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

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.

2 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

3 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:

3.1 Sequenzdiagramm Python Treiber Initialisierung

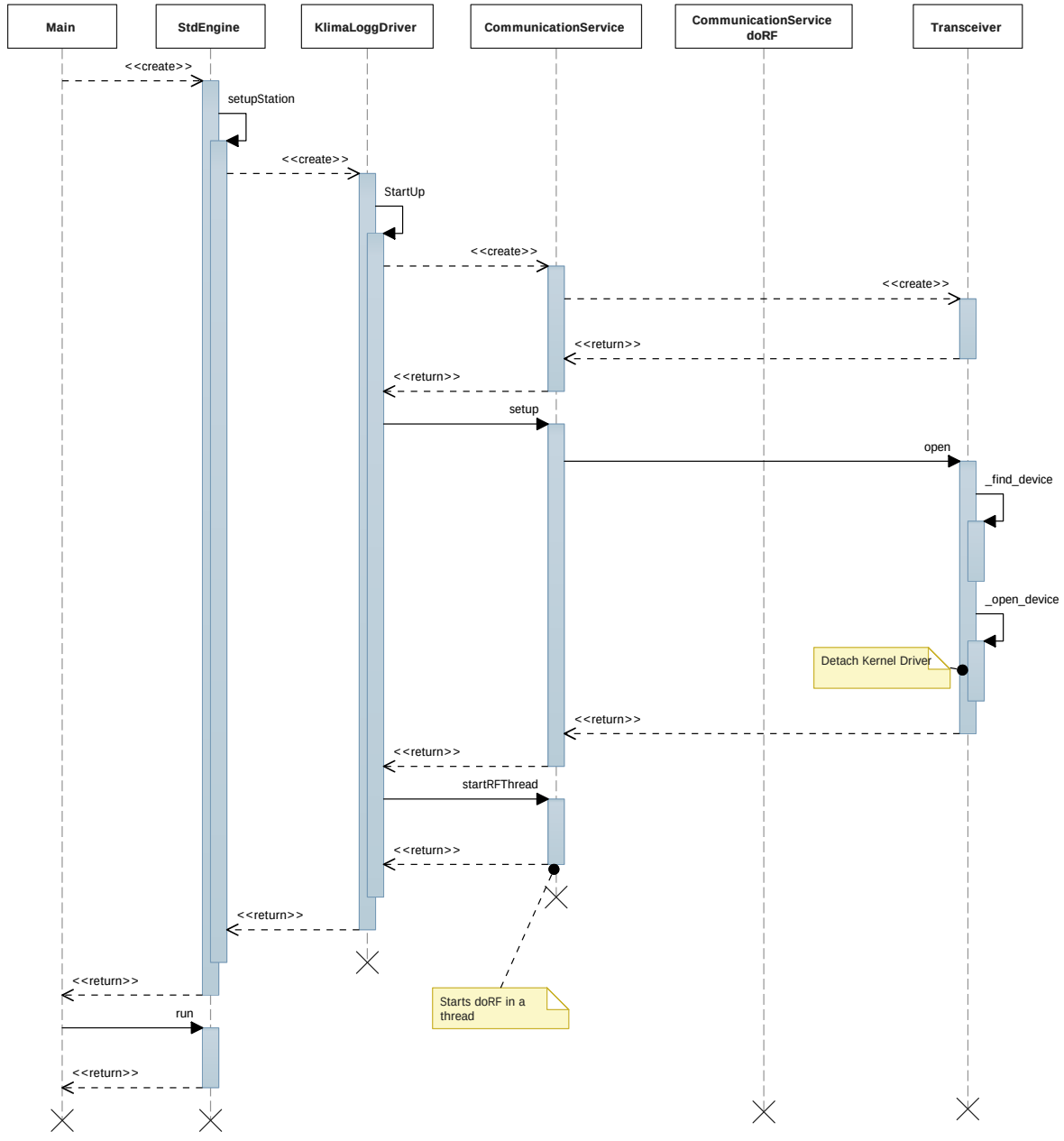

Abbildung 1: Sequenzdiagramm Treiber Initialisierung

3.2 Sequenzdiagramm Python Treiber Kommunikation

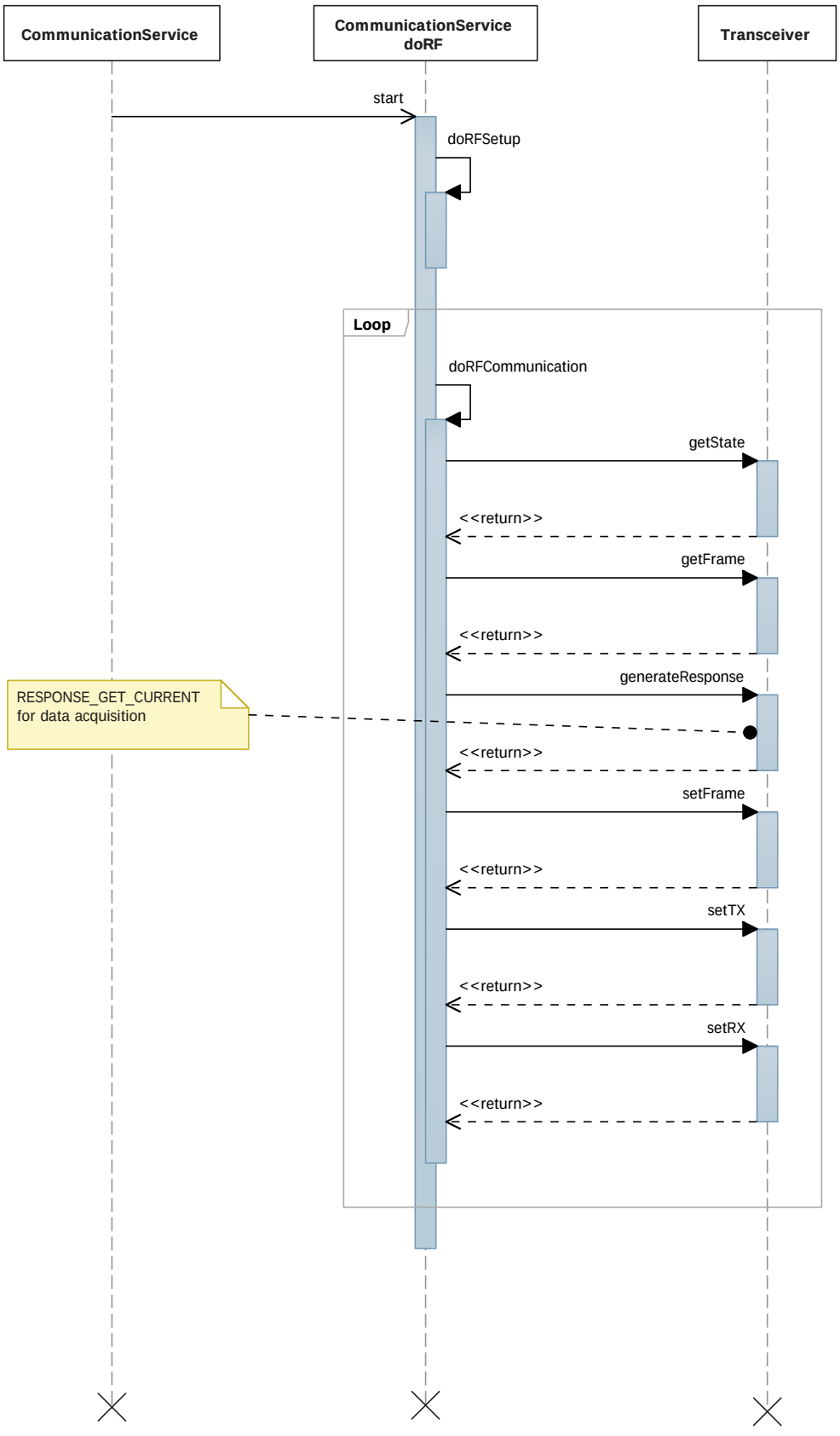

Abbildung 2: Sequenzdiagramm Treiber Kommunikation

3.3 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.

3.3.1 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

3.3.2 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.

3.3.3 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

4 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.

4.1 Diagramm

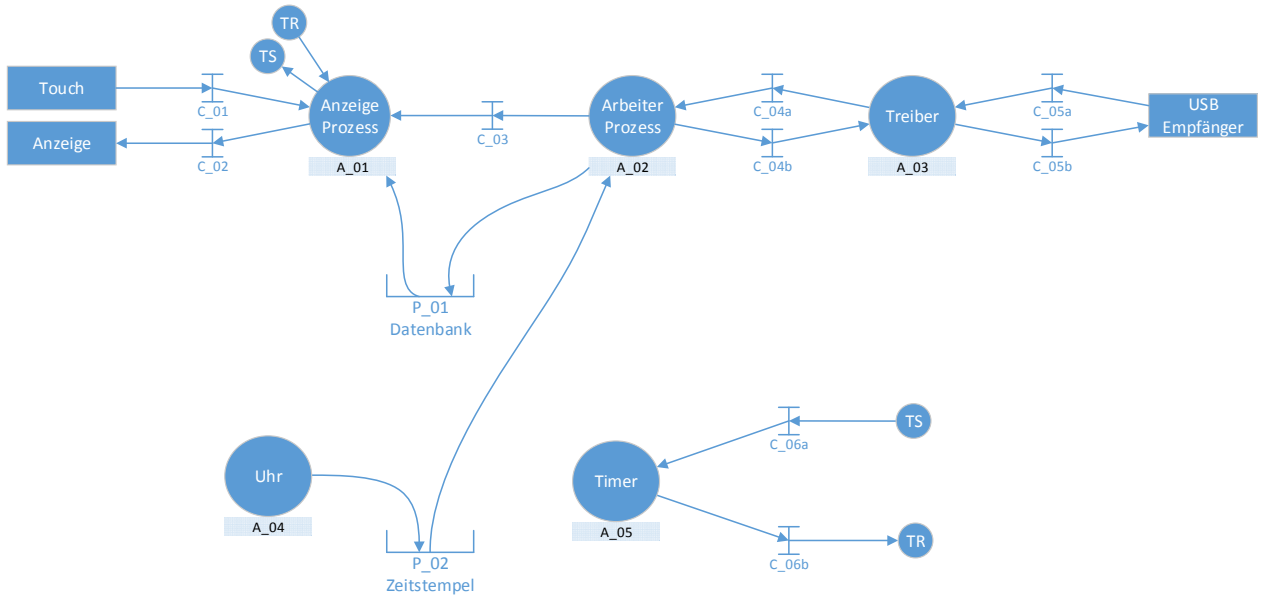

Abbildung 3: MASCOT Diagramm

4.2 Aktivitätsbeschreibung

4.2.1 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

4.2.2 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

4.2.3 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

4.2.4 Uhr A_04
Der Prozess „Uhr“ ist vom System bereitgestellt und dient nur zur Vervollständigung des Designs

4.2.5 Timer A_05
Der Prozess „Timer“ wird von QT Framework bereitgestellt und dient nur zur Vervollständigung des Designs

4.3 Channel - Beschreibung

4.3.1 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.

4.3.2 Error Auswertung C_03

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

4.3.3 Treiber Daten lesen C_04a
- USB Frame [byte[ ]] Daten eines USB Frames, genaue Beschreibung siehe Reverse Engineering - Datenbeschreibung

4.3.4 Treiber Daten schreiben C_04b
- Index [int], teilt dem Treiber mit, von welchem Index die Daten gelesen werden soll

4.3.5 USB Tranceiver Daten lesen C_05a
- USB Frame, Data Message

4.3.6 USB Tranceiver Daten schreiben C_05b
- USB Frame, Control Message

4.3.7 Timer Response C_06a
- Eregnis (nach ablauf der gesetzen Zeit)

4.3.8 Timer Setzen C_06b
- Zeit in ms nach welcher das „Timer Response“ Ereignis eintritt

4.4 Pool Beschreibung

4.4.1 Datenbank P_01
Datenbank welche die Messwerte und zugehörigen Zeitstempel in einer einfachen Form abspeichert.

4.4.2 Zeitstempel P_02
Ablage des aktuellen Zeitstempels als UNIX Timestamp (Anzahl verstrichene Sekunden seit dem 01.01.1970).

5 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

6 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.

7 Abbildungsverzeichnis

8 Tabellenverzeichnis