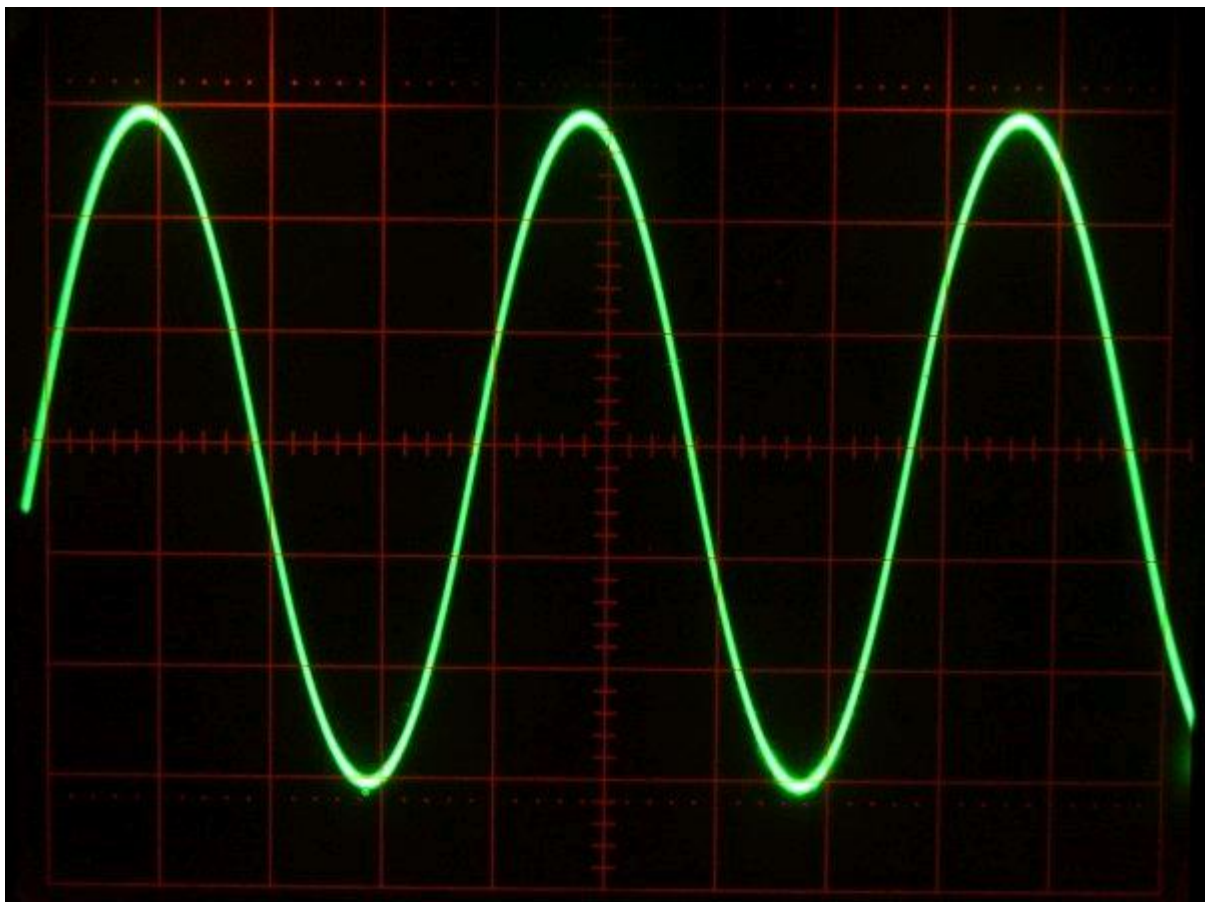


Low Cost PCI Digital-Scope

Linux Driver



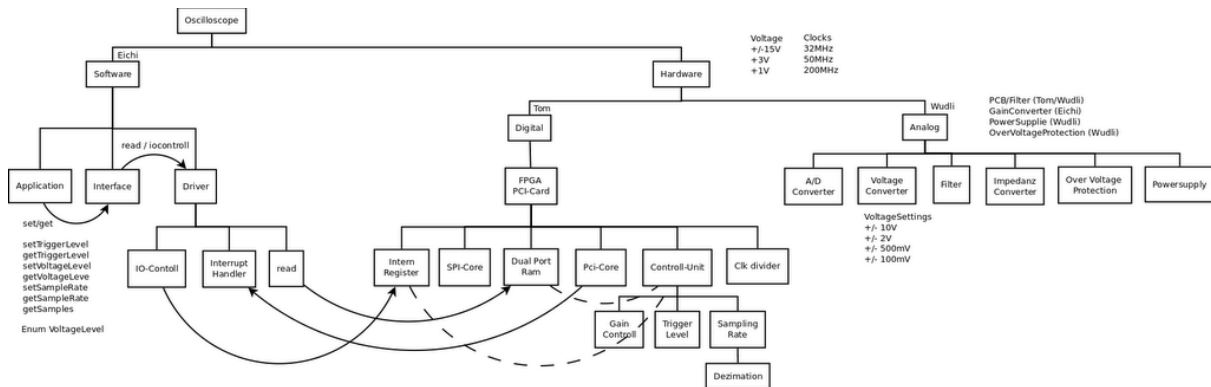
Projekt Homepage: <http://code.google.com/p/lcpd-scope/>
Autoren: T. Kurmann, R. Woodtli, S. Eichenberger
e-mail: kurmt1@bfh.ch, woodr1@bfh.ch, eichs2@bfh.ch
Dozent: R. Weber

Table of Contents

Einführung.....	1
Hardware	1
Raggedstone Spartan 3-E PCI Development Board.....	1
Intel Atom Motherboard.....	1
A/D Wandler ADC0820	2
Testaufbau.....	3
Software.....	4
Ubuntu.....	4
Treiber.....	4
Treiber Design.....	5
Beschreibung der wichtigsten Funktionen.....	5
Initialize	5
Probe	5
IOCTL.....	6
Read	7
Demonstration (Programm linuxPlot)	7
Probleme.....	9
Schlussbetrachtung.....	9

Einführung

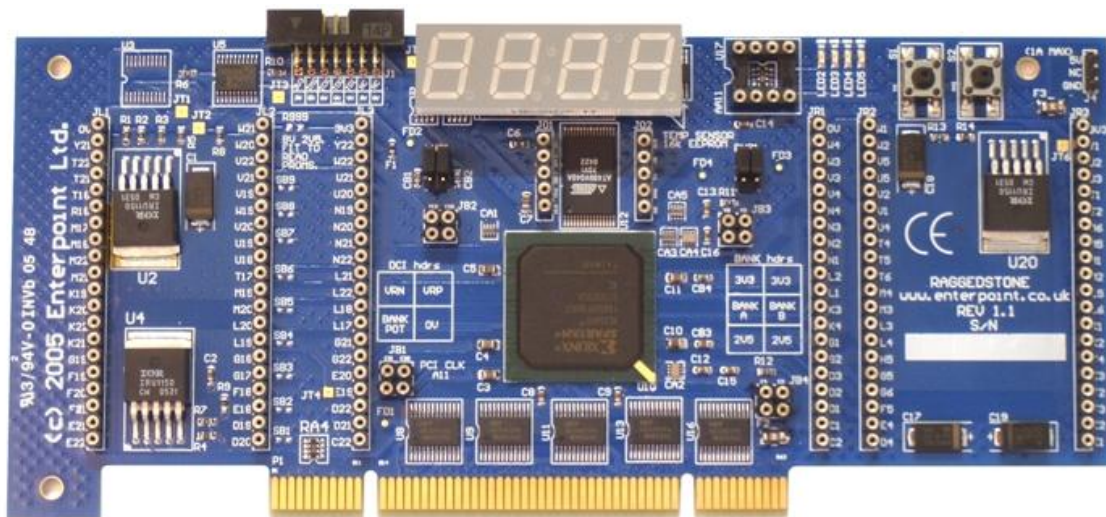
Als Teil einer fachübergreifenden Projektarbeit wurde das Low Cost PCI Digital-Scope (LCPD-Scope) ins Leben gerufen. Das Ziel dieser Arbeit ist es, ein Oszilloskop mit einer Bandbreite von 20MHz zu entwickeln. Die Daten sollen an jedem beliebigen Computer erfasst werden können. Deshalb entschieden wir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



Hardware

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.



Weitere Informationen findet man unter:

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

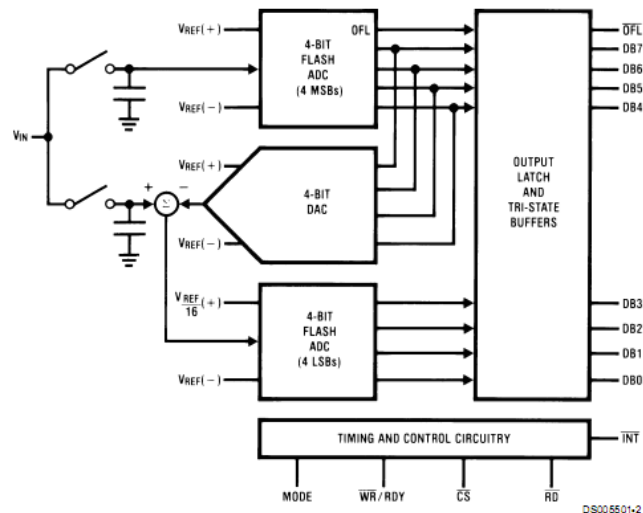
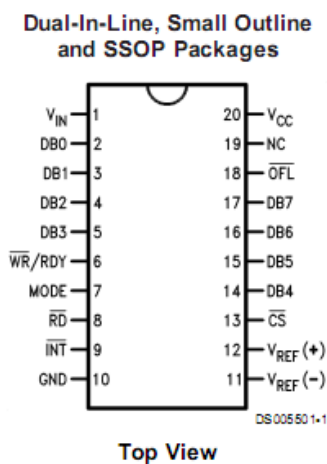
Intel Atom Motherboard

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

Motherbaord, basierend auf einer Intel Atom CPU (Intel NM10 Express). Auf dem Rechner läuft Ubuntu Linux 10.04 mit Kernel Version 2.6.32-16.

A/D Wandler ADC0820

Weil nicht genug Zeit vorhanden war bis zum Abgabetermin des Linux Treibers haben wir anstatt den originalen 200 MSPS A/D Wandler (ADSP62P28 von Texas Instruments), einen kleinen, einfachen parallelen 8 Bit A/D Wandler (adc0820 von National Semiconductor) ins Design integriert. Dies ermöglichte uns den Treiber mit reellen Daten zu testen und dann auch in unserem User-Space Programm zu visualisieren.



Die Ansteuerung wurde im FPGA implementiert und ist für den PC nicht sichtbar.

Testaufbau



Abbildung 1 Gesamtaufbau

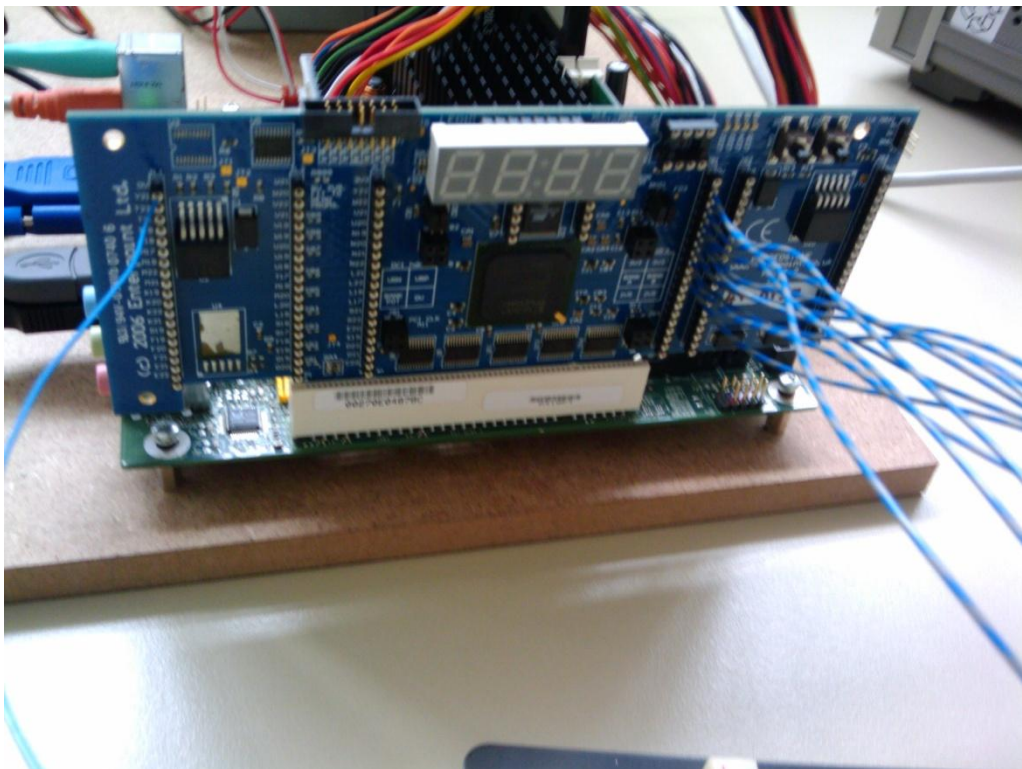


Abbildung 2 Raggedstone 1

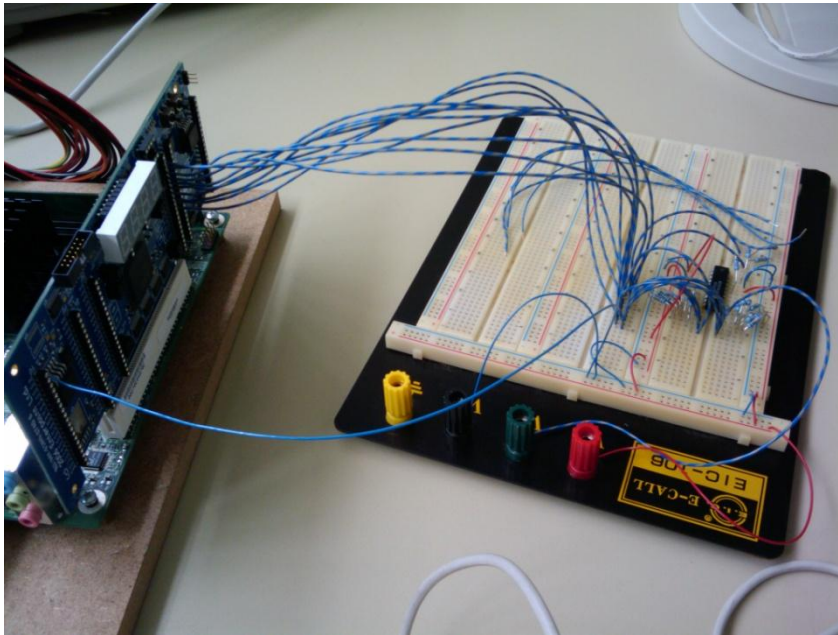


Abbildung 3 ADC0820 Beschaltet

Da das es bei diesem Teil der Projektarbeit nur um die Treiberentwicklung ging, gehen wir nicht näher auf den Testaufbau ein.

Software

Ubuntu

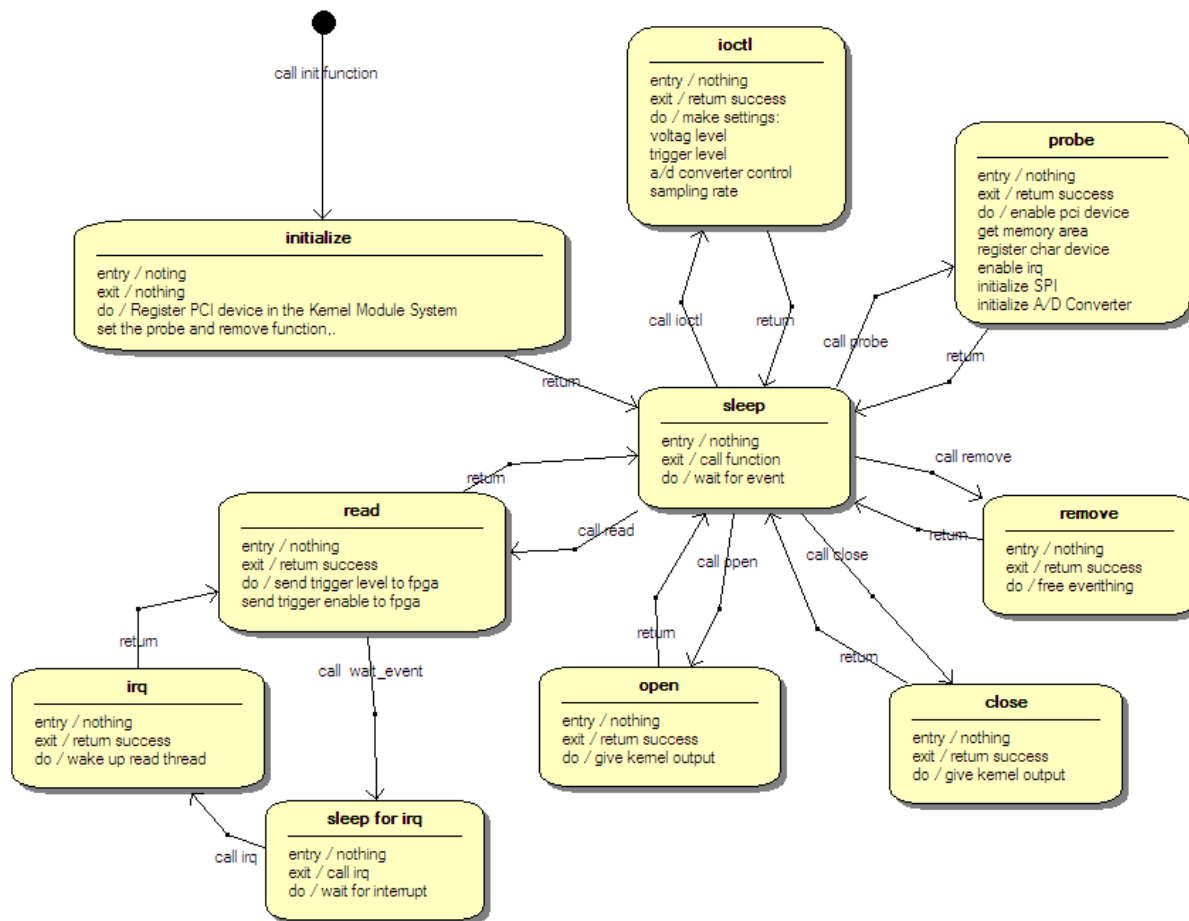
Wir wählten als Betriebssystem das neue Ubuntu 10.04 mit der Kernel Version 2.6.32-16. Dies gibt uns einen einigermaßen sinnvolle Mischung aus Stabilität und neue Kernel Versionen. Ubuntu bietet zusätzlich eine schöne graphische Oberfläche (Gnome) um unser Java User-Space Programm zu implementieren.

Treiber

Wir schreiben 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>.

Treiber Design



Beschreibung der wichtigsten Funktionen

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
  
```

Probe

Wird nun ein Gerät gefunden, dass 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.

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
SPISPSR	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

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.

SPISPSR: 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, 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.

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.

Demonstration (Programm linuxPlot)

Zur Demonstration schreiben wir ein kleines Programm, welches Daten von einem adc0820 A/D-Wandler von National über das FPGA einliest und danach über GNU-Plot darstellt. Es ist sehr einfach gehalten demonstriert aber, welchen Zweck der Treiber erfüllen muss. Im Elektronik Projekt wird dann der adc0820 durch den ADS68P28 ersetzt welcher jedoch etwas komplizierter angesteuert werden muss und ausserdem ein SMD-Gehäuse besitzt wodurch wir ihn nicht auf einem Steckbrett verwenden konnten.

Screenshots zum Programm:

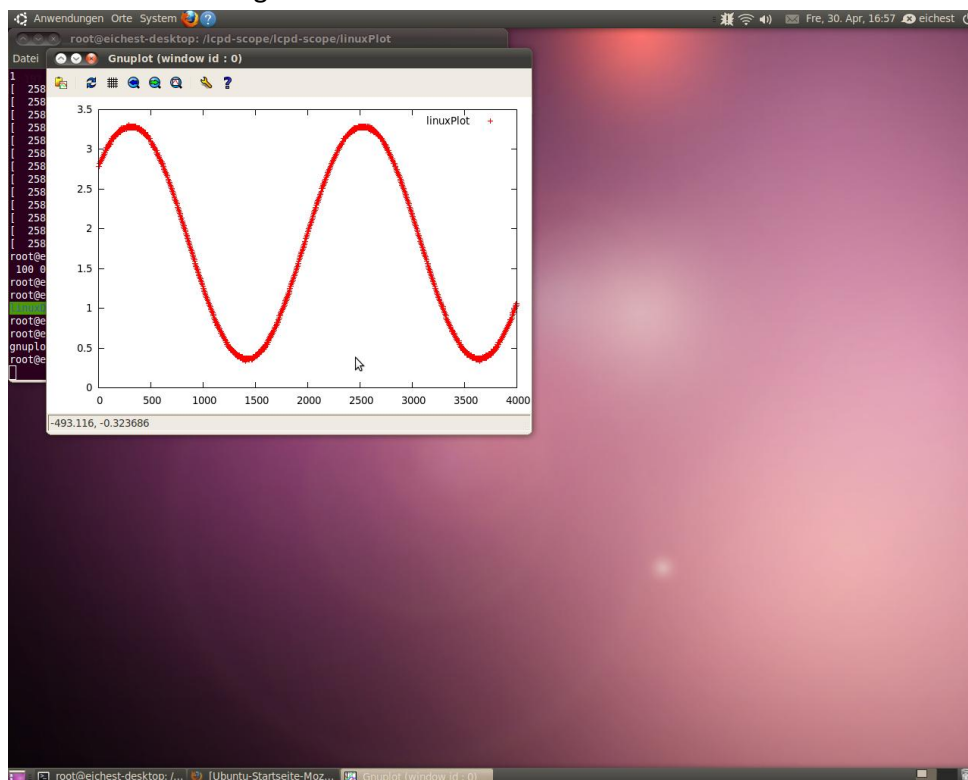


Abbildung 4 Sinus in linuxPlot unter Ubuntu 10.04

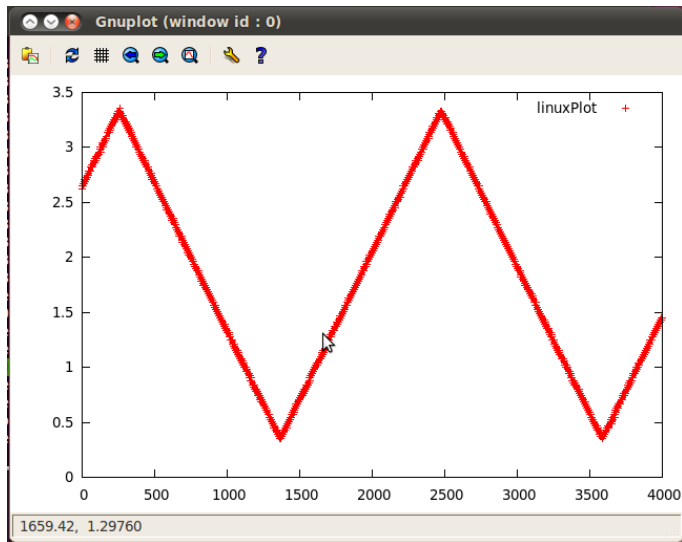


Abbildung 5 linux Plot Darstellung eines Dreiecksignals

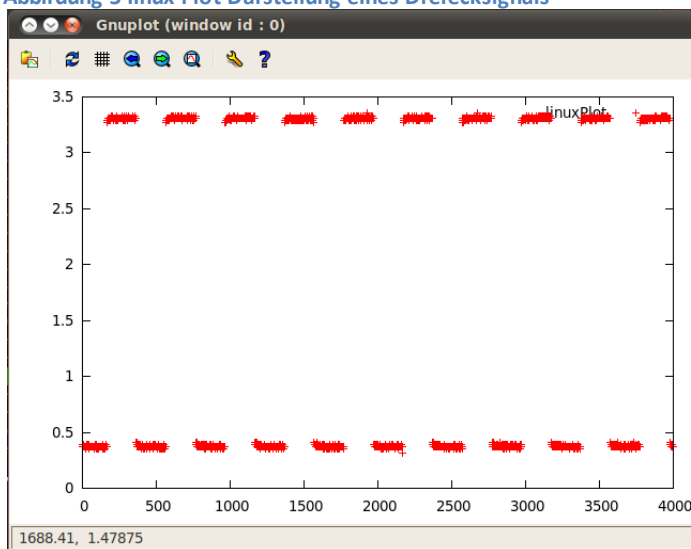


Abbildung 6 linuxPlot Darstellung eines Rechtecksignals

Wie man sehen kann wird die Spannung richtig skaliert, die Zeit jedoch nicht umgerechnet. Dies aus dem Grund, dass bei unserer Implementierung im FPGA die Zeit leicht verschieden sein kann. Mit dem neuen A/D-Wandler ist das berechnen kein Problem mehr, da wir dort die Samplingrate einstellen können.

Wie man das Programm verwenden muss, kann man dem Readme entnehmen. Messungen zu SPI

Da man den ADS68P28 über eine SPI-Schnittstelle konfigurieren kann, besitzt der FPGA auch einen SPI-Core. Zum Testen dieses Cores zeichnen wir das Ausgangssignal mithilfe eines Logic-Analysers auf.

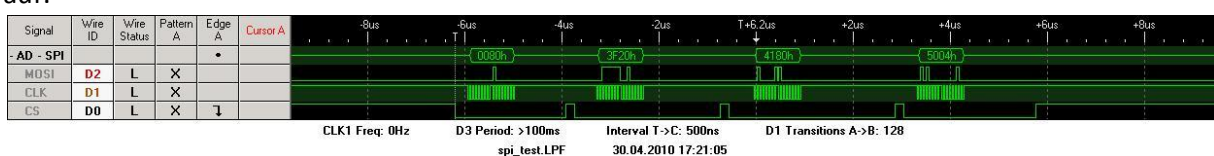


Abbildung 7 SPI Initialisierung des A/D-Wandlers

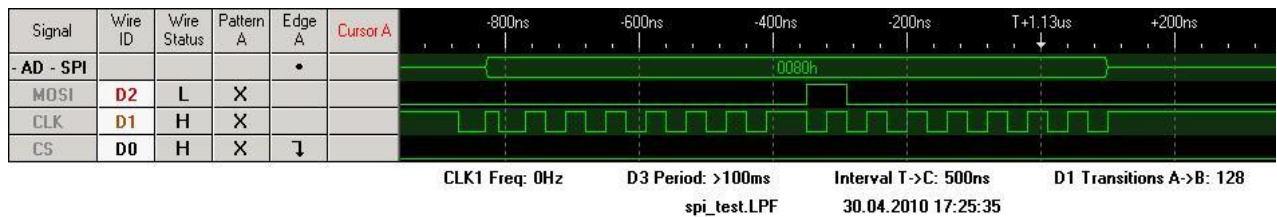


Abbildung 8 SPI erstes Frame

Die Pakete die zur Initialisierung geschickt werden müssen, scheinen unserer Ansicht nach richtig zu sein.

Zur Erklärung, das Frame ist 16-Bit orientiert. Die ersten 8Bit übertragen die Register Adresse (in Abb.5 also 0) und dann den Initialisierungswert (in Abb.5 0x80).

Probleme

Bei der ganzen Implementierung stiessen wir auf ein grösseres Problem dies aufgrund der schnellen Weiterentwicklung des Kernels.

So steht in den meisten Quellen die Interrupt Nummer finde man ganz einfach über die Funktion mit folgenden Parametern `pci_read_config_byte(dev, PCI_INTERRUPT_LINE, &myirq);` heraus. Das stellte sich jedoch als falsch heraus. Man findet die Nummer über `dev->irq` (eigentlich noch einfacher!). Interessanterweise gab aber die Funktion auch keinen Fehler zurück, sondern behauptete es habe geklappt.

Ein weiteres Problem war das `copy_to_user` mit der falschen Puffergrösse rechnete da wir vergassen das Ganze noch `* sizeof(int)` zu rechnen, dies führte bei den ersten Versuchen ständig zum Absturz.

Schlussbetrachtung

Alles in allem funktioniert unser Treiber aktuell soweit, dass wir mit der Hardware kommunizieren können. Es war uns sogar möglich einen 8Bit A/D-Wandler erfolgreich auszulesen und mit GNU-Plot darzustellen. Im Verlauf unseres Projekts wird es sicherlich noch einige Änderungen bezüglich Adressen und neuer Einstellungen geben, diese dürften aber nicht mehr sehr schwer zu implementieren sein.

Eine schöne und möglicherweise in Zukunft neue Methode die Werte auszulesen wäre, dazu die DMA des Prozessors zu benutzen. Wir erhoffen uns dadurch eine schnellere Datenübertragung. Denn mit den Momentanen Einstellungen kommen wir nur auf etwa 10-20MB/s. Möglich sollten jedoch mehr als das doppelte sein (momentan 5MHz*4 Byte, theoretisch 33MHz*4 Byte).