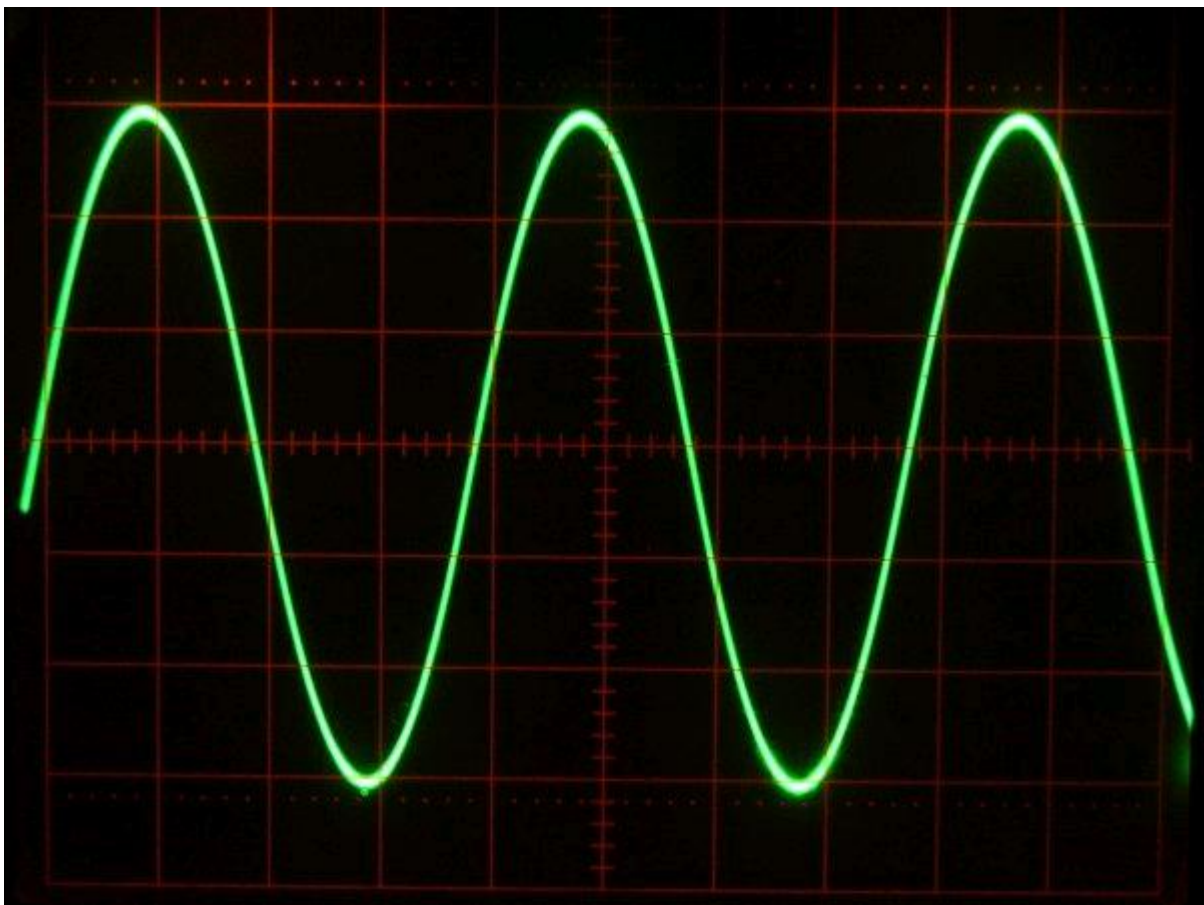


# Low Cost PCI Digital-Scope

## *Java User Interface*

---



**Projekt Homepage:** <http://code.google.com/p/lcpd-scope/>  
**Autoren:** T.Kurmann, R. Woodtli, S. Eichenberger  
**e-mail:** kurmt1@bfh.ch, woodr1@bfh.ch, eichs2@bfh.ch  
**Dozent:** I. Oesch

## Inhaltsverzeichnis

1	Einführung .....	1
2	Hardware .....	1
2.1	Raggedstone Spartan 3-E PCI Development Board .....	1
2.2	Intel Atom Motherboard .....	2
2.3	Aufbau .....	2
3	Software .....	3
3.1	Ubuntu .....	3
3.2	Treiber .....	3
3.3	Treiber Design .....	4
3.4	Beschreibung der wichtigsten Funktionen .....	4
3.4.1	Initialize .....	4
3.4.2	Probe .....	4
3.4.3	IOCTL .....	5
3.4.4	Read .....	6
3.5	C-Driver Inteface .....	6
3.6	Java User Interface .....	7
3.7	Java User Interface Design .....	9
3.7.1	LCPDControl .....	11
3.7.2	LCPDUserInterface .....	11
3.7.3	LCPDDriverInterface .....	11
3.7.4	LCPDChart .....	12
3.7.5	LVPDData .....	12
3.7.6	LCPDConverterSettings .....	12
4	Tests .....	12
5	Bedienungsanleitung .....	14
6	Schlussbetrachtung .....	14

## 1 Einführung

Als Teil einer fachübergreifenden Projektarbeit wurde das Low Cost PCI Digital-Scope (LCPD-Scope) ins Leben gerufen. Das Ziel dieser Arbeit ist es, ein Oszilloskop mit einer Bandbreite von 20MHz zu entwickeln. Die Daten sollen an jedem beliebigen Computer erfasst werden können. Deshalb entschieden wir uns für eine PCI-Karte mit einem FPGA darauf. Auf diese Karte wird ein weiterer Print mit einem A/D-Wandler gesteckt.

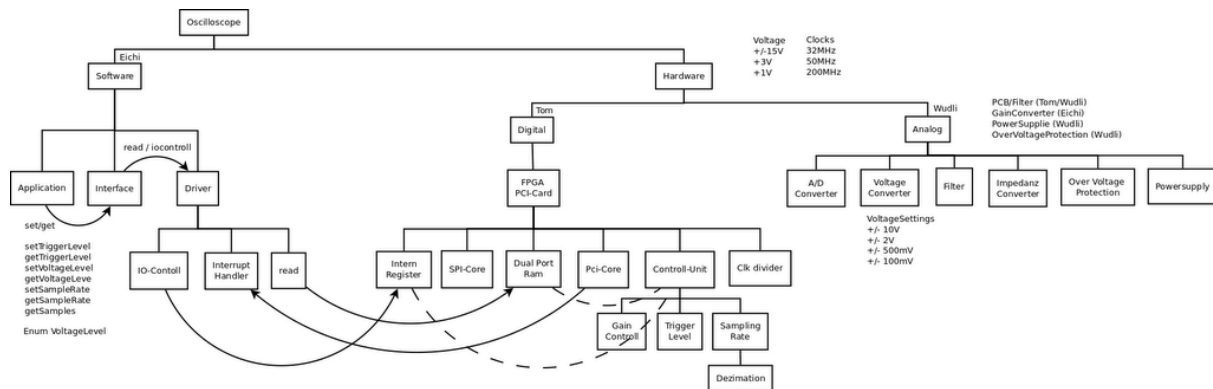


Abbildung 1 Übersicht über das gesamte Projekt

## 2 Hardware

### 2.1 Raggedstone Spartan 3-E PCI Development Board

Wir entschieden uns, um den Hardwareaufwand nicht noch mehr zu erhöhen, ein Board von der Firma Enterpoint (<http://www.enterpoint.co.uk/>) zu benutzen. Das Board besteht aus einem Spartan 3-E (XC3S400) und mehreren Peripherieausgängen.

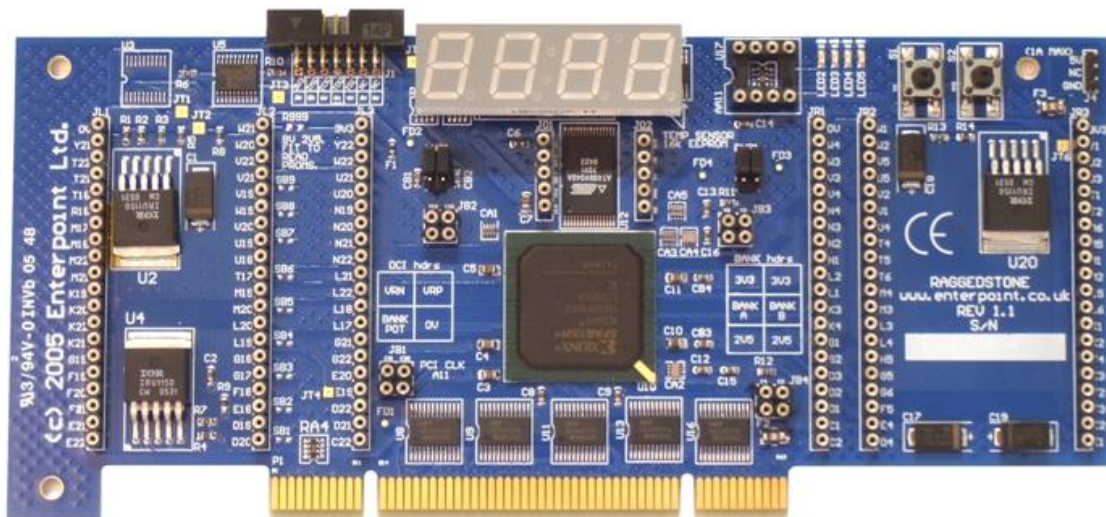


Abbildung 2 Raggedstone Spartan 3-E PCI Board

Weitere Informationen findet man unter:

<http://www.enterpoint.co.uk/moelbryn/raggedstone1.html>

## 2.2 Intel Atom Motherboard

Die PCI Karte alleine reicht noch nicht ganz für ein Oszilloskop. Damit der Anwender die A/D Werte sinnvoll weiterverwenden kann, müssen sie visualisiert werden. Diese Aufgabe übernimmt das Motherboard, basierend auf einer Intel Atom CPU (Intel NM10 Express). Auf dem Rechner läuft Ubuntu Linux 10.04 mit Kernel Version 2.6.32-16.

## 2.3 Aufbau

### Analog

Im Modul Elektronik 4 wurde im Analogteil, die Schaltung welche mit der PCI Karte kommuniziert gefertigt (Abbildungen).

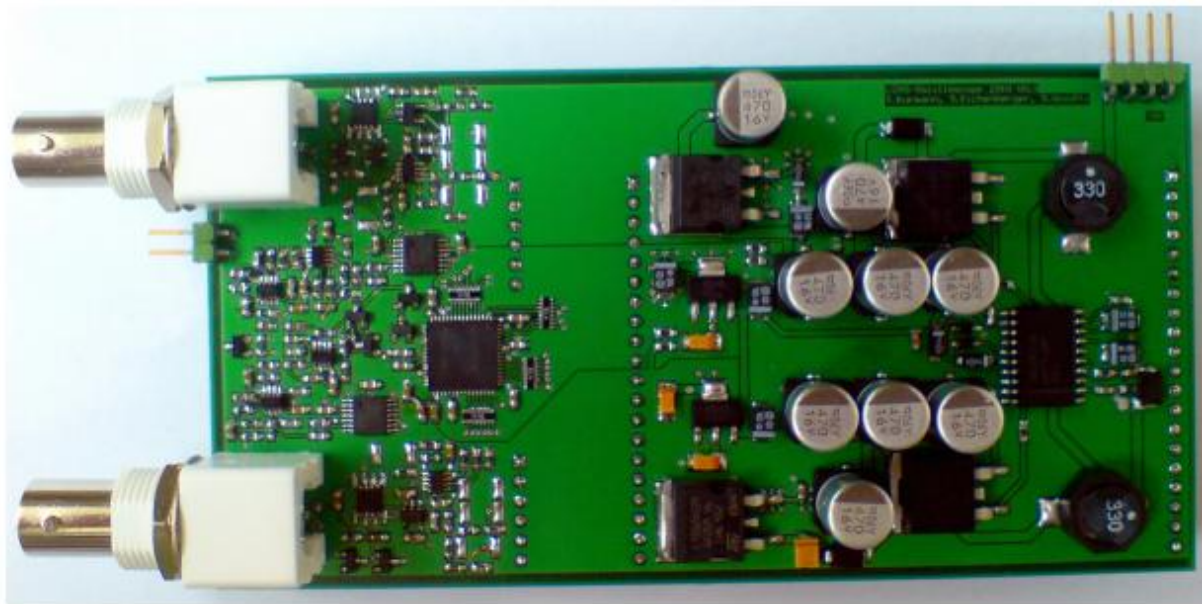


Abbildung 3 4-Layer A/D Print inklusive Powersupply



Abbildung 4 A/D Print Huckepack auf Raggedstone

Die Schaltung besteht aus zwei Kanälen und Speisungen. Die Speisungen bestehen aus Linearreglern, einer Z-Diode und einem Schaltregler. Die Schaltung des Reglers wurde vom Datenblatt übernommen. Da nicht alle angegebenen Bauelemente bei Farnell oder Distrelec gefunden wurden T.Kurmann, R. Woodtli, S. Eichenberger

der zu Teuer waren, setzten wir gleichwertige Ersatztypen ein. Die Eingangsstufe wurde zuerst mit Active Filter Designe Application von Texas Instruments entworfen und dann in LTSpice Simuliert. Danach wurde von uns ein Prototyp hergestellt. Das Layout machten wir auf vier Lagen, da wir Digitale und Analoge Signale sauber voneinander trennen mussten. Wir bestellten drei Leiterplatten, bei der Ersten bestückten wir nur den Digitalen Teil und die dazu benötigten Linearregler, damit wir den VHDL Code testen konnten. Auf die zweite Platine bestückten wir den Analogteil für einen Kanal um die Eingangsstufe zu testen und den Frequenzgang zu messen. Bei der Letzten Platine bestückten wir zuerst nur den Schaltregler. Danach bestückten wir einen Leiterplatte komplett um einen gesamten Prototyp zu erstellen.

Die genauere Dokumentation zur Hardware findet man im entsprechenden Dokument, auf unserer Projektseite.

## 3 Software

### 3.1 Ubuntu

Wir wählten als Betriebssystem das neue Ubuntu 10.04 mit der Kernel Version 2.6.32-16. Dies gibt uns einen einigermaßen sinnvolle Mischung aus Stabilität und neue Kernel Versionen. Ubuntu bietet zusätzlich eine schöne graphische Oberfläche (Gnome) um unser Java User-Space Programm zu implementieren.

### 3.2 Treiber

Wir schrieben den Treiber nicht von Grund auf neu, sondern er basiert auf einer Version von <http://projects.varxec.net/raggedstone1>. Wir konnten aber nur den Teil zum initialisieren des PCI device übernehmen, denn das read und der Interrupt fehlt in dieser Version ganz. Ausserdem musste auch das IOCTL völlig neu implementiert werden.

Als Hilfe diente uns das e-book <http://oreilly.com/catalog/linuxdrive3/book/index.csp>.



### 3.3 Treiber Design

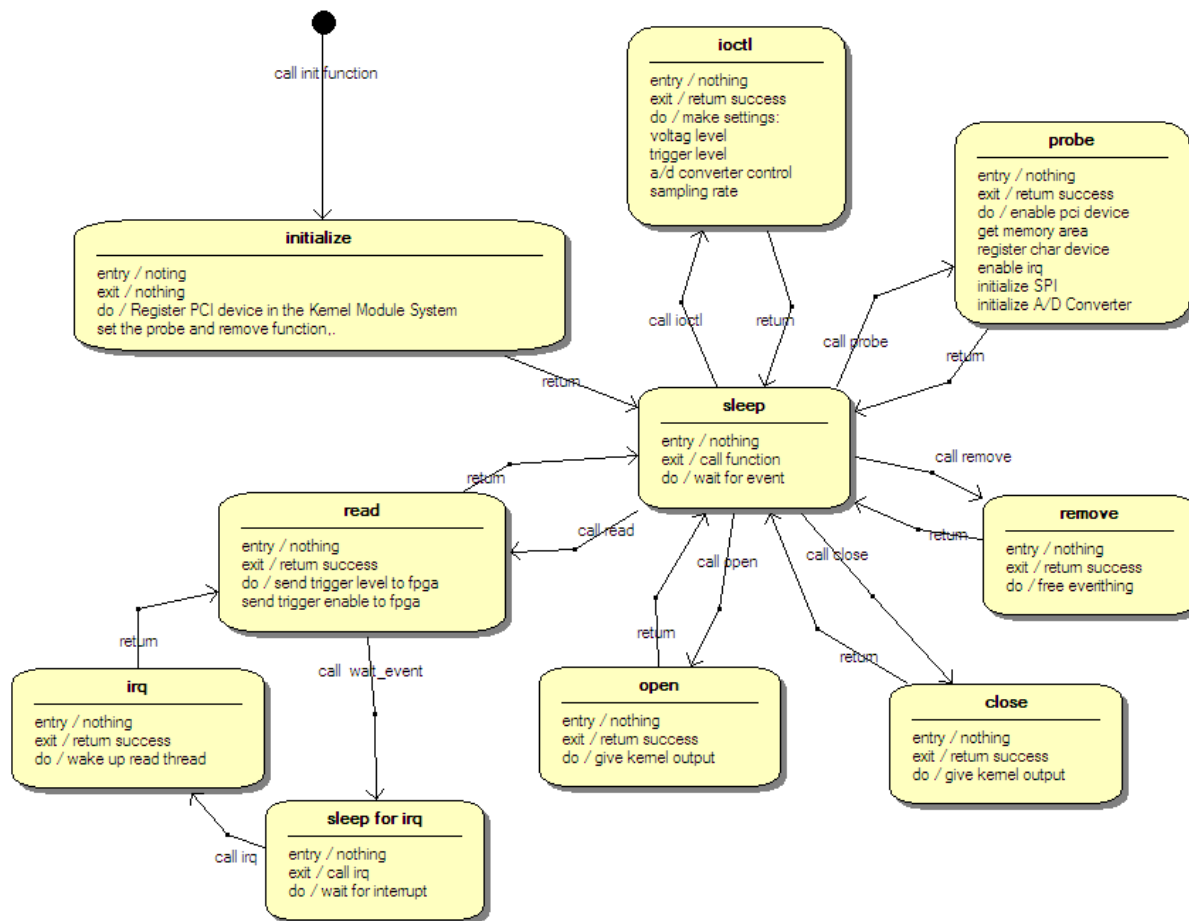


Abbildung 5 Treiber Layout

### 3.4 Beschreibung der wichtigsten Funktionen

#### 3.4.1 Initialize

Im State Initialize wird der Treiber initialisiert, das heisst in unserem Fall nur das dem Kernel mitgeteilt wird was für Funktionen aufgerufen werden müssen, wenn er ein PCI Gerät findet welches der Angegebenen Vendor und Produkt ID entspricht. Diese IDs müssten gekauft werden, da wir das jedoch nicht wollen nehmen wir diese von Xilinx.

```

VENDOR_ID_XILINX:    0x10ee
DEVICE_ID_XILINX    0x2010
  
```

#### 3.4.2 Probe

Wird nun ein Gerät gefunden, das unseren IDs entspricht, wird es geprobt. Hier findet nun die eigentliche Initialisierung statt. Denn nun kann man sicher sein, das es ein Gerät gibt, für welches der Treiber geschrieben wurde.

Es werden Speicher Adressen für PCI angefordert. Diese physikalischen Adressen müssen danach noch in den virtuellen Speicherbereich übertragen werden. Hat man das geschafft könnte man das PCI Gerät bereits ansprechen.

Damit man nun über eine Datei auf die verschiedenen Funktionen (read, ioctl) zugreifen kann, muss der Treiber auch noch als char device registriert werden.

Danach wird der Interrupt installiert und der SPI Controller wird initialisiert, um darüber dann den A/D Wandler zu initialisieren.

### 3.4.3 IOCTL

Über die IOCTL Funktion können alle Einstellungen am A/D-Wandler gemacht werden. Weiter können darin die verschiedenen Parameter für das A/D-Wandler Control Modul im FPGA gesetzt und verändert werden. Um zu verdeutlichen was bewirkt wird wenn man auf welche Adresse zugreift wurde ein Memory-Map erstellt.

SampleRate	0x000	0
BufferSize	0x004	
Reserved	...	
TriggerLevelA	0x014	
VoltageLevelA	0x018	
Reserved	...	
TriggerLevelB	0x028	
VoltageLevelB	0x02C	
Reserved	...	
Reserved	0x05C	92
0x060		96
...		
Samples CHA/CHB		
...		
0x4000		16384
SPIPSR	0x4004	16388
SPISPDR	0x4008	
SPISPER	0x400C	
SPISPCR	0x4010	16400
Reserved	...	
GPIO Trigger	0x4020	16416
GPIO SPI CS	0x4024	
GPIO Reserved	0x4028	
GPIO Reserved	0x402B	16427

Abbildung 6 Speicher Bereich im FPGA

Wir hoffen die Namen der ersten 96 Adressen sind eindeutig, weshalb nicht genauer auf die Funktion eingegangen wird. Es wird in 4er Schritten adressiert, da das FPGA 32Bit Daten hat. Die Samples des A/D-Wandlers können von den nächsten bis zu 4000 Adressen = 16000Byte gelesen werden.

Ab Adresse 16388 stehen die Register für den VHDL SPI Core.

SPIPSR: Status Register  
 SPISPDR: Data Register  
 SPISPER: Extension Register  
 SPISPCR: Control Register

Werden nähere Informationen benötigt so sollte das Datenblatt des SPI-Cores beigezogen werden, dieser stammt nicht von uns. Man findet alle Daten auf [http://opencores.org/project/simple\\_spi](http://opencores.org/project/simple_spi), oder können bei uns geholt werden.

Damit wir noch einige GPIOs benutzen können, oder gewisse Flags einfacher gesetzt werden können wurde der Speicherbereich oberhalb von 16416-16427 für diese Funktion vorgesehen. Der GPIO

Trigger startet im FPGA das Warten auf den Triggerlevel. Das GPIO SPI CS muss gesetzt/gelöscht werden um den SPI-Slave zu aktivieren.

#### 3.4.4 Read

Das Read dient zum auslesen der Sampling Daten. Zuerst wird der aktuelle Triggerlevel geschrieben, danach wird über das GPIO Trigger Flag das FPGA aufgefordert den Puffer zu füllen, sobald der Triggerlevel überschritten wird.

Damit wir keine Ressourcen verschwenden legt sich die Read Funktion schlafen. Sie wird nur geweckt wenn das FPGA genügend Daten gepuffert hat und ein Interrupt auslöst. Es ist aber auch möglich dass das Interrupt nicht genügend schnell kommt, und ein Timeout ausgelöst werden muss. Aktuell beträgt es 5s dieser Wert wird später sicherlich herunter korrigiert, ist aber für Testzwecke geeignet.

Die Daten werden dann vom FPGA ausgelesen und in den User-Space kopiert wo sie der Applikation zur Verfügung stehen.

### 3.5 C-Driver Interface

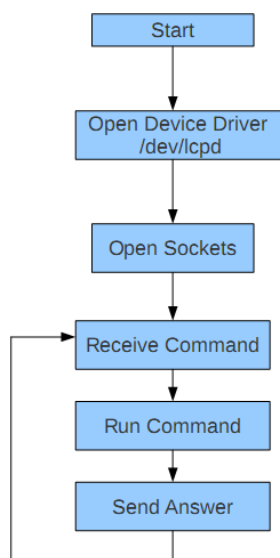


Abbildung 7 Software Design C-Interface

Das C-Interface wurde als Server Interface implementiert. Wird der Server gestartet versucht er sein Geräte Device zu öffnen und wartet danach auf Port 1234 auf eingehende Verbindungen. Das erste Zeichen das er jeweils empfangen will ist 67. Empfängt er dieses entschlüsselt er die verschiedenen Kommandos. Möglich ist zum Beispiel A/D Wandler initialisieren und danach Daten senden oder auch nur Daten senden, etc. Genauer kann man dem Kommentar im File entnehmen. Der Server ist als Schnittstelle zwischen Treiber und Userinterface gedacht.

Am Anfang hatten wir die Idee das C Interface über das Native C Interface von Java zu implementieren. Wir Überlegten uns danach aber das wenn wir eine Serverapplikation schreiben eine einfachere Portierbarkeit erreichen. Durch diesen Trick, muss nur der Server neu geschrieben werden um neue Treiber wie z.B. die Soundkarte zu unterstützen.

Dies kam uns dann auch zu gute, da wir leider mit dem VHDL Code nicht nachkamen und deshalb auf die Soundkarte auswichen. Dies funktioniert unter Linux soweit, dass man Eingaben über das



Mikrofon aufzeichnen kann. Leider stimmen die Timings noch nicht ausserdem hat man eine Verzögerung welche noch nicht behoben werden konnte. Man sieht aber immerhin wenn ein Signal durch das Mikrofon aufgezeichnet wird.

### 3.6 Java User Interface

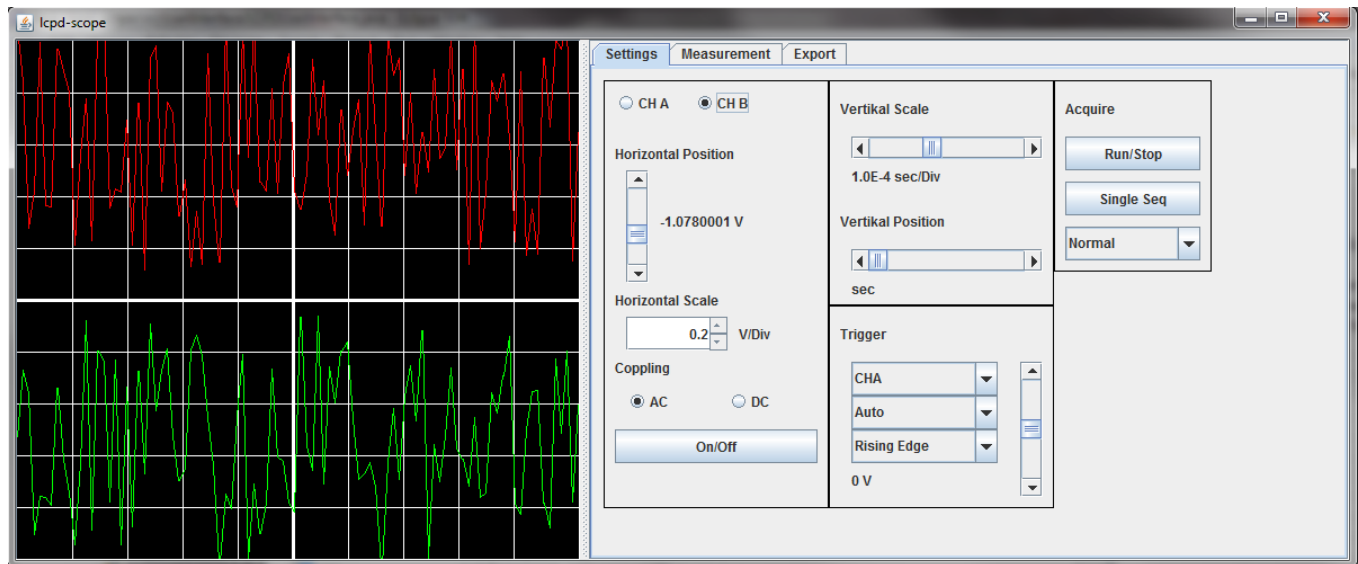


Abbildung 8 User Interface mit Reiter Settings zufalls Werte

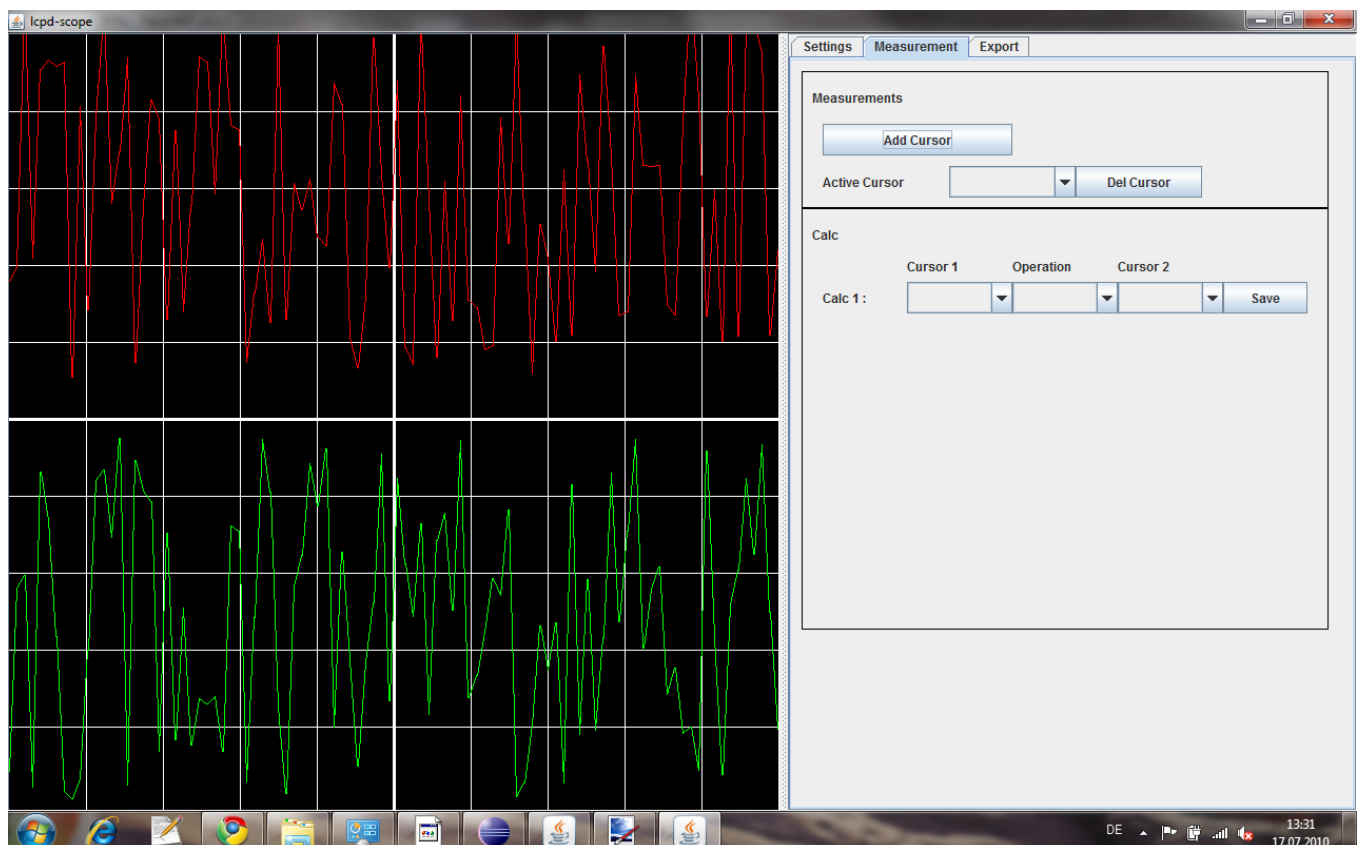


Abbildung 9 User Interface Maximiert mit Messurment Tools zufalls Werte

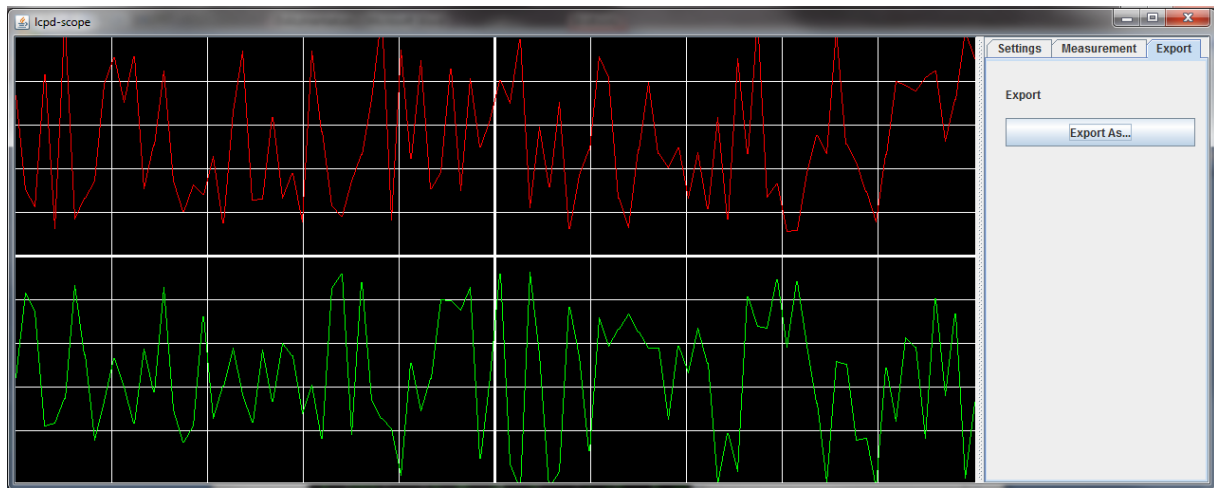


Abbildung 10 User Interface mit Export Reiter zufalls Werte

Das Java User Interface kümmert sich um die Darstellung der gesampelten Daten. Ausserdem kann die Darstellungsweise verstellt werden.

Nachfolgend die Funktionen, welche das GUI haben sollte.

- x-y Mode
- Anzeige Fenster passt sich der momentanen Fenster Grösse an
- Anzeige Bereich braucht fast die ganze höhe
- FFT-Mode
- Triggerlevel im Anzeige Bereich verstellbar aber auch optional mit Schieber
- Voltage Level mit + und - Buttons einstellbar
- Maus im Anzeige Bereich als Fadenkreuz
- Umschaltbar x100 x10 x1 Strom Shunt
- Messtools für Frequenz, Vpp, Veff,
- Zoom Funktion
- Export Funktion, cvs, jpg, png, plot
- Data stream over network sockets
- Single Shot, Normal, Automatic
- Channel Wahl
- Zeit schieben (Anfangs Zeit), Spannung schieben
- DC/AC umschalter
- Auto-Set
- Raster im Anzeige Bereich
- Trigger falling or rising edge

### 3.7 Java User Interface Design

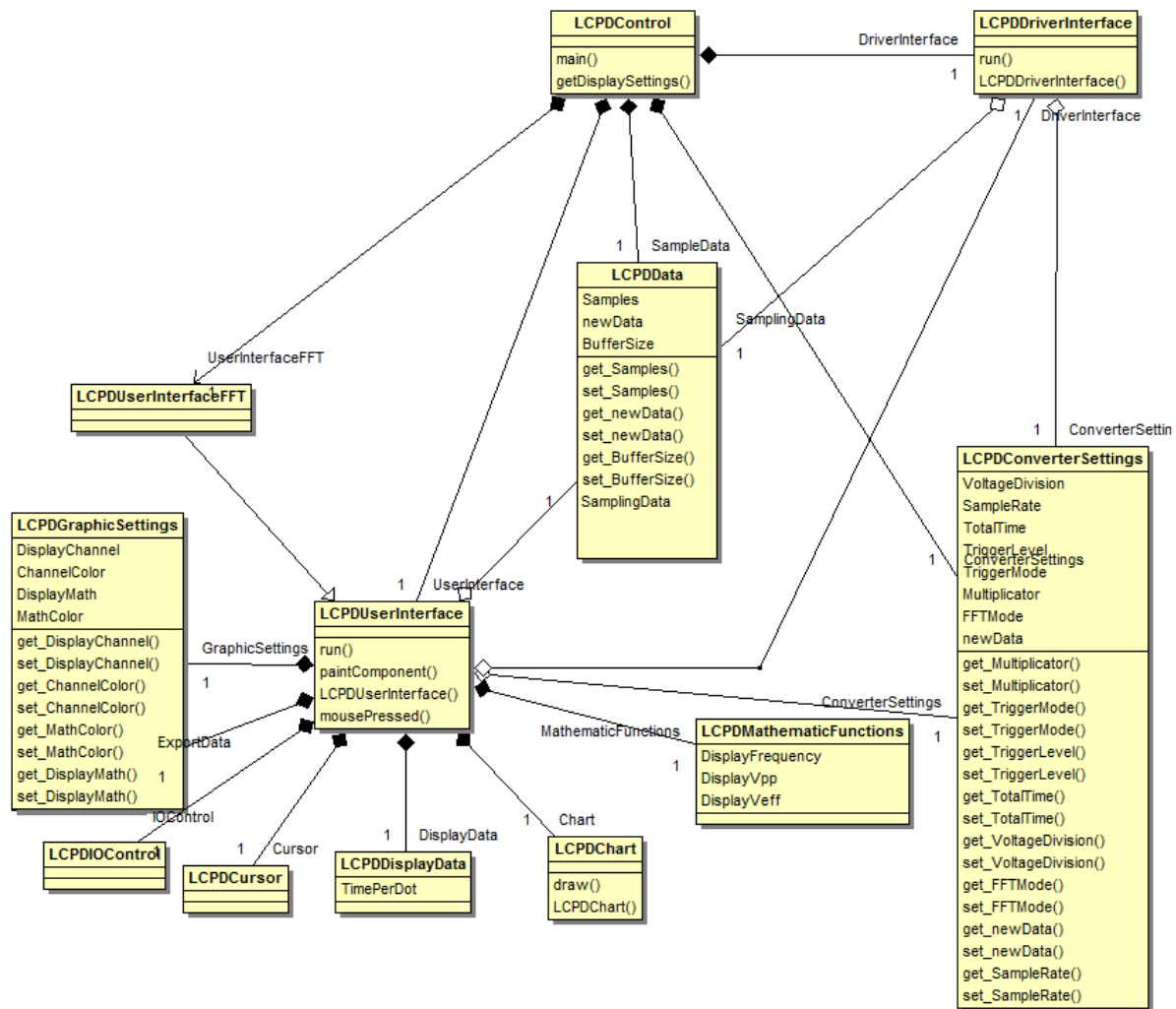


Abbildung 11 Klassenübersicht

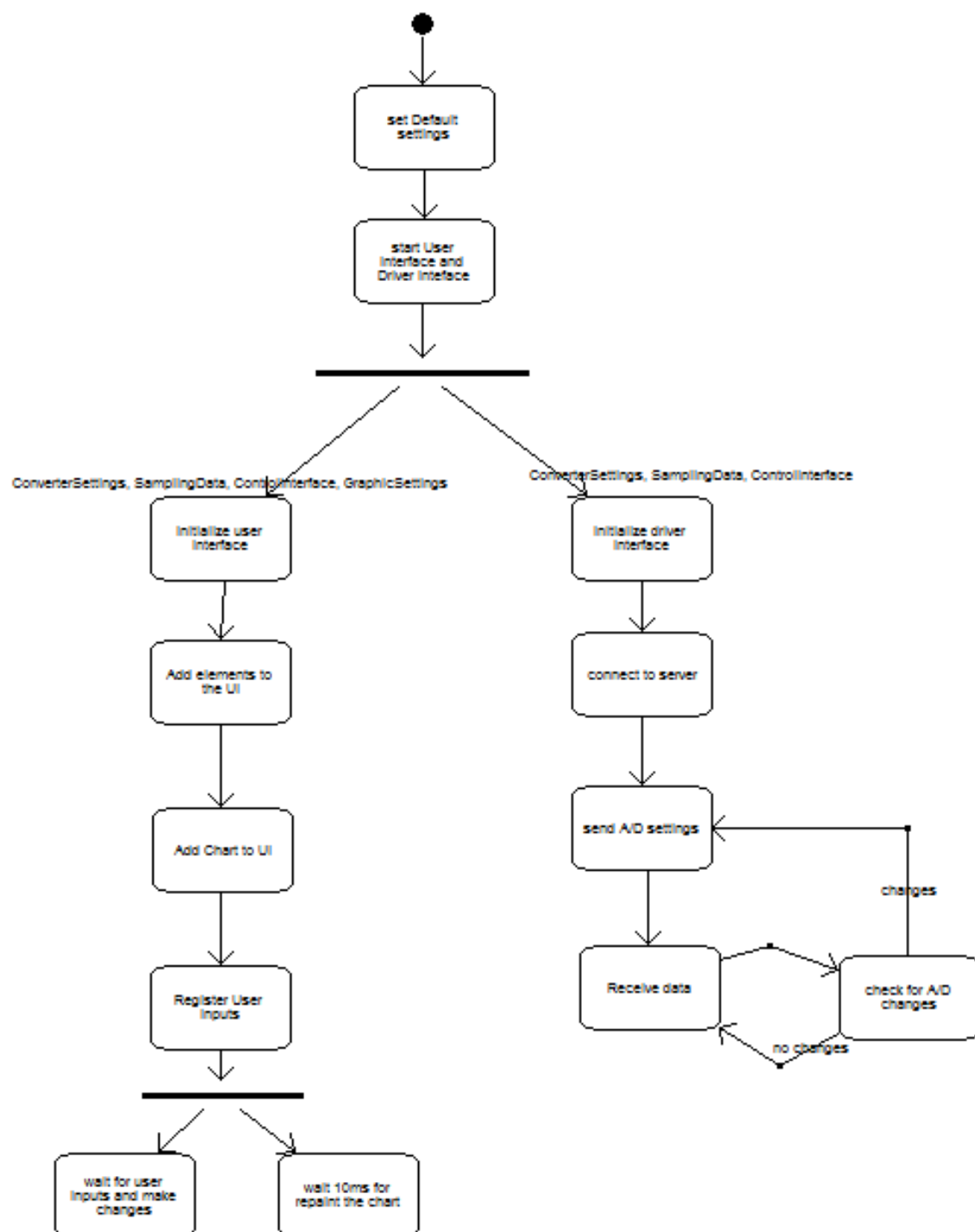


Abbildung 12 Aktivitäten bei Start

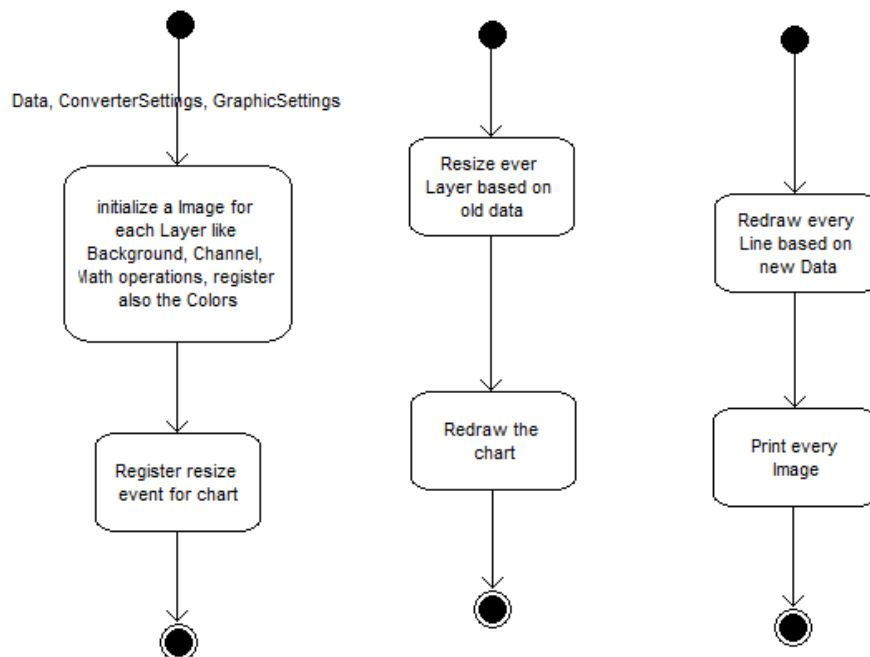


Abbildung 13 Aktivitäten der Kennlinie

### 3.7.1 LCPDControl

Aus dieser Klasse heraus startet das Programm. Es werden die Grundeinstellungen vorgenommen und die zwei verschiedenen Threads gestartet. Danach wird nur gewartet, dass die beiden anderen Threads zurückkehren um das Programm zu beenden.

### 3.7.2 LCPDUserInterface

Im User interface werden sämtliche Elemente in verschiedenen Containern platziert und diese dann dem Frame hinzugefügt. Durch das Platzieren der Elemente in Containern, ist das Layout flexibler. Man kann ganze Baukästen verschieben anstatt nur einzelne Elemente.

Danach werden für die Elemente Listener definiert, welche ihre entsprechenden Funktionen ausführen. So wird etwa durch das wechseln der Radio Buttons CHA/CHB die Einstellung des jeweiligen Kanals zum bearbeiten geladen. Durch verstellen am Vertical Scale Schieber wird die Zeit pro Division gesetzt.

Alle 20ms wird die LCPDChart aufgefordert sich neu zu zeichnen, dadurch ist sicher gestellt das immer die neusten Daten dargestellt werden. Da dass das Chart nicht gelöscht wird flimmert es auch nicht.

### 3.7.3 LCPDDriverInterface

Dies ist unsere Schnittstelle zum Server, es werden neue Einstellungen für den A/D Wandler oder den Treiber zur C-Schnittstelle gesendet und neue Daten empfangen. Dadurch dass es in einem separaten Thread läuft behindert es die Benutzer Schnittstelle nicht. Spannungen sind immer in Volt (float), Zeiten in Sekunden (float).

### 3.7.4 LCPDChart

In dieser Klasse wird sich um die Darstellung der Spannungswerte gekümmert. Die Daten werden immer auf die aktuelle Anzeige Grösse hochgerechnet. Im Zeit Bereich kann es auch passieren das Pro Pixel mehrere Spannungswerte dargestellt werden, dies haben wir auch bei Oszilloskopen beobachten, was zu einer stärker gesättigten Anzeige führt.

Alle Linien, der Hintergrund, die Anzeige von Einstellungen sowie der Cursor werden in einem separaten Bild aufgebaut der Bereich welcher keine Anzeige Daten enthält wird transparent. Durch diesen Trick können wir verschiedenen Layer generieren und diese dann plotten. Sie haben folgende Reihenfolge:

1. Background
2. Chart 1
3. Chart 2
4. Cursor (nicht implementiert)
5. Mathematik Funktionen (nicht implementiert)
6. Einstellungen (nicht implementiert)

Durch diese Layer sind wir extrem flexibel was wir anzeigen und exportieren wollen. Ausserdem muss auch nicht immer das ganze Bild neu gezeichnet werden, sondern es reicht im Normalfall nur die Linien neu zu zeichnen, was wesentlich schneller geht als auch noch den Hintergrund.

### 3.7.5 LVPDData

Enthält die Daten von dem A/D-Wandler bzw. von einem Treiber.

### 3.7.6 LCPDConverterSettings

Enthält die Einstellungen für den A/D-Wandler bzw. den Treiber.

## 4 Tests

Funktion	Ergebnis
x-t Modus	i.O.
x-y Modus	Nicht implementiert
Fft Modus	Nicht implementiert
Anpassen der Anzeige grössse auf Fenstergrösse	i.O.
Triggerlevel in Anzeige verstellbar	Nicht implementiert
Voltage Level Verstellbar	i.O.
Maus als Fadenkreuz im Anzeigebereich	Nicht implementiert
Umschalter von x100 x10 Strom	Nicht implementiert
Mess/Math Tools	Nicht implementiert
Zoom Funktion	Nicht implementiert
Export Funktion	Nicht implementiert
Data stream over Network	Läuft, ist aber noch fehlerhaft. Vor allem der Server hat noch Fehler
Automatic Modus	Auf Anzeige Seite i.O. da jedoch noch nicht alle Funktionen des UI implementiert sind gibt es noch Fehler
Normal Mouds	Auf Anzeige Seite i.O. da jedoch noch nicht alle Funktionen des UI implementiert sind gibt es noch Fehler



Single Shot	Auf Anzeige Seite i.O. da jedoch noch nicht alle Funktionen des UI implementiert sind gibt es noch Fehler. Ausserdem fehlt auch noch die Funktion im driver interface.
Channel wahl	i.O.
Zeit schieben	Nicht implementiert
Spannung schieben	i.O. hat noch einige Tücken welche auskorrigiert werden müssten
Auto set	Nicht implementiert
Raster im Anzeige bereich	i.O.
Trigger falling rising edge	Keine Funktion
C-Driver Interface	Funktioniert teilweise, ist aber nicht Threadsave und liest Daten verzögert ein. Ausserdem wurde nur ein Kanal implementiert und es wird vom Mikrophon aufgezeichnet.

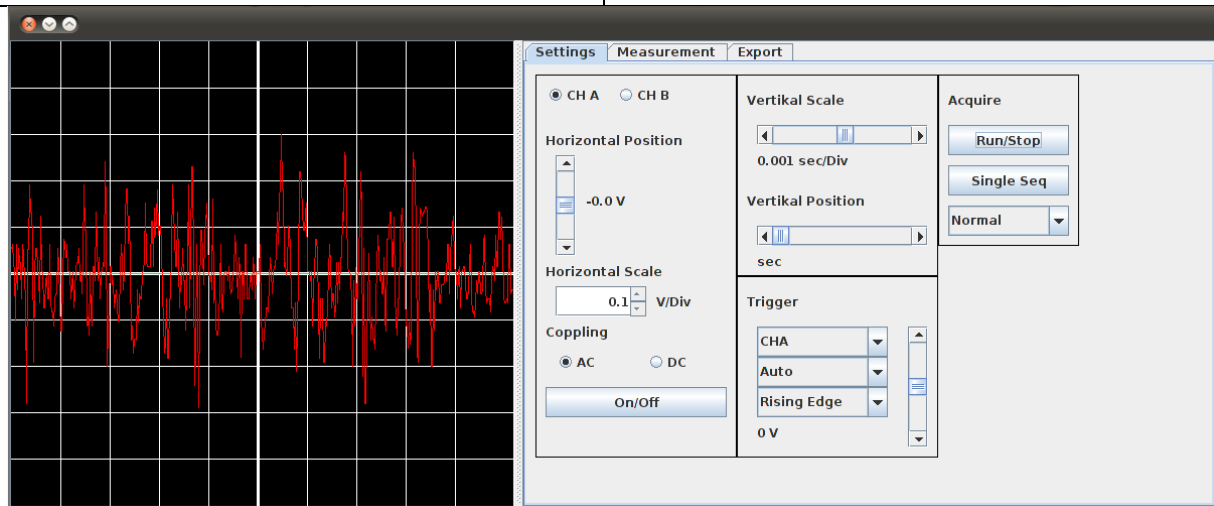


Abbildung 14 Aufzeichnung klopfen von Mikrophon

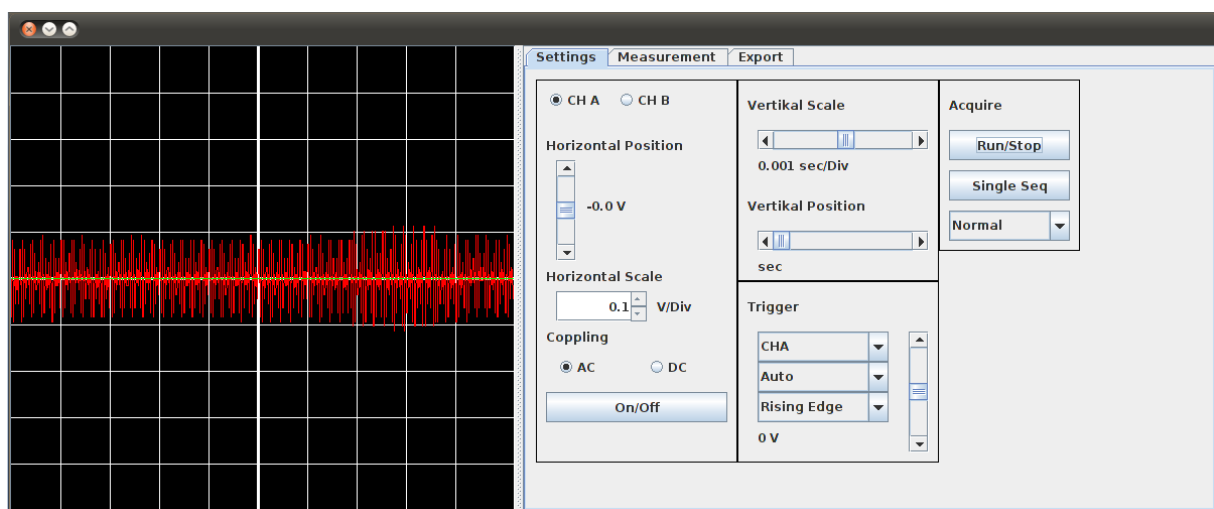


Abbildung 15 Aufzeichnung Pfeifen

## 5 Bedienungsanleitung

Will man das Programm testen, hat man zwei Möglichkeiten. Beim auf starten wird man gefragt ob man mit simulierten Daten oder Daten vom Server Darstellen will. Hier sollte man Simulierte Daten wählen es sei denn man hat unter Linux den Server gestartet.

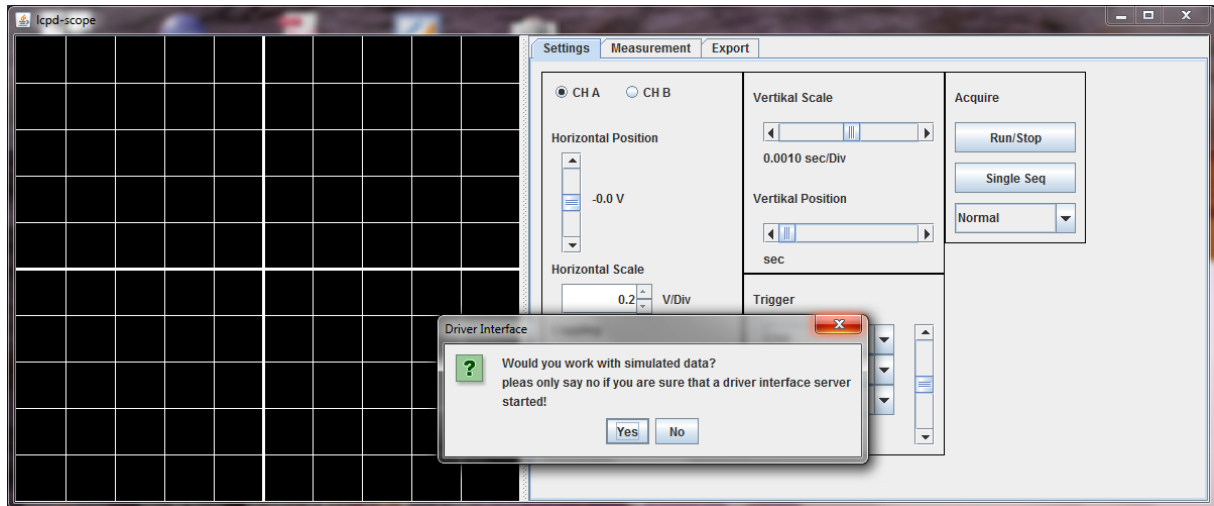


Abbildung 16 Abfrage bei start

Das User Interface sollte dann selbsterklärend sein, es ist jedoch zu beachten, dass viele Funktionen noch nicht implementiert sind!

Will man den Server ausprobieren, hat man zuerst den Server unter Linux zu starten (aktuell funktioniert er nur dort) und danach die Java Applikation auf demselben Computer. Danach sollten die Daten gestreamt werden.

```
lcpd-scope-server [C/C++ Application]
Communication Thread: started
Waiting for Connection |
Sample Thread: started
```

Abbildung 17 C-Server Interface

## 6 Schlussbetrachtung

Der Zeitplan in diesem Projekt über alle Fächer war von Anfang an sehr eng und auf eine Woche in der UFZ ausgedehnt worden. Als wir dann vor allem im VHDL auf unerwartet grosse Probleme stiessen, hatten wir Schwierigkeiten die Aufgaben noch sinnvoll einzuteilen. Deshalb fuhren wir in

der Informatik mit den Anforderungen etwas zurück, das Ziel wäre gewesen ein Signal von unserem Print darzustellen. Leider blieb der VHDL teil innerhalb der kurzen Zeitspanne aber ein unlösbares Problem, weshalb das nicht funktionierte. Dadurch sahen wir uns gezwungen auf die Soundkarte umzusteigen, was uns teilweise auch gelang.

Dadurch dass alle Teammitglieder ab der zweiten Woche arbeiten mussten/durften, hatten wir danach nicht mehr die Zeit diesen Ansatz vollständig umzusetzen.

Der ganze Java Code ist aber durch die Anfängliche Projektdefinition sehr gut erweiterbar. Die meisten Gerüste für die verschiedenen Funktionen sind immer noch vorhanden, wurden aber nicht implementiert. Dadurch müsste es möglich sein aus unserem Ansatz ein brauchbares Projekt zu kriegen.

Die Arbeiten waren interessant, wir lernten viel über Oszilloskope und deren Eigenheiten. Leider war es uns auch mit zusätzlicher Zeit nicht möglich unsere Erwartungen zu erfüllen. Schlussendlich war im wahrsten Sinne des Wortes die Zeit abgelaufen.