|  |
| --- |
| Entwicklung und Integration eines USB-Treibers für den Datenlogger „KlimaLogg Pro“  **Integrationsprojekt CAS EBX** |
| Studiengang: CAS Embedded Linux and Android  Autor: Christian Binder, Daniel Reimann, Urs Suhner  Betreuer: Phillipe Seewer  Datum: 30.09.2015 |

Inhaltsverzeichnis

[1 Projektbeschreibung 3](#_Toc431238719)

[2 Rahmenbedingungen 4](#_Toc431238720)

[3 Reverse Engineering 5](#_Toc431238721)

[3.1 Sequenzdiagramm Python Treiber Initialisierung 5](#_Toc431238722)

[3.2 Sequenzdiagramm Python Treiber Kommunikation 6](#_Toc431238723)

[3.3 Datenbeschreibung Messwerte 7](#_Toc431238724)

[3.3.1 USB Frame Header 7](#_Toc431238725)

[3.3.2 USB Current Data 8](#_Toc431238726)

[3.3.3 History Data 9](#_Toc431238727)

[4 MASCOT Grobdesign 10](#_Toc431238728)

[4.1 Diagramm 11](#_Toc431238729)

[4.2 Aktivitätsbeschreibung 11](#_Toc431238730)

[4.2.1 Anzeige-Prozess A\_01 11](#_Toc431238731)

[4.2.2 Arbeiter-Prozess A\_02 11](#_Toc431238732)

[4.2.3 Treiber A\_03 11](#_Toc431238733)

[4.2.4 Uhr A\_04 12](#_Toc431238734)

[4.2.5 Timer A\_05 12](#_Toc431238735)

[4.3 Channel- Beschreibung 12](#_Toc431238736)

[4.3.1 User Interaction Channels C\_01/C\_02 12](#_Toc431238737)

[4.3.2 Error Auswertung C\_03 12](#_Toc431238738)

[4.3.3 Treiber Daten lesen C\_04a 12](#_Toc431238739)

[4.3.4 Treiber Daten schreiben C\_04b 12](#_Toc431238740)

[4.3.5 USB Tranceiver Daten lesen C\_05a 12](#_Toc431238741)

[4.3.6 USB Tranceiver Daten schreiben C\_05b 12](#_Toc431238742)

[4.3.7 Timer Response C\_06a 12](#_Toc431238743)

[4.3.8 Timer Setzen C\_06b 12](#_Toc431238744)

[4.4 Pool Beschreibung 12](#_Toc431238745)

[4.4.1 Datenbank P\_01 12](#_Toc431238746)

[4.4.2 Zeitstempel P\_02 12](#_Toc431238747)

[5 Abgabe Strukur 13](#_Toc431238748)

[6 InstallationsAnleitung 14](#_Toc431238749)

[Schlussfolgerungen/Fazit 16](#_Toc431238750)

[7 Abbildungsverzeichnis 17](#_Toc431238751)

[8 Tabellenverzeichnis 17](#_Toc431238752)

# Projektbeschreibung

Das Integrationsprojekt CAS EBX basiert auf einen Datenlogger der Firma TFA vom Typ „KlimaLogger Pro“. Mit diesem Datenlogger können bis zu 8 Aussensensoren und einem Innensensor angeschlossen werden. Die Aussensensoren werden über Funk angesprochen und zeichnen Temperatur oder Temperatur und Feuchtigkeit auf. Der Innensensor ist im Datenlogger verbaut und misst ebenfalls Temperatur und Feuchtigkeit. In einem vorangehenden Projekt wurde dieser Datenlogger bereits eingesetzt um eine Taupunkt Lüftungssteuerung zu realisieren. Für die Taupunkt Lüftungssteuerung wurde ein Python Treiber eingesetzt um die Messdaten von dem Datenlogger zu erhalten. Im jetzigen Integrationsprojekt soll dieser Python Treiber durch ein von uns entwickelter Linux Treiber ersetzt werden. Der Source Code des Python Treibers ist vorhanden und dient somit als Grundlage der Entwicklung. Für die Darstellung der Messdaten wird ein QT-GUI verwendet, welche die Historie der Messdaten aufzeigt. Zur einfacheren Analyse der Daten soll der darzustellende Zeitraum wählbar sein.

# Rahmenbedingungen

|  |  |
| --- | --- |
| Bedingung | erfüllt |
| Realisierung auf/für eine bestehende Hardware: Falls Hardware noch evauliert oder zusammengebaut werden muss, wird das Thema abgelehnt. | Ja |
| Realisierung auf/für ein bestehendes Linux-Umfeld: Ein angepasster/portierter Kernel, sowie notwendige Infrastruktur (rootfs, etc) sollte minimal bestehend und falls notwendig nur angepasst werden müssen. | Ja |
| Die Lösung muss zwingend die erwähnten Beurteilungspunkte enthalten: Kooperation/Synchronisation/Design, sowie Userspace und Kernel-Komponenten. | Ja |

Tabelle 1: Rahmenbedingungen

# Reverse Engineering

Wie in der Einleitung erwähnt stand nur ein Python Treiber als Grundlage zur Verfügung. Der Python Treiber musste erstmals in seinem Vorgehen verstanden werden. Für die Darstellung des Ablaufs eignet sich ein Sequenz Diagramm:

## Sequenzdiagramm Python Treiber Initialisierung

Abbildung 1: Sequenzdiagramm Treiber Initialisierung

## Sequenzdiagramm Python Treiber Kommunikation



Abbildung 2: Sequenzdiagramm Treiber Kommunikation

## Datenbeschreibung Messwerte

Die Messwerte können als History Data und als CurrentData übermittelt werden. CurrentData ist ein Momentanwert ohne Zeitstempel. Ein History Data Packet beinhaltet bis zu 6 Messresultate. Ein Messresultat besteht aus Daten von insgesamt 9 Sensoren.

Die Messwerte als solches werden in 4 bit Einheiten übetragen. Es gibt jeweils 4 bit pro Dezimalstelle.

### USB Frame Header

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **MSB** |  |  |  |  |  |  | **LSB** |
| **0** |  | | | | | | | |
| **1** |  | | | | | | |  |
| **2** | ByteCount | | | | | | | |
| **3** | BufferId High | | | | | | | |
| **4** | BufferId Low | | | | | | | |
| **5** | Logger ID | | | | | | | |
| **6** | ResponseType | | | | | | | |
| **7** |  | SignalQuality | | | | | | |
| **8** | Checksum High | | | | | | | |
| **9** | Checksum Low | | | | | | | |

Tabelle 2: USB Frane Header

Der Inhalt des Frames ist durch den ResponseType definiert:

|  |  |
| --- | --- |
| Mask | Response Type |
| 0x10 | RESPONSE\_DATA\_WRITTEN |
| 0x20 | RESPONSE\_GET\_CONFIG |
| 0x30 | RESPONSE\_GET\_CURRENT |
| 0x40 | RESPONSE\_GET\_HISTORY |
| 0x50 | RESPONE\_REQUEST |
| 0x50 | RESPONSE\_REQ\_READ\_HISTORY |
| 0x51 | RESPONSE\_REQ\_FIRST\_CONFIG |
| 0x52 | RESPONSE\_REQ\_SET\_CONFIG |
| 0x53 | RESPONSE\_REQ\_SET\_TIME |

Tabelle 3: Response Types

### USB Current Data

Das Current Data Datenpaket beinhaltet keinen Zeitstempel. Da keine Möglichkeit besteht, das BeagleBone mit der internen Uhr des Klimaloggers zu synchronisieren, wird dieser Datensatz nicht verwendet. Zur Vollständigkeit ist er trotzdem aufgeführt.

|  |  |  |
| --- | --- | --- |
| **Index** | **MSB** | **LSB** |
| **10** | Hum\_0\_MaxDT\_Year 10 | Hum\_0\_MaxDT\_Year 1 |
| **11** | Hum\_0\_MaxDT\_Month 1 | Hum\_0\_MaxDT\_Day 10 |
| **12** | Hum\_0\_MaxDT\_Day 1 | Hum\_0\_MaxDT\_Tim1 |
| **13** | Hum\_0\_MaxDT\_Tim2 | Hum\_0\_MaxDT\_Tim3 |
| **14** | Hum\_0\_MinDT\_Year 10 | Hum\_0\_MinDT\_Year 1 |
| **15** | Hum\_0\_MinDT\_Month 1 | Hum\_0\_MinDT\_Day 10 |
| **16** | Hum\_0\_MinDT\_Day 1 | Hum\_0\_MinDT\_Tim1 |
| **17** | Hum\_0\_MinDT\_Tim2 | Hum\_0\_MinDT\_Tim3 |
| **18** | Hum\_0 \_Max10 | Hum\_0\_Max 1 |
| **19** | Hum\_0\_Min 10 | Hum\_0\_Min 1 |
| **20** | Hum\_0 10 | Hum\_0 1 |
| **21** | \'0' | Temp\_0\_MaxDT\_Year 10 |
| **22** | Temp\_0\_MaxDT\_Year 1 | Temp\_0\_MaxDT\_Month 1 |
| **23** | Temp\_0\_MaxDT\_Day 10 | Temp\_0\_MaxDT\_Day 1 |
| **24** | Temp\_0\_MaxDT\_Tim1 | Temp\_0\_MaxDT\_Tim2 |
| **25** | Temp\_0\_MaxDT\_Tim3 | Temp\_0\_MinDT\_Year 10 |
| **26** | Temp\_0\_MinDT\_Year 1 | Temp\_0\_MinDT\_Month 1 |
| **27** | Temp\_0\_MinDT\_Day 10 | Temp\_0\_MinDT\_Day 1 |
| **28** | Temp\_0\_MinDT\_Tim1 | Temp\_0\_MinDT\_Tim2 |
| **29** | Temp\_0\_MinDT\_Tim3 | Temp\_0\_Max 10 |
| **30** | Temp\_0\_Max 1 | Temp\_0\_Max 0.1 |
| **31** | Temp\_0\_Min 10 | Temp\_0\_Min 1 |
| **32** | Temp\_0\_Min 0.1 | Temp\_0 10 |
| **33** | Temp\_0 1 | Temp\_0 0.1 |

Tabelle 4: CurrentData Beschreibung

Dieser Block wiederholt sich für jeden Sensor.

### History Data

In der History Data sind 2 verschiedene Datentypen versteckt. Zum einen den History Record, zum anderen AlarmData. Die Unterscheidung wird im letzten Block gemacht. Ist im letzten Block 0xEE gespeichert handelt es sich um AlarmData. Für dieses Projekt sind wir nur an den HistoryRecord intressiert. In der Table sind jedoch beide DatenTypen als Beispiel aufgezeichnet. Der HistoryRecord ist von Index 16 – 43 und der AlarmData Block ist von 44 -71.Sofern keine Alarme definiert wurden werden 6 HistoryRecords übertragen.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **MSB** | | **LSB** |  |
| **10** | LatestAddr | | |
| **11** |
| **12** |
| **13** | thisAddr | | |
| **14** |
| **15** |
| **16** | Pos\_6\_Hum\_8 10 | Pos\_6\_Hum\_8 1 | |
| **17** | Pos\_6\_Hum\_7 10 | Pos\_6\_Hum\_7 1 | |
| **18** | Pos\_6\_Hum\_6 10 | Pos\_6\_Hum\_6 1 | |
| **19** | Pos\_6\_Hum\_5 10 | Pos\_6\_Hum\_5 1 | |
| **20** | Pos\_6\_Hum\_4 10 | Pos\_6\_Hum\_4 1 | |
| **21** | Pos\_6\_Hum\_3 10 | Pos\_6\_Hum\_3 1 | |
| **22** | Pos\_6\_Hum\_2 10 | Pos\_6\_Hum\_2 1 | |
| **23** | Pos\_6\_Hum\_1 10 | Pos\_6\_Hum\_1 1 | |
| **24** | Pos\_6\_Hum\_0 10 | Pos\_6\_Hum\_0 1 | |
| **25** | \'0' | Pos\_6\_Temp\_8 10 | |
| **26** | Pos\_6\_Temp\_8 1 | Pos\_6\_Temp\_8 0.1 | |
| **27** | Pos\_6\_Temp\_7 10 | Pos\_6\_Temp\_7 1 | |
| **28** | Pos\_6\_Temp\_7 0.1 | Pos\_6\_Temp\_6 10 | |
| **29** | Pos\_6\_Temp\_6 1 | Pos\_6\_Temp\_6 0.1 | |
| **30** | Pos\_6\_Temp\_5 10 | Pos\_6\_Temp\_5 1 | |
| **31** | Pos\_6\_Temp\_5 0.1 | Pos\_6\_Temp\_4 10 | |
| **32** | Pos\_6\_Temp\_4 1 | Pos\_6\_Temp\_4 0.1 | |
| **33** | Pos\_6\_Temp\_3 10 | Pos\_6\_Temp\_3 1 | |
| **34** | Pos\_6\_Temp\_3 0.1 | Pos\_6\_Temp\_2 10 | |
| **35** | Pos\_6\_Temp\_2 1 | Pos\_6\_Temp\_2 0.1 | |
| **36** | Pos\_6\_Temp\_1 10 | Pos\_6\_Temp\_1 1 | |
| **37** | Pos\_6\_Temp\_1 0.1 | Pos\_6\_Temp\_0 10 | |
| **38** | Pos\_6\_Temp\_0 1 | Pos\_6\_Temp\_0 0.1 | |
| **39** | PosDT\_6\_Year 10 | PosDT\_6\_Year 1 | |
| **40** | PosDT\_6\_Month 10 | PosDT\_6\_Month 1 | |
| **41** | PosDT\_6\_Day 10 | PosDT\_6\_Day 1 | |
| **42** | PosDT\_6\_Hours 10 | PosDT\_6\_Hours 1 | |
| **43** | PosDT\_6\_Minutes 10 | PosDT\_6\_Minutes 1 | |

Tabelle 5: History Record Beschreibung

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **44** | \'0' | | | | |
| **45** |
| **46** |
| **47** |
| **48** |
| **49** |
| **50** |
| **51** |
| **52** |
| **53** |
| **54** |
| **55** |
| **56** |
| **57** | Pos\_5\_Hum\_Hi 10 | | | | Pos\_5\_Hum\_Hi 1 |
| **58** | Pos\_5\_Hum\_Lo 10 | | | | Pos\_5\_Hum\_Lo 1 |
| **59** | Pos\_5\_Hum 10 | | | | Pos\_5\_Hum 1 |
| **60** | Pos\_5\_Temp\_Hi 10 | | | | Pos\_5\_Temp\_Hi 1 |
| **61** | Pos\_5\_Temp\_Hi 0.1 | | | | Pos\_5\_Temp\_Lo 10 |
| **62** | Pos\_5\_Temp\_Lo 1 | | | | Pos\_5\_Temp\_Lo 0.1 |
| **63** |  | \'0' |  |  | Pos\_5\_Temp 10 |
| **64** | Pos\_5\_Temp 1 | | | | Pos\_5\_Temp 0.1 |
| **65** | Pos\_5\_AlarmData | | | | Pos\_5\_Sensor |
| **66** | PosDT\_5\_Year 10 | | | | PosDT\_5\_Year 1 |
| **67** | PosDT\_5\_Month 10 | | | | PosDT\_5\_Month 1 |
| **68** | PosDT\_5\_Day 10 | | | | PosDT\_5\_Day 1 |
| **69** | PosDT\_5\_Hours 10 | | | | PosDT\_5\_Hours 1 |
| **70** | PosDT\_5\_Minutes 10 | | | | PosDT\_5\_Minutes 1 |
| **71** | 0xEE | | | | |

Tabelle 6: Alarm Data Beschreibung

# MASCOT Grobdesign

Für die Grobe Erfassung der Problematik wurde ein MASCOT Design entworfen. Das MASCOT Design stellt die nebenläufigen Abhängigkeiten in einer einfachen und Übersichtlichen Weise dar.

## Diagramm



Abbildung 3: MASCOT Diagramm

## Aktivitätsbeschreibung

### Anzeige-Prozess A\_01

Der Anzeige Prozess wird vom QtCreator zur Verfügung gestellt. Er behandelt die Interaktion mit dem Touch und der Anzeige.

* Singal C\_03 [readErrno]
  + Wenn Errno == 200 und MsgBox nicht sichtbar:
    - Zeige MsgBox (Information an Users -> USB Knopf muss gedrückt werden)
  + Wenn Errno == 0 und MsgBox sichtbar:
    - Schliesse MsgBox
* Timer A\_05 TR Event:
  + Gewünschte Daten aus P\_01 auslesen (Gewünschte Daten werden durch das wählen eines Zeitintervalls festgelegt)
  + Neu zeichen des Grafikplots

### Arbeiter-Prozess A\_02

* Lese Daten aus C\_04a
  + Schreibe C\_03[readErrno]
  + Wenn Daten vorhanden:
    - Ist Datensatz HistoryData, schreibe Daten in P\_01(Datenbank)
* Wiederhole Schritt 1 bis zum Applikationsende

### Treiber A\_03

* Lese C\_04b
  + Wenn leer, fahre fort
  + Parse Config Data (last index, alarm data)
* Schreibe C\_05b (Request)
* Lese C\_05a
  + Wenn Daten == HistoryData oder CurrentData, schreibe Daten in C\_04a
  + Wenn Daten == error, schreibe errno in C\_04a

### Uhr A\_04

Der Prozess „Uhr“ ist vom System bereitgestellt und dient nur zur Vervollständigung des Designs

### Timer A\_05

Der Prozess „Timer“ wird von QT Framework bereitgestellt und dient nur zur Vervollständigung des Designs

## Channel- Beschreibung

### User Interaction Channels C\_01/C\_02

Die Channels C\_01 und C\_02 werden vom QT Creator bereitgestellt und können nicht genauer erläutert werden.

### Error Auswertung C\_03

* Signal [readErrno(int)], um Interaktionen mit dem User aus dem Kernel steuern zu können

### Treiber Daten lesen C\_04a

* USB Frame [byte[ ] ] Daten eines USB Frames, genaue Beschreibung siehe Reverse Engineering - Datenbeschreibung

### Treiber Daten schreiben C\_04b

* Index [int], teilt dem Treiber mit, von welchem Index die Daten gelesen werden soll

### USB Tranceiver Daten lesen C\_05a

* USB Frame, Data Message

### USB Tranceiver Daten schreiben C\_05b

* USB Frame, Control Message

### Timer Response C\_06a

* Eregnis (nach ablauf der gesetzen Zeit)

### Timer Setzen C\_06b

* Zeit in ms nach welcher das „Timer Response“ Ereignis eintritt

## Pool Beschreibung

### Datenbank P\_01

Datenbank welche die Messwerte und zugehörigen Zeitstempel in einer einfachen Form abspeichert.

### Zeitstempel P\_02

Ablage des aktuellen Zeitstempels als UNIX Timestamp (Anzahl verstrichene Sekunden seit dem 01.01.1970).

# Abgabe Strukur

In der folgenden Auflistung finden Sie eine Übersicht der abgegebenen Dokumente:

KlimaLogg/  
3rd\_party/  
    include/  
      GLES2/  
        gl2.h  
        gl2ext.h  
        gl2platform.h  
      KHR/  
        khrplatform.h  
    qcustomplot/  
      QCustomPlot.tar.gz  
    sqlite3/  
      sqlite-amalgamation-3081101.zip  
      Makefile  
      make\_env\_target  
      make\_env\_host  
database/  
    KlimaLoggPro.sdb    
doc/  
    Entwicklung und Integration eines USB.docx  
    Entwicklung und Integration eines USB.pdf  
    kl-106.py  
praesentation/  
    Integrationsprojekt\_danir1\_suhnu1\_chrib1.odp  
    Integrationsprojekt\_danir1\_suhnu1\_chrib1.pdf   
Qt/  
    KlimaLoggProOnBBB/  
      KlimaLoggProOnBBB.pro  
      KlimaLoggProOnBBB.pro.user  
      bitconverter.h  
      definitions.h  
      kldatabase.h  
      mainwindow.h  
      qcustomplot.h  
      readdataworker.h  
      bitconverter.cpp  
      kldatabase.cpp  
      main.cpp  
      mainwindow.cpp  
      qcustomplot.cpp  
      readdataworker.cpp  
      mainwindow.ui   
build-KlimaLoggProOnBBB-BBB\_BFH\_Cape-Debug/  
      KlimaLoggProOnBBB  
startup/  
    KlimaLoggProOnBBB.sh  
usbdriver/  
    src/  
      kl\_usb\_drv.c  
      kl\_usb\_drv.h    
      Makefile  
    bin/  
      kl\_usb\_drv.ko

# InstallationsAnleitung

Basis ist das Embedded Linux Debian Jessie aus dem Tools & Chains Unterricht. Gemäss dem Lab 1-1 Embedded Linux Configuration - Ubuntu 14.04 LTS sind diese Schritte aus Punkt 2.16 Linux Root File System und Punkt 2.17 Linux Kernel auszuführen:  
  ***cd /opt/embedded/bbb***  
  ***sudo tar -xjvf ~/Downloads/armhf-rootfs-debian-jessie-bfh.tar.bz2  
  cd /opt/embedded/bbb/kernel  
  tar -xjvf ~/Downloads/old/linux-dev-am33x-v3.18.tar.bz2  
  cd linux-dev-am33x-v3.18/  
  make mrproper  
  make bbb\_defconfig***  
  
Jetzt muss zusätzlich im Kernel das Modul HID-Generic auf Modul gesetzt werden, damit wir danach stattdessen unser USB-HID-Modul für den KlimaLoggPro laden können.  
***make menuconfig***  
  
Auswählen von Device Drivers > HID Support > Generic HID driver   
selektieren und mittels M als Modul auswählen.  
  
Mit <Save> und <OK> die Konfiguration abschliessen und speichern.  
Mit drei Mal <Exit> die Konfiguration verlassen.  
  
Weiterfahren gemäss Punkt 2.17:  
  ***make -j5  
  make modules -j5  
  sudo -s  
  export ARCH=arm  
  export CROSS\_COMPILE=arm-linux-  
  export PROJECT=/opt/embedded/bbb  
  export INSTALL\_MOD\_PATH=/opt/embedded/bbb/rootfs  
  make modules\_install  
  exit***    
Gemäss Punkt 2.21 Touchsceen calibration  
***root@BBB-BFH-Cape:~# cd /usr/local/bin  
  root@BBB-BFH-Cape:/usr/local/bin# ./ts\_calibrate***      
Weiter gemäss Punkt 2.22 Install steps for the SGX drivers  
***root@BBB-BFH-Cape:/usr/local/bin# cd /opt/gfxinstall  
  root@BBB-BFH-Cape:/opt/gfxinstall# ./sgx-install.sh  
  root@BBB-BFH-Cape:/opt/gfxinstall# reboot***  
    
Jetzt kann der neue Driver kl\_usb\_drv.ko auf das BBB gebracht werden  
***cp <Quellpfad>/usbdriver/kl\_usb\_drv.ko /opt/embedded/bbb/rootfs/lib/modules/3.18.5+  
  root@BBB-BFH-Cape:/lib/modules/3.18.5+# depmod -a  
  root@BBB-BFH-Cape:/lib/modules/3.18.5+# modprobe kl\_usb\_drv***    
Kompilieren von SQLite fürs BBB:  
Download der Sourcen sqlite-amalgamation-3081101.zip von <https://www.sqlite.org/download.html> und entpacken des Files. im <Quellpfad>/sqlite3/sqlite-amalgamation-3081101mit angepassten Files Makefile, make\_env\_host, make\_env\_target aus dem Kurs Tools & Chains durchführen von   
***make  
  make install***  
    
Kopieren der KlimaLoggPro SQLite Datenbank auf den BBB:  
***mkdir /usr/local/bin/database/***  
  ***cp <Quellpfad>/database/KlimaLoggPro.sdb /usr/local/bin/database/KlimaLoggPro.sdb***    
Ergänzen der fehlenden GLES2 Header Dateien für den Compile in Qt:  
  ***root@host:~# cp -r <Quellpfad>/3rd\_party/include/\* /opt/embedded/bbb/rootfs/usr/include/***  
Die kompilierte Qt Applikation auf den BBB kopieren  
  ***root@granit:~# cp <Quellpfad>/build-KlimaLoggProOnBBB-BBB\_BFH\_Cape-Debug/KlimaLoggProOnBBB /opt/embedded/bbb/rootfs/usr/local/bin/KlimaLoggProOnBBB***    
Die Zeitzone auf dem BBB noch auf Mittel Europa setzen  
***root@BBB-BFH-Cape:~# cp  /usr/share/zoneinfo/CET /etc/localtime***Die Kontrolle mittels  
***root@BBB-BFH-Cape:~# date***sollte etwas im Format CEST bzw CET zurückgeben wie z.B.  
  Tue Apr 21 08:27:08 CEST 2015  
    
Analog zum Lab 5.3 - Run your program at start-up aus dem Tools & Chains Unterricht.  
kann jetzt die Applikation in den Startup eingebunden werden  
***root@BBB-BFH-Cape:~# systemctl disable getty@tty1***das Startup Script auf den BBB kopieren  
***cp <Quellpfad>/startup/KlimaLoggProOnBBB.sh /opt/embedded/bbb/rootfs/etc/init.d  
  root@BBB-BFH-Cape:~# update-rc.d KlimaLoggProOnBBB.sh defaults***    
Nun ist alles bereit für einen   
***root@BBB-BFH-Cape:~# reboot***

# Schlussfolgerungen/Fazit

Die Bedenken zum Start waren gross. Keiner von uns konnte Python programmieren und unserer einziger Anhaltspunkt für den Linux Treiber war der vorhandene in Python geschriebener Treiber! Also mussten wir uns zuerst einmal in den Python Code einarbeiten, die Abläufe und die Datenpakete analysieren. Die Erleichterung war gross als wir dann eine USB Message mit den von uns geschriebenen C Code senden konnten und etwas Sinnvolles von dem USB Tranceiver zurückkam. Dann ging es mit grossen Schritten voran und das im Vorfeld entwickelte MASCOT Design konnte in Code umgesetzt werden.

Die Projektarbeit half uns das im Unterricht erlernte Wissen zu festigen. Vor allem im Treiber Bereich konnten wir vieles Anwenden und lernen.

# Abbildungsverzeichnis

[Abbildung 1: Sequenzdiagramm Treiber Initialisierung 5](file:///C:\Users\Christian\Documents\GitHub\KlimaLogg\doc\Entwicklung%20und%20Integration%20eines%20USB.docx#_Toc431238753)

[Abbildung 2: Sequenzdiagramm Treiber Kommunikation 6](file:///C:\Users\Christian\Documents\GitHub\KlimaLogg\doc\Entwicklung%20und%20Integration%20eines%20USB.docx#_Toc431238754)

[Abbildung 3: MASCOT Diagramm 11](#_Toc431238755)

# Tabellenverzeichnis

[Tabelle 1: Rahmenbedingungen 4](#_Toc431238756)

[Tabelle 2: USB Frane Header 7](#_Toc431238757)

[Tabelle 3: Response Types 7](#_Toc431238758)

[Tabelle 4: CurrentData Beschreibung 8](#_Toc431238759)

[Tabelle 5: History Record Beschreibung 9](#_Toc431238760)

[Tabelle 6: Alarm Data Beschreibung 10](#_Toc431238761)