Hochschule Esslingen
University of Applied Sciences

Studienarbeit

Quadrocopter - Hardware Migration auf das Raspberry Pi Board

im Studiengang Technische Informatik
der Fakultät Informationstechnik
Wintersemester 2015/2016

Chris Mönch

Prüfer: Prof. Dr. Jörg Friedrich
Zweitprüfer: M.Sc. Vikas Agrawal

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 2. November 2015 _____

Unterschrift

Kurz-Zusammenfassung

In dieser Dokumentation sind sämtliche Schritte festgehalten, welche benötigt werden um die Software und Peripherie Geräte vom Quadrocopter auf das Neue Rasperry Pi Board zu migrieren. Da dieses die alte Hardware HCS12-Dragon Board ersetzen soll. Ziel ist es die Software der einzelnen Komponenten anzupassen, sodass der Quadrocopter mit dem Raspberry Pi flugfähig ist.

Zusätzlich sind hier Grundlegende Bausteine festgehalten, welche sich mit der Virtuellen Maschine und Eclipse beschäftigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Erläuterung der Aufgabenstellung und den Randbedingungen

2 Grundlagen

2.1 Virtuelle Machine

Für das Projekt "Quadrocopter"wird eine Virtuelle Maschine verwendet. Hierfür wird eine bereits fertig funktionierende Vorlage bereitgestellt.

Auf dieser VM Läuft das Betriebssystem Ubuntu v14.0.

Für die Verwendung der VM wird ein Programm wie z.b. VirtualBox von Oracle oder VMware Workstation Player(abgespeckte Version von VMware Workstation, kostenlos für private Nutzung) benötigt.

Änderungen an denn VMware Einstellungen müssen nicht vorgenommen werden.

2.1.1 Ubuntu

2.1.1.1 Autostart bei Bootvorgang

Achtung: Wenn es sich bei dem Script oder Programm um ein Endlos-Script/Programm handelt oder eine User Interaktion benötigt wird das Linux Betriebssystem nicht mehr Starten!

Es ist für einmalige Einstellungen Vorgesehen

File erzeugen in init.d: sudo nano /etc/init.d/myAutostartScript

```
sudo nano /etc/init.d/myAutostartScript
```

Inhalt anpassen:

```
#! /bin/sh
### BEGIN INIT INFO

# Short-Description: Starts myProgramm
# Description:
### END INIT INFO

case "$1" in
start)
```

```
10 echo "noip␣wird␣gestartet"
11 # Starte Programm
12 /usr/local/bin/myProgramm
13 ;;
14 stop)
15 echo "myProgramm␣wird␣beendet"
16 # Beende Programm
17 killall myProgramm
18 ;;
19 *)
20 echo "Benutzt:␣/etc/init.d/myAutostartScript␣{start|stop}"
21 exit 1
22 ;;
23 esac
24
25 exit 0
```

Später wird beim Start des PC oder PI´s das Script automatisch mit dem Parameter 'start' ausgeführt. Beim Herunterfahren wird natürlich der Parameter 'Stop' übergeben. Für jeden anderen Fall wird hier "Benutzt: /etc/init.d/myAutostartScript start|stopäusgegeben.

Anschließend muss das Programm noch ausführbar gemacht werden.

```
1 sudo chmod 755 /etc/init.d/myAutostartScript
```

Um es zu Testen kann das Script mit denn folgenden Befehlen gestartet bzw. gestoppt werden.

```
1 sudo /etc/init.d/myAutostartScript start
2 sudo /etc/init.d/myAutostartScript stop
```

Wenn dieser Manuelle Test erfolgreich war, muss es nun noch eingestellt werden, das es beim booten aufgerufen werden.

```
1 sudo update-rc.d NameDesSkripts defaults
```

Mit folgendem Befehl kann man das Script vom Autostart wieder entfernen entfernen:

```
1 sudo update-rc.d -f  NameDesSkripts remove
```

2.1.1.2 Autostart bei LogIn für User

Für das Starten von Programmen für bestimmte User können die .desktop Dateien verwendet werden. Die Müssen in dem jeweiligen Home Verzeichnisses des Users unter dem folgenden Pfad abgelegt werden: .config/autostart/.

Der Name darf bis auf die Endung .Desktop frei gewählt werden. z.B. *myAutostart.desktop*.

Ein Neues File kann mit

```
sudo nano ~/.config/autostart/myAutostart.desktop
```

erzeugt werden. Der Aufbau einer .desktop ist folgend dargestellt:

```
[Desktop Entry]
Type=Application
Terminal=true
Exec=/home/user/workspace/RaspberryDemoUdpSendHost/Debug/RaspberryDemoUdpSer
Hidden=false
NoDisplay=false
X-GNOME-Autostart-enabled=true
Name[en_US]=RaspberryDemoUdpSendHost
Name=RaspberryDemoUdpSendHost
Comment[en_US]=
Comment=
```

Benötigte Parameter:

Zum erstellen eines Programm starters ist der Typ Apllication zu wählen. Ecex beinhaltet die ausführbare Datei welche im Terminal gestartet werden soll. Der Name gibt denn Name der gestarteten Anwendung an.

Optionale Parameter: Um ein Terminal sichtbar zu machen muss die Einstellung Temrinal auf true gesetzt werden.

Über denn Parameter X-GNOME-Autostart-enabled kann man das Autostarten des Programmes aussetzen wenn man dieses auf false stellt. Standardmäßig ist er auf True gesetzt.

Falls das Programm bei Beendigung erneut gestartet werden soll ist dies über X-GNOME-AutoRestartmöglich.

2.1.2 VMWare Workstation Player

Der VMware Player kann direkt hier runter geladen werden.

Um die Kopie der VM zu verwenden müssen sie eine bereits existierende VM öffnen. Wählen sie die .vmx aus uns Starten sie die VM. Bei Frage ob es sich hierbei um eine 'Moved' oder 'Copied' VM handelt, wählen sie 'I Copied It' aus.

Die VM kann verwendet werden.

2.1.3 VirtualBox

Für die VirtualBox, welche z.B. hier Verfügbar ist, kann durch Änderungen in den Einstellung ebenfalls .vmdk Dateien Verwenden. Das File sollte aber keines falls konvertiert werden und im .vmdk Format bestehen bleiben.

Zunächst muss eine Neue Virtuelle Maschine des Typs Linux mit Betriebssystem Ubuntu 64Bit Version erstellt werden.Weißen sie der VM 2GB Hauptspeicher zu. Als Festplatte wählen sie aus das sie Eine Bereits Vorhandene Festplatte verwenden möchten. Klicken sie daraufhin auf denn Ordner mir Grünem Pfeil und wählen sie die .vmdk der kopierten VM aus und erzeugen sie die VM.

Wählen sie nun die gerade erstellt VM aus und klicken sie auf Ändern. Im Menü 'System' muss das *IO-APIC aktivieren* aktiviert sein. Unter dem Menüpunkt 'Massenspeicher' entfernen sie, falls vorhanden, die .vmdk unter SATA-Controller und fügen sie diese unter Controller:IDE neu hinzu. Indem sie auf 'Festplatte hinzufügen' klicken und bestätigen sie die Einstellungen.

Nun kann die .vmdk auch unter VirtualBox verwendet werden.

2.2 Hardware

2.2.1 Raspberry Pi

2.2.2 Peripherie

2.2.2.1 Motoren

2.2.2.2 Ggyroscope

2.2.2.3 Abstandlaser

2.3 IDE

In der VM ist die verwendete IDE 'Eclipse' mit zwei verschiedenen GCC bereits installiert. Einer ist der sogenannte Native Compiler, dieser Übersetzt in denn Maschinen Code welcher für die aktuelle Plattform benötigt wird. Desweiteren gibt es einen Cross Compiler welcher verwendet wird um Maschinencode für eine andere Plattform zu generieren.

2.3.1 Schritte vom Quellcode zur ausführbaren Datei

Zuerst werden sämtliche definierte Konstenten, Makros und Präprozessor Anweisungen durch die angegebenen Werte ersetzt.

Der Compiler hat die Hauptaufgabe, aus denn Quellcode (.c, .cpp und .h) in nativen Maschinencode zu übersetzen, welcher nur von einem bestimmten Prozessor gelesen werden kann. Ausnahmen hierbei sind z.B. Compiler in der .Net (C#) Umgebung. Bei diesen werden die Source Dateien in eine sogenannte 'Intermedia Languages' übersetzt, welche unabhängig vom Zielsystem sind. Bei der Ausführung des Codes wird der 'Intermedia Languages' durch den Compiler, bei jedem Start erneut, in denn Prozessor spezifischen Maschinencode übersetzt.

Weitere Aufgaben des Compilers bestehen darin Syntax Fehler zu Überprüfung, Code Optimierungen, Entfernen von nicht erreichbaren Codes, Einfügen von Funktionen.

Als Ausgabe liefert der Compiler Code denn sogenannten Objekt Code.

Die Einzelnen Objekt Files oder Module müssen noch miteinander Verbunden werden. Dies übernimmt der Linker, dieser (Wenn Erfolgreich) generiert die ausführbare Datei. Vom Linker werden unter anderem Funktionen aus der Standard Libary hinzugefügt und alle Objektdateien welche noch nicht aufgelöste Symbole enthalten überprüft und gelöst. Man Unterscheidet hier zwischen statischem und dynamischen linken. Das statische Linken wird einmalig durchgeführt, dynamisches linken hingegen bei jedes mal zur Laufzeit.

Zuletzt gibt es noch denn Loader, dieser lädt Teile des ausführbaren Code in denn Hauptspeicher, welche in nächster Zeit benötigt werden könnten.

2.3.2 Makefile

Makefiles sind eine Möglichkeit (mittels des Programmes 'make') um aus mehreren Quell Files und Bibliotheken über Objekt Files einen ausführbaren Code zu generieren. Das besondere an dieser Methode ist, das beim ausführen des Makefiles bloß geänderte Quell Dateien neu kompiliert werden und mit denn darauffolgenden binden an die genveränderten Objekt Dateien. Kurz: Es lassen sich Abhängigkeiten definieren. Dies ist Zeitsparend da nicht bei Änderung eines Files das komplette Projekt compiliert werden muss.

http://www.oreilly.de/german/freebooks/rlinux3ger/ch133.html

http://docs.sfz-bw.de/phag/skripte/makefiles.pdf

http://www.mikrocontroller.net/attachment/82261/Make.pdf

2.3.3 Host Programme

Um ein Project für eine Host Anwendung zu erstellen, muss unter Eclipse ein neues *C/C++ Projekt* mit Dem Projekt Typ *Executable Empty Project* und denn Toolchains *Linux GCC* angelegt werden.

2.3.4 Raspberry Pi Programme

Um ein Project für eine Host Anwendung zu erstellen, muss unter Eclipse ein neues *C/C++ Projekt* mit Dem Projekt Typ *Executable Empty Project* und denn Toolchains *Cross GCC* angelegt werden. //TODO

2.3.5 Beispiel Code

Anbei befindet sich zwei Code Segmente, welche eine Nachricht über UDP/IP vom Raspberry Pi an denn Host sendet, mit diesen können die zuvor beschriebenen Schritte zum Test durchgeführt werden.

RaspberryDemoUdpSendHost.cpp:

```
1  /*
2  * RaspberryDemoUdpSendHost.cpp
3  *
4  *  Created on: Oct 23, 2015
5  *      Author: Chris Mönch
6  */
7
8  #include <iostream>
9  #include <stdio.h>
10 #include <sys/socket.h>
11 #include <netinet/in.h>
12 #include <string.h>
13 #include <unistd.h>
14
15 using namespace std;
16
17 int main(int argc, char * argv[]) {
18
19 cout << "Start\n";
20
21 int clientSocket, nBytesMessage, nBytesMessage2;
22 char message[12] = "Hello_World";
23 char message2[16] = "AnotherMessage";
24
```

```
25 nBytesMessage= sizeof(message)/ sizeof(message[0]);
26 nBytesMessage2 =sizeof(message2)/ sizeof(message2[0]);
27
28 struct sockaddr_in serverAddress;
29 socklen_t addressSize;
30
31 /*Create UDP socket*/
32 clientSocket = socket(PF_INET, SOCK_DGRAM, 0);
33
34 /*Configure settings in address struct*/
35 serverAddress.sin_family = AF_INET;
36 serverAddress.sin_port = htons(9999);
37 serverAddress.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
38 memset(serverAddress.sin_zero, '\0', sizeof
      serverAddress.sin_zero);
39
40 /*Initialize size variable to be used later on*/
41 addressSize = sizeof(serverAddress);
42
43 printf("Start␣Sending␣Messages\n");
44
45 while(1){
46 sleep(1);
47 sendto(clientSocket,message,nBytesMessage,0,(struct sockaddr
      *)&serverAddress,addressSize);
48 sleep(1);
49 sendto(clientSocket,message2,nBytesMessage2,0,(struct sockaddr
      *)&serverAddress,addressSize);
50 printf("And␣send␣again....\n");
51 }
52
53 return 0;
54 }
```

RaspberryDemoUdpReceiveHost.cpp:

```
1 /*
2 * RaspberryDemoUdpReceiveHost.cpp
3 *
4 *  Created on: Oct 23, 2015
5 *      Author: Chris Mönch
6 */
7
8 #include <sys/types.h>
9 #include <sys/socket.h>
10 #include <netinet/in.h>
11 #include <arpa/inet.h>
```

```
12 #include <netdb.h>
13 #include <stdio.h>
14 #include <unistd.h>
15 #include <string.h>
16 #include <stdlib.h>
17 #include <errno.h>
18 #include <time.h>
19
20 #define LOCAL_SERVER_PORT 9999
21 #define BUF 255
22
23 using namespace std;
24
25 int main(int argc, char * argv[]) {
26 int s, rc, n;
27 socklen_t len;
28 struct sockaddr_in cliAddr, servAddr;
29 char puffer[BUF];
30 const int y = 1;
31 s = socket (AF_INET, SOCK_DGRAM, 0);
32 if (s < 0) {
33 printf ("%s: Kann Socket nicht öffnen ...(%s)\n");
34 return 1;
35 }
36
37 /* Lokalen Server Port bind(en) */
38 servAddr.sin_family = AF_INET;
39 servAddr.sin_addr.s_addr = htonl (INADDR_ANY);
40 servAddr.sin_port = htons (LOCAL_SERVER_PORT);
41 setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &y, sizeof(int));
42 rc = bind ( s, (struct sockaddr *) &servAddr,
43 sizeof (servAddr));
44 if (rc < 0) {
45 printf ("%s: Kann Portnummern %d nicht binden (%s)\n");
46 return 1;
47 }
48 printf ("%s: Wartet auf Daten am Port (UDP) %u\n",
49 argv[0], LOCAL_SERVER_PORT);
50 /* Serverschleife */
51 while (1) {
52 /* Puffer initialisieren */
53 memset (puffer, 0, BUF);
54 /* Nachrichten empfangen */
55 len = sizeof (cliAddr);
56 n = recvfrom ( s, puffer, BUF, 0,(struct sockaddr *) &cliAddr,
    &len );
```

```
57 if (n < 0) {
58 printf ("%s: Kann keine Daten empfangen ...\n",
59 argv[0] );
60 continue;
61 }
62
63 /* Erhaltene Nachricht ausgeben */
64 printf ("%s \n", puffer);
65
66 }
67 return 0;
68
69 }
```

3 Realisierung

Beschreibung der HW- und SW-Realisierung

3.1 Unter-Kapitel in Realisierung

Beispiel Text

3.2 Weiteres Unter-Kapitel in Realisierung

3.3 Und noch ein Unter-Kapitel in Realisierung

3.4 Literaturverweise

Verweise im Text: [1] und [Gun04].

4 Ergebnisse

„Neuigkeiten“ Messergebnisse

5 Schluss

Ergebnis-Bewertung, Zusammenfassung und Ausblick

A Kapitel im Anhang

Alles was den Hauptteil unnötig vergrößert hätte, z. B. HW-/SW-Dokumentationen, Bedienungsanleitungen, Code-Listings, Diagramme

Literaturverzeichnis

[1] Thomas Nonnenmacher, LaTeX Grundlagen - Setzen einer wissenschaftlichen Arbeit Skript, 2008,http://www.stz-softwaretechnik.de; *(Bei STZ Internetseite unter Publikationen - Skripte) [V. 2.0 26.02.08]*

[Gun04] Karsten Günther, LaTeX2 — Das umfassende Handbuch, Galileo Computing, 2004, http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-768; *1. Auflage*