1**.
2. In der ersten Zeile wird festgelegt wo sich das Wurzelverzeichnis von Contiki befindet (zwei Ordner oberhalb des derzeitigen Verzeichnisses).
3. die zweite Zeile des Makfiles, legt **IPv6** anstelle von IPv4 als Kommunikationsprotokoll fest.
4. die letzte Zeile sorgt dafür, dass das globale Contiki **Makefile.include**, welches im obersten Contiki Verzeichnis liegt mit eingebunden wird.
5. nach dem Abspeichern, muss wieder in die **Cygwin** Console und weiter in das Verzeichnis von **projekt1** gewechselt werden.

```
1 CONTIKI = ../..
2 UIP_CONF_IPV6=1
3 include $(CONTIKI)/Makefile.include
```

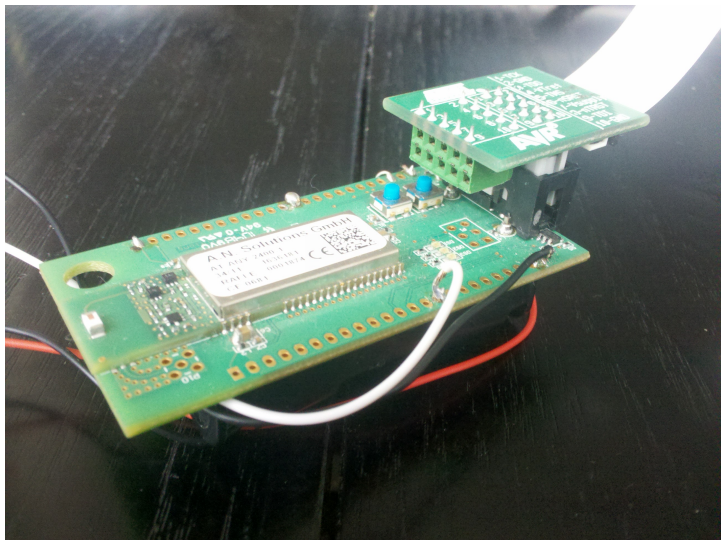
Contiki Prozesse - make und upload

1. Wir befinden uns nun in **Cygwin** und im Ordner **projekt1**
2. Der Befehl **make** startet nun den Kompilierungsvorgang und die Erzeugung des **.elf-Files**.
3. War alles erfolgreich, so sollte automatisch das Programm **avr-size** gestartet werden, welches wiederum folgenden Output liefern sollte:

```
AVR Memory Usage
----------------
Device: atmega1281

Program:   48546 bytes (37.0% Full)
(.text + .data + .bootloader)

Data:       3961 bytes (48.4% Full)
(.data + .bss + .noinit)

EEPROM:        8 bytes (0.2% Full)
(.eeprom)
```

Contiki Prozesse - make und upload

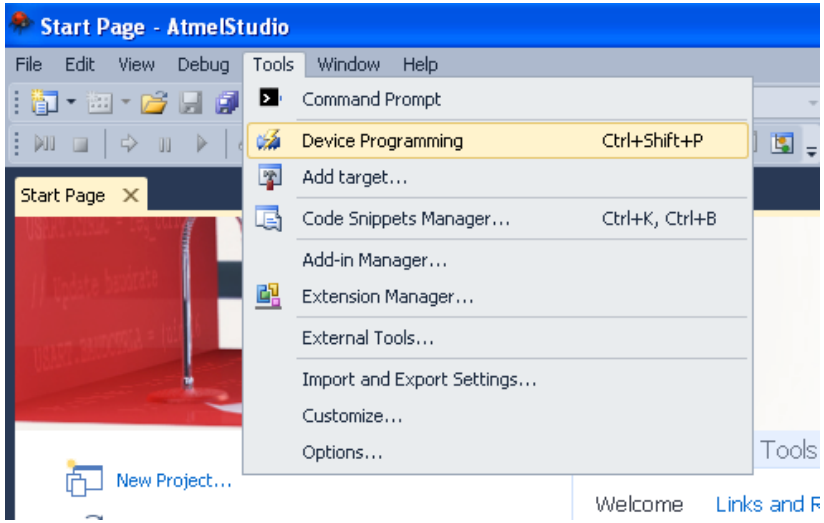
1. Jetzt folgt der Upload auf den Mikrocontroller, dazu den Programmer JTAGiceMKII wie gezeigt mit Entwicklungsboard verbinden:



2. Das Entwicklungsboard sowie den Programmer einschalten und im Anschluss das AVR Studio starten.

Contiki Prozesse - make und upload

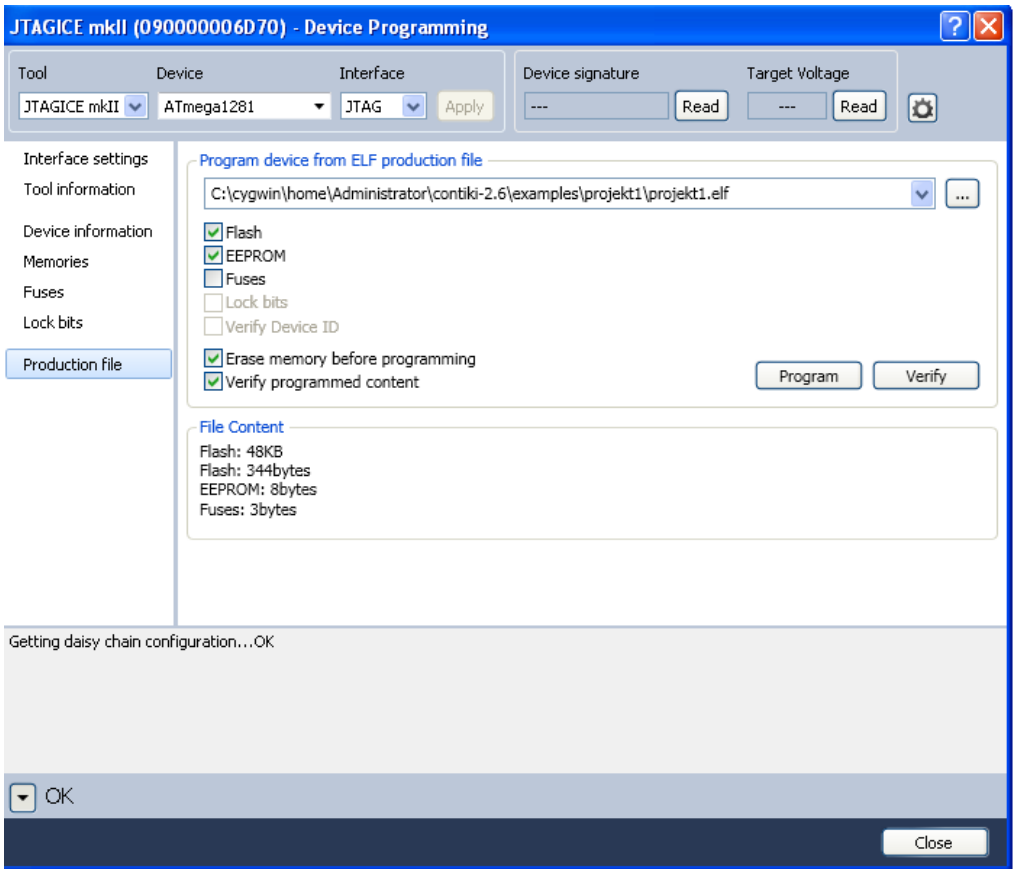
1. Im AVR Atmel Studio im Menü oben auf Tools / Device Programming klicken (siehe folgende Abbildung):



2. Im nun folgenden Dialogfenster, die Einstellungen für den Mikrocontroller wählen und mit *apply* bestätigen(siehe Abbildung unten).

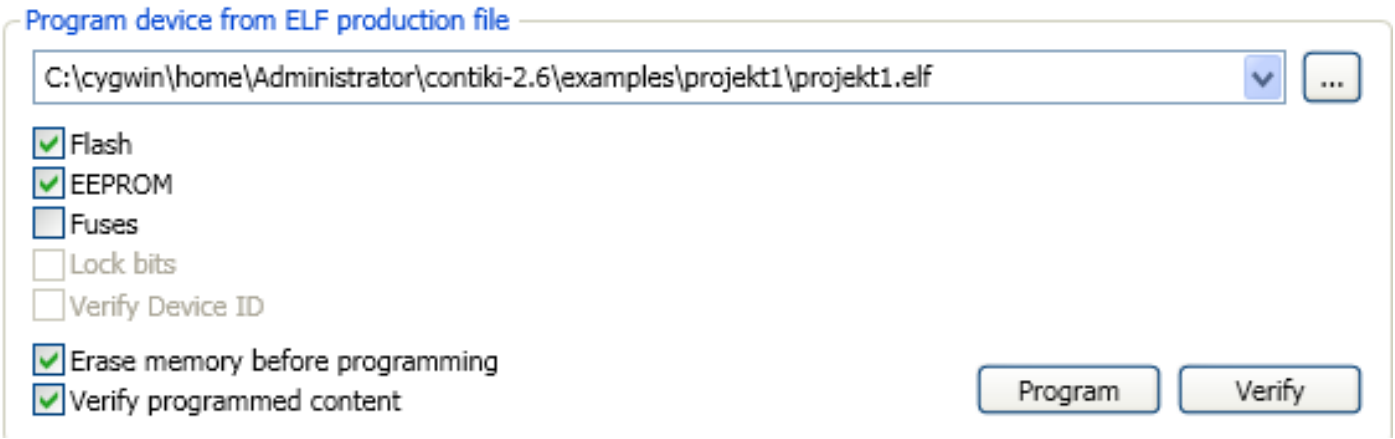
Contiki Prozesse - make und upload

1. Da die zuvor generierte .elf-Datei hochgeladen werden soll, muss nun im linken Auswahlbereich *Production file* gewählt werden. (siehe folgende Abbildung):



Contiki Prozesse - make und upload

1. Weiter muss nun über den Button [...] zum generierten .elf-File navigiert werden bzw. der Pfad eingetragen werden. Außerdem müssen die Haken für *Flash* und *EEPROM* gesetzt werden (siehe folgende Abbildung):



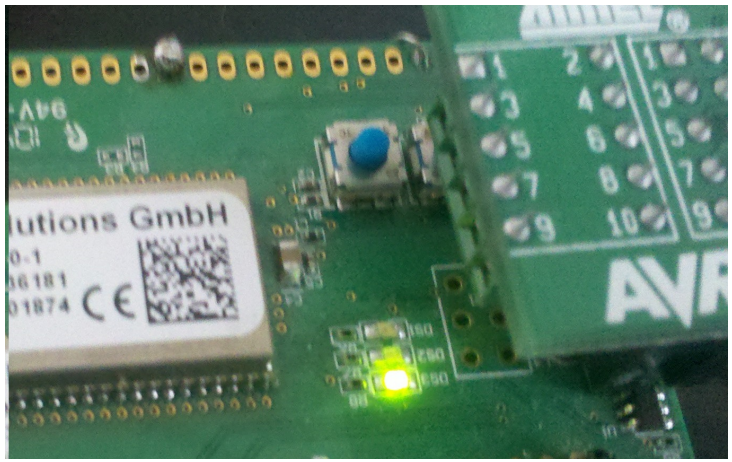
2. Mit einem letzten Klick auf *Program* wird anschließend der Upload gestartet.

Contiki Prozesse - make und upload

1. War der upload erfolgreich, so erscheinen die unten gezeigte Schritte mit *OK* bestätigt.

```
Erasing device... OK
Programming Flash...OK
Verifying Flash...OK
Programming EEPROM...OK
Verifying EEPROM...OK
```

2. Beim Betrachten des Entwicklungsboards sollte außerdem jedem ein Licht aufgehen.



Quellen

- https://github.com/contiki-os/contiki/wiki
- http://contiki.sourceforge.net/docs/2.6/
- http://senstools.gforge.inria.fr/doku.php?id=contiki

$\Rightarrow$ alle Beispiele (inklusive Sender Quellcodes für Linux) sind unter https://github.com/szehl/contiki_tutorial zu finden.