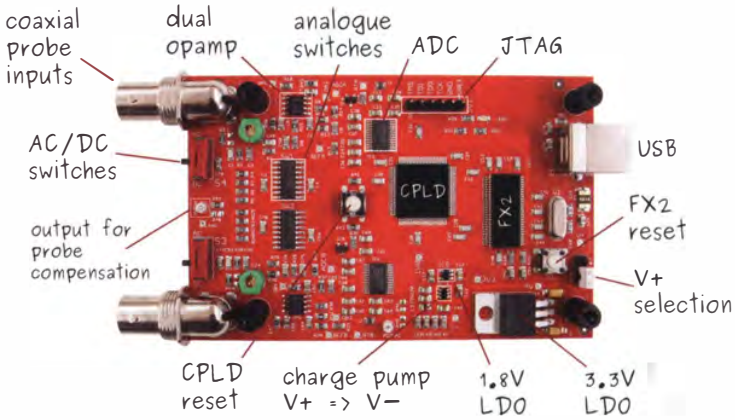
Jelly Scope



Open Source Android-Oszilloskop

Das JellyScope heißt eigentlich mit Geburtsnamen OsciPrime und ist eine Machbarkeitsstudie. Die Autoren wollten zeigen, dass Android-basierte Rechner-Anwendungen, gleich ob Singleboard-Computer, Tablet-PC oder Smartphone, sehr wohl für den Technik-Einsatz, wenn nicht gar für industrielle Anwendungen geeignet sind. Die hier beschriebene Speicher-Scope-Hardware ist per Creative Commons, die Software per GPLv2 lizenziert.

Manuel Di Cerbo und Andreas Rudolf, Nexus-Computing

Für die Entwicklung eines Software-Oszilloskops mit fast so vielen Features wie ein übliches Hardware-Oszilloskop wurde das Android Application Framework verwendet. Die Applikation besteht aus einer Android-Activity für die Interaktion des Anwenders und die Anzeige von Daten, die durch ein externes Hardware-Frontend - hier ein Android Smartphone oder ein Tablet - erfasst werden. Die Aktivität bezieht sich also auf Eingaben des Anwenders und die grafische Ausgabe. Der Dienst behandelt und steuert die Datenerfassung in Echtzeit. Darüber hinaus unterstützt die Android-Applikation ein USB-Host-Interface, was einen Programmteil erfordert, der in C oder C++ geschrieben ist, da Android keine API hat, die via USB-Host-Mode Zugriff auf angeschlossene Geräte ermöglicht.

Zu den Zielen gehörte, dass die Software keine erfassten Samples verlieren und eben echtzeitfähig sein sollte. Eine Anforderung war, dass neben einem Continuous mode, der permanent Daten auf dem Bildschirm erneuert, auch ein Single shot mode vorhanden sein sollte, der über einen Hardware-Trigger verfügt und Signalteile hochauflösend darstellen kann (mit 48 MHz abgetastet). Ein Wunsch war, auch Audio von einem normalen Handy als Datenquelle nutzen zu können. Das führte zu der Idee, eine Entwickler-API zu erstellen, die mit jeder zusätzlichen Datenquelle umgehen kann (Audio, USB, Bluetooth, Netzwerk-Stream, Datei-Stream etc.). Aus diesen Gründen kann man das Oszilloskop-Framework sehr universell einsetzen.

USB-Verbindung

Um die hohen Abtastraten zu erreichen, wurde für die Verbindung des externen Frontends mit dem Android-Gerät USB gewählt. Mit einem erreichbaren Durchsatz von 40 MB/s stellt diese Schnittstelle genug Geschwindigkeit für eine ausreichende Sample-Rate zur Verfügung. Da bei einem vorangegangenen Projekt schon Erfahrungen mit dem FX2-USB-Mikrocontroller von Cypress gesammelt wurden, kam dieser Controller bei diesem Projekt wieder zum Einsatz. Es gibt allerdings große

Unterschiede zur vorherigen Spectrum-Analyzer-Applikation. Allein die Datenrate ist jetzt um den Faktor 1.000 höher. Daher ist das Slave-FIFO-Feature des FX2-Controllers genutzt worden, um die Daten schnell einsammeln und mit einem FPGA oder CPLD austauschen zu können.

Hardware

Für das Projekt wurde eine extra Platine für die Signalerfassung entwickelt. Das Board wird über USB versorgt und hat ein analoges sowie ein digitales Frontend. Beim analogen Frontend können die zu messenden Signale einstellbar verstärkt werden, so dass sie passend mit A/D-Konvertern abtastbar sind. Die gesampelten Signale werden dann an ein CPLD übergeben, das als Master für die FIFO-Queue des FX2-Mikrocontrollers fungiert.

Analog & digital

Das analoge und das digitale Frontend wurden auf einem einzigen Board kombiniert. Per CPLD werden die digitalisierten Samples an den FX2-Mikrocontroller übergeben. Die erfassten Daten werden per USB übertragen, und mit Hilfe eines I²C-Interface werden die Setup-Infos an das CPLD übergeben.

Das bei diesem Projekt entwickelte externe Frontend beherbergt einen analogen und einen digitalen Teil. Bild 1 zeigt das Konzept des Boards, Bild 2 das komplette Schaltbild. Sowohl das analoge als auch das digitale Frontend werden jeweils detailliert in den folgen Abschnitten besprochen.

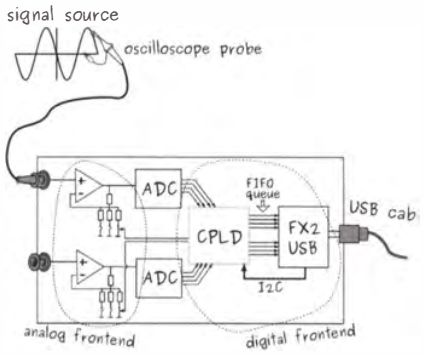


Bild 1. Das Konzept des JellyScope-Boards.

Signalpfad

Das analoge Frontend beginnt bei der Spitze des Oszilloskop-Tastkopfs und endet am Eingang des A/D-Konverters. Auf diesem Weg muss das zu messende Sig-

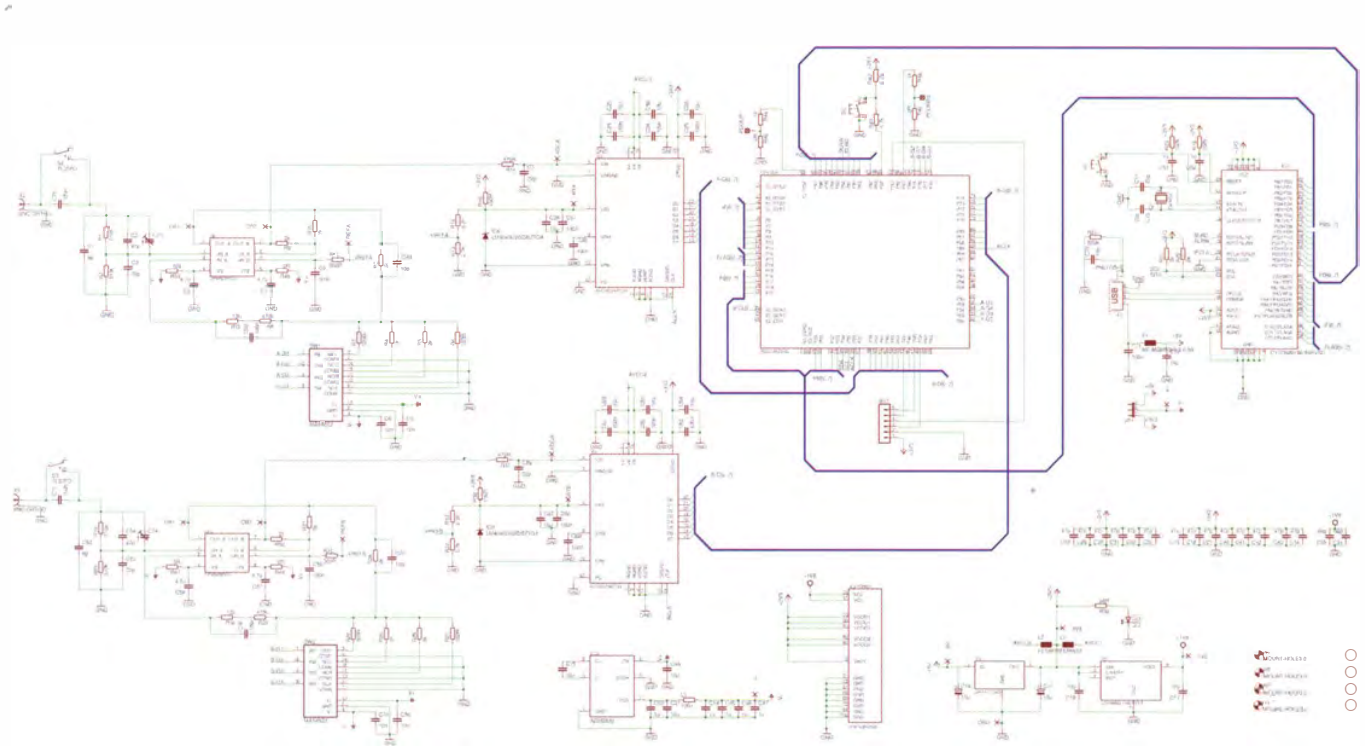


Bild 2. Das komplette Schaldbild des Frontends.

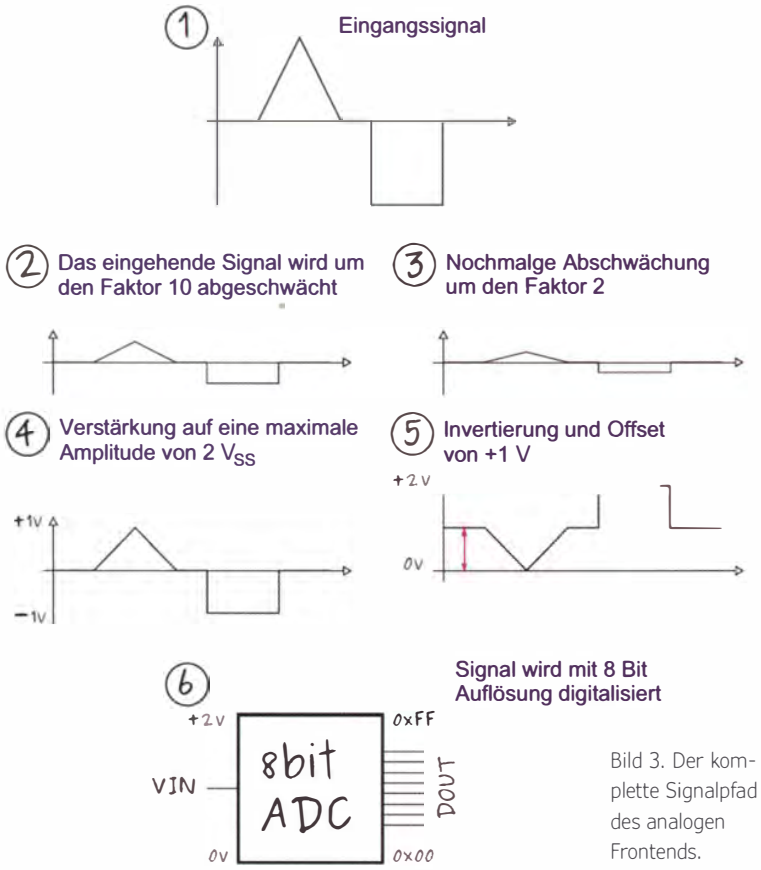


Bild 3. Der komplette Signalpfad des analogen Frontends.

nal entsprechend konditioniert werden, damit es optimal per ADC digitalisiert werden kann. Mit dem verwendeten unipolaren ADC können natürlich nur positive Signale erfasst werden. Allerdings will man mit einem Oszilloskop ja auch Wechselspannungen messen können. Aus diesem Grund muss in solchen Fällen der Pegel entsprechend angehoben werden. Doch bevor es zu sehr in die Details geht, empfiehlt sich ein Blick auf Bild 3, das den kompletten Signalpfad des analogen Frontends abbildet. Nachfolgend die Erläuterung , was auf diesem Bild zu sehen ist.

1. Das Originalsignal.
2. Das Oszilloskop ist für 10:1-Tastköpfe konzipiert, weshalb das eingehende Signal um den Faktor 10 abgeschwächt ist.
3. Vor der Verstärkerstufe wird es nochmal um den Faktor 2 abgeschwächt, damit höhere Signalamplituden gemessen werden können.
4. Bei der ersten Opamp-Stufe wird das Signal entweder unverstärkt wie bei einem Spannungsfolger weitergegeben oder eben noch durch eine nichtinvertierende Verstärkerstufe geführt. Da der nutzbare Eingangsspannungsbereich des ACD auf 2 V eingestellt ist, sollte die Amplitude des verstärkten Signals 2 V Spitze-Spitze-Spannung nicht überschreiten.
5. In der zweiten und letzten Opamp-Stufe wird das Signal invertiert und um 1 V angehoben., Die Invertierung hat keinen praktischen Nutzen. Sie muss später im digitalen Frontend wieder aufgehoben werden. Das ist sehr einfach, da man hierzu lediglich das Resultat des ADC negiert.
6. Das konditionierte Signal wird dann vom ADC mit 8-Bit-Auflösung digitalisiert. Da der Eingangsspannungsbereich des ADCs 2 V beträgt, werden alle Spannungen, die diesen Wert überschreiten, als 0xFF am Ausgang erscheinen. Entsprechend werden alle Spannungen unter 0 V den Wert 0x00 annehmen. Spannungen zwischen 0 und 2 V werden linear in den Wertebereich einer Unsigned-Byte-Variablen umgesetzt.

Nachdem nun klar ist, wie der analoge Teil vom Prinzip her operiert, steht die Übersetzung in reale Hardware an. Mit Elektronik-Erfahrung sollte die Schaltung weitgehend selbsterklärend sein. Um alle denkbaren Unklarheiten zu beseitigen, folgt noch eine kurze Erläuterung im nächsten Abschnitt, die dann noch von detaillierteren Betrachtungen ergänzt wird.

Im ersten Rechteck links in der LTSpice-Simulationsschaltung in Bild 4 sieht man das Mo-

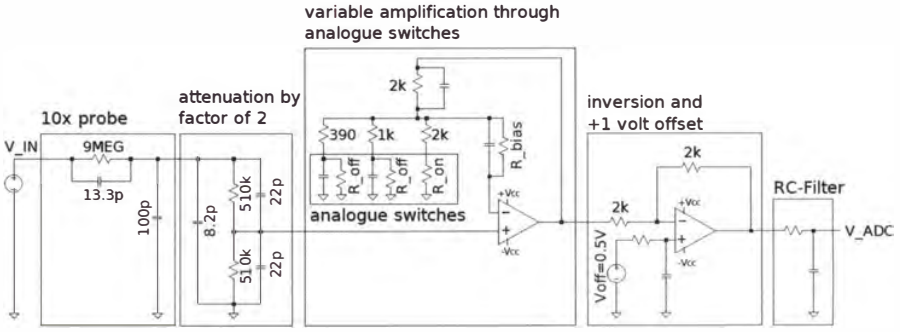


Bild 4. LTSpice-Simulationsschaltung der analogen Stufe.

dell eines 10:1-Tastkopfs. Anschließend wird das Signal nochmals mit einem ohmschen und kapazitiven Spannungsteiler durch den Faktor 2 geteilt. Die folgende Opamp-Stufe ist nichtinvertierend beschaltet. Die Analogschalter schalten entweder nach Masse oder sind offen. Mit ihnen wird die Verstärkung des Opamps eingestellt. Dann invertiert ein weiterer Opamp das Signal und hebt es um 1 V an. Zum Schluss kommt noch ein optionaler RC-Filter vor dem ADC-Eingang.

Tastköpfe

Wenn man elektrische Signale misst, sollte man darauf achten, dass man das zu messende Signal nicht durch die Messvorrichtung bzw. deren Last beeinträchtigt. Mit anderen Worten: Die Eingangsimpedanz des Messgeräts sollte hoch sein. Übliche 10:1-Tastköpfe für Oszilloskope haben einen 9-MΩ-Widerstand eingebaut. Zusammen mit den 1 MΩ des Oszilloskop-Eingangs ergibt das einen Gesamtwiderstand des Eingangs von 10 MΩ. Als erste Näherung passt ein gewöhnlicher Spannungsteiler als Modell ganz gut, denn der Widerstand des Koax-Kabels ist vernachlässigbar. Dafür liegt dann aber auch nur ein Zehntel der Eingangsamplitude am Oszilloskop-Eingang. Doch außer ohmschen Widerständen spielt auch noch die Kapazität des Koax-Kabels gegen Masse eine Rolle. Diese Kapazität schwankt von Tastkopf zu Tastkopf. Man kann grob von etwa 100 pF pro Meter Kabel ausgehen. Bei Gleichspannungen hat diese Kapazität keinen Effekt. Bei sich ändernden Spannungen hingegen ergäbe sich so ein Tiefpass für den Wechselspannungsanteil. Um diesen störenden Effekt zu kompensieren, befindet sich parallel zum 9-MΩ-Widerstand noch ein einstellbarer Kondensator.

Wie schon erwähnt, beträgt der Eingangswiderstand eines Oszilloskops 1 MΩ. Hinzu kommt noch eine definierte Eingangskapazität von 20 pF. Zusammen mit der Kabelkapazität muss man also mit rund 120 pF am Eingang des Oszilloskops rechnen. Damit diese Kapazität voll kompensiert wird, indem sich ein kapazitiver Spannungsteiler im Verhältnis 10:1 bildet, muss der einstellbare Kondensator im Tastkopf auf ein Neuntel der 120 pF (≈ 13,3 pF) abgeglichen werden. In der Praxis berechnet man natürlich nicht erst die nötige Kapazität im Tastkopf, sondern legt ein Rechtecksignal an und verstellt den Trimmkondensator mit einem kleinen Schraubendreher so lange, bis das Rechtecksignal die richtige Kurvenform hat.

Abschwächer

Ganz analog dazu ist die Abschwächung auf 50 % des Signals realisiert. Ein ohmscher und ein kapazitiver Spannungsteiler halbieren die Signalamplitude. Wichtig ist hier lediglich, dass die Eingangsimpedanz wie für ein Oszilloskop üblich 1 MΩ || 20 pF beträgt. Die beiden 510-kΩ-Widerstände am Eingang addieren sich genau genug zu 1 MΩ. Der Einfluss des folgenden Opamp-Eingangs ist vernachlässigbar, da er wie für Opamps typisch im Bereich mehrerer hundert MΩ liegt. Die Eingangskapazität ergibt sich durch die beiden in Serie geschalteten 22-pF-Kondensatoren, die parallel zum einzelnen 8,2-pF-Kondensator liegen, was rechnerisch auf 19,2 pF hinausläuft. Hinzu kommt noch die Eingangskapazität des folgenden Opamps von 0,5 pF und schon ist man auf dem richtigen Niveau. Nimmt man stattdessen aber einen Opamp mit höherer Eingangskapazität, kann man statt dem 8,2-pF-Kondensator kompensatorisch einen kleineren Wert einsetzen.

Variable Verstärkung

Ein nichtinvertierender Verstärker entspricht der folgenden Gleichung:

$$V_{out} = \left(1 + \frac{R1}{R2}\right) \times V_{in}$$

Wie ist das zu verstehen? Vereinfacht gesagt erzeugt ein gegengekoppelter Verstärker ein solches Ausgangssignal, dass sein positiver und sein negativer Eingang die gleiche Spannung aufweisen, denn deren Differenz wird ja bekanntlich (sehr hoch) verstärkt. Da Vin am positiven Eingang liegt, steuert der Opamp seinen Ausgang so, dass der negative dem positiven Eingang folgt. Das ist hier dann der Fall, wenn die Spannung am Abgriff zwischen den Widerstände R1 und R2 genau Vin entspricht. Da R1 und R2 einen Spannungsteiler ergeben, muss die Ausgangsspannung gegenüber dem positiven Eingang um genau die Abschwächung des Spannungsteilers verstärkt sein.

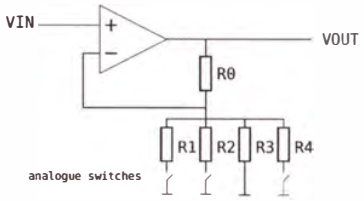


Bild 5. Nichtinvertierende Verstärkerstufe mit variabler Verstärkung.

Um unterschiedliche aber diskrete Verstärkungen zu realisieren, werden unterschiedliche Widerstände per Analogschalter parallelgeschaltet bzw. hier auf Masse gelegt (Bild 5). Ein idealer Analogschalter hat im geöffneten Zustand einen unendlich hohen Widerstand und exakt 0 Ω, wenn er geschlossen ist. Idealerweise hat er eine Kapazität von 0 F. Reale Analogschalter sind zwar

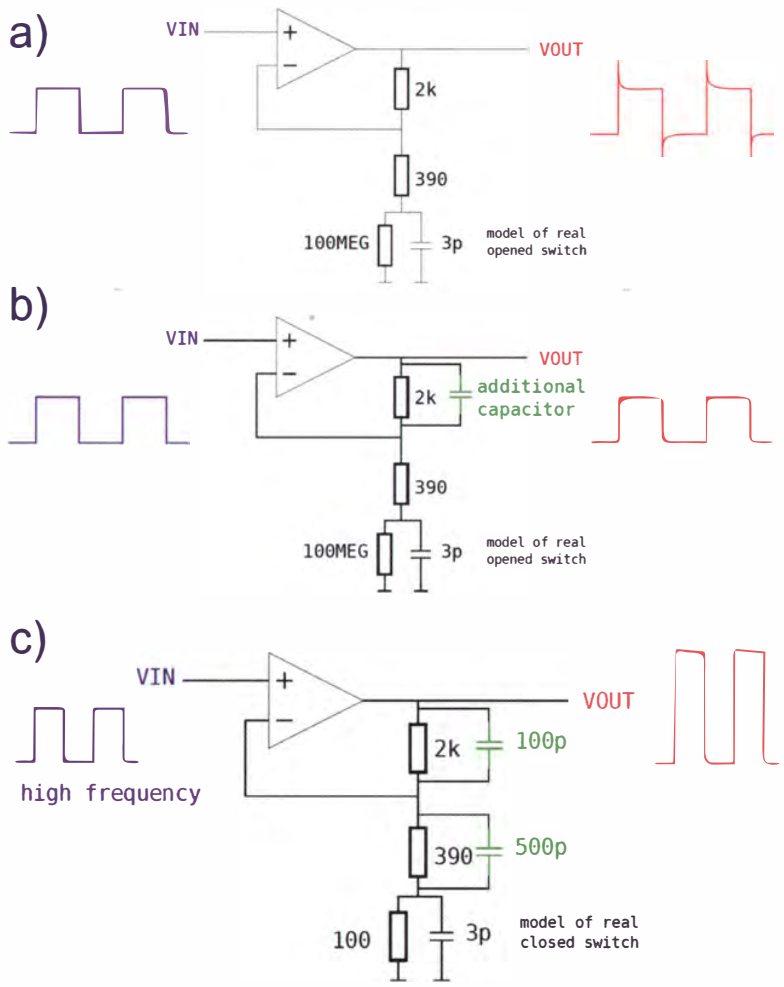


Bild 6. Ein einzelner Analogschalter (a). Um den Effekt der parasitären Kapazität zu verhindern, könnte man einen zusätzlichen Kondensator einbauen (b). Ein weiterer Kondensator könnte jeweils zum unteren Widerstand parallelgeschaltet werden, um die Verstärkung ohne Verzerrungen zu erhöhen (c).

sehr hochohmig, wenn sie geöffnet sind, doch im geschlossenen Zustand haben sie einen nicht zu vernachlässigenden Widerstand in der Größenordnung von 100 Ω. Auch die parasitären Kapazitäten muss man bei der Konstruktion eines analogen Frontend berücksichtigen.

Wie schon erwähnt, werden die Analogschalter in dieser Schaltung so eingesetzt, dass immer ein Anschluss auf Masse liegt. Der Vorteil dieser Anordnung ist, dass der Opamp als Spannungsfolger mit einer Verstärkung von 1 fungiert, wenn alle Analogschalter geöffnet sind. Mit jedem Analogschalter kann man durch sein Schließen dann eine definierte Verstärkung einstellen. Mit vier Schaltern ergeben sich mit den zugehörigen Widerständen dann fünf unterschiedliche Verstärkungen. Der Nachteil dieser Schaltungstechnik ist, dass man den realen Ron dieser Analogschalter berücksichtigen muss, da er mit in die Verstärkung eingeht. Hinzu kommen noch die Effekte der parasitären Kapazitäten auf das Signal, die besonders bei offenen Schaltern zum Tragen kommen.

Betrachtet man der Einfachheit halber nur einen einzigen Analogschalter (Bild 6a): Angenommen, er hat geöffnet einen Widerstand von 100 MΩ und eine parasitäre Kapazität von 3 pF. Zwar wird bei entsprechend geringer Frequenz des Signals dessen Amplitude nicht sehr beeinträchtigt, aber die Flanken des Rechtecks sind durch überschwingendes Verhalten verformt. Diese Verformungen sind das Ergebnis der parasitären Kapazität des Analogschalters im Gegenkopplungskreis des Opamps. Die Flanken des Signals enthalten ja hochfrequente Signalanteile. Diese werden durch den Tiefpasseffekt der parasitären Kapazität im Gegenkopplung abgeschwächt. Entsprechend werden sie dann vom Opamp übermäßig verstärkt, was dann diese Overshoot-Effekte ergibt.

Um den Effekt dieser parasitären Kapazität zu verhindern, könnte man wie in Bild 6b einen zusätzlich Kondensator einbauen. Ein Wert von 100 pF wird die Verformung deutlich reduzieren. Allerdings erkauft man sich das durch eine Verringerung der Bandbreite des analogen Frontends, auch mit aktiver Verstärkung wie in Bild 6c. Um diesen Effekt zu neutralisieren, könnte ein weiterer Kondensator jeweils zum unteren Widerstand parallelgeschaltet werden. Dies ist qualitativ in Bild 6c dargestellt. Aber als diese Maßnahmen real am Board durchgeführt werden sollten, brachten diese parallelen Kondensatoren keine rechte Wirkung. Hierzu muss gesagt werden, dass die Verformungen bei der einfachen Verstärkung am deutlichsten waren und mit der Verstärkung abnehmen. Es wurde deshalb versucht, einen Kompromiss zwischen geringer Verformung bei niedrigen Verstärkungen und ausreichender Bandbreite zu finden, indem ein zusätzlicher Kondensator parallel zum oberen Widerstand platziert wurde.

Invertierung und Anhebung um 1 V

Ein invertierend beschalteter Operationsverstärker wie in Bild 5.15 entspricht der folgenden Gleichung:

$$V_{out} = -\frac{R_2}{R_1} \times V_{in}$$

Wenn R1 = R2 ist entspricht das Ausgangssignal dem invertierten Eingangssignal. Die Anhebung des Signalpegels wird dadurch realisiert, dass an den positiven Eingang eine definierte Spannung wie in Bild 5.16 gelegt wird. Diese Schaltung verarbeitet also die Eingangsspannung Vin und die Offset-Spannung Voff. Das Funktionsprinzip der Überlagerung wird klar, wenn man zunächst davon ausgeht, dass Voff = 0 V ist. Nun bestimmt nur Vin die Ausgangsspannung Vout. In diesem Fall funktioniert die Schaltung exakt so wie die von Bild 5.15. Und daher gilt:

$$V_{out1} = -\frac{R_2}{R_1} \times V_{in} = -V_{in}$$

Nun gilt zur Abwechslung Vin = 0 V. Jetzt liefert nur noch Voff einen Beitrag zum Signal von Vout. Wenn Vin auf Masse gelegt wird (siehe Bild 5.17), entspricht die Schaltung einem nichtinvertierenden Verstärker. Von daher ist der Beitrag von Voff zu Vout:

$$V_{out2} = \left(1 + \frac{R2}{R1}\right) \times V_{off} = 2 \times V_{off}$$

Addiert man beide Beiträge zu Vout, ergibt sich:

$$V_{out} = V_{out1} + V_{out2} = -V_{in} + 2 \times V_{off}$$

Für eine Pegelanhebung um 1 V, muss Voff einen Wert von 0,5 V annehmen.

RC-FILTER

Das RC-Filter am Ende des analogen Frontends ist nicht absolut notwendig. Es fungiert als Tiefpass und unterdrückt Overshoot-Phänomene, wie sie im Abschnitt Variable Verstärkung beschrieben wurden. Den Kondensator kann man auch einfach weglassen. Der Widerstand ist aber notwendig. Er reduziert den Strom, der dann aufgrund der internen Schutzdioden fließt, wenn die Spannung am Eingang des ADC zu groß oder zu klein wird.

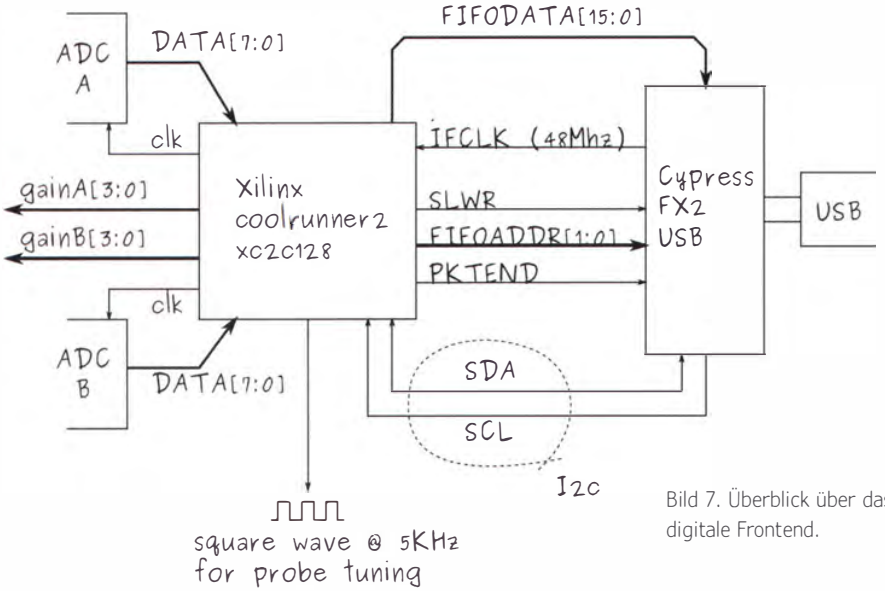


Bild 7. Überblick über das digitale Frontend.

Digitales Frontend

Das digitale Frontend fängt da an, wo das analoge aufhört: am ADC - genauer gesagt am Ausgang des A/D-Konverters. Es endet bei den USB-Transfers an das Host-Gerät. Bild 7 gibt einen Überblick über das digitale Frontend. Die beiden wichtigsten Komponenten sind ein CPLD vom Typ Coolrunner 2 (xc2c128) von Xilinx und ein USB-Controller vom Typ FX2 von Cypress. Die Samples werden an das CPLD weitergereicht und dann in das FIFO-Interface des FX2 eingereiht, damit sie letztlich via USB zum Host-Gerät gelangen. Der FX2-Mikrocontroller unterstützt drei unterschiedliche Betriebs-Modi: einen Ports-Mode, GPIF (General Purpose Interface Master) und Slave-FIFO. Für unsere Zwecke passt Slave-FIFO am besten. Wie die Bezeichnung nahe legt, braucht es hierfür einen externen Master, bei dem es sich um das CPLD dieser Anwendung handelt.

Der externe Master setzt die FIFOADDR-Pins, um einen bestimmten Buffer des FX2 zu adressieren. Wann immer nun der Master auf den SLWR-Pin aktiviert, werden die Daten an den FIFODATA-Pins in diesen adressierten Buffer geschrieben. Da das FIFO-Interface mit 48 MHz getaktet wird und FIFODATA 16 Bit breit ist, ergibt sich ein Datendurchsatz von immerhin 768 MBit/s. Damit nun keine Samples verloren gehen, muss man aufpassen, denn die theoretische Grenze von High-Speed-USB liegt bei nur 480 MBit/s. Deshalb gibt es zwei unterschiedliche Sampling-Modi: einen kontinuierlichen und einen Hardware-getriggerten Modus. Im Continuous mode wird der SLWR-Pin nur bei jedem achten Takt aktiviert, was einer effektiven Frequenz von 6 MHz entspricht. Die resultierende

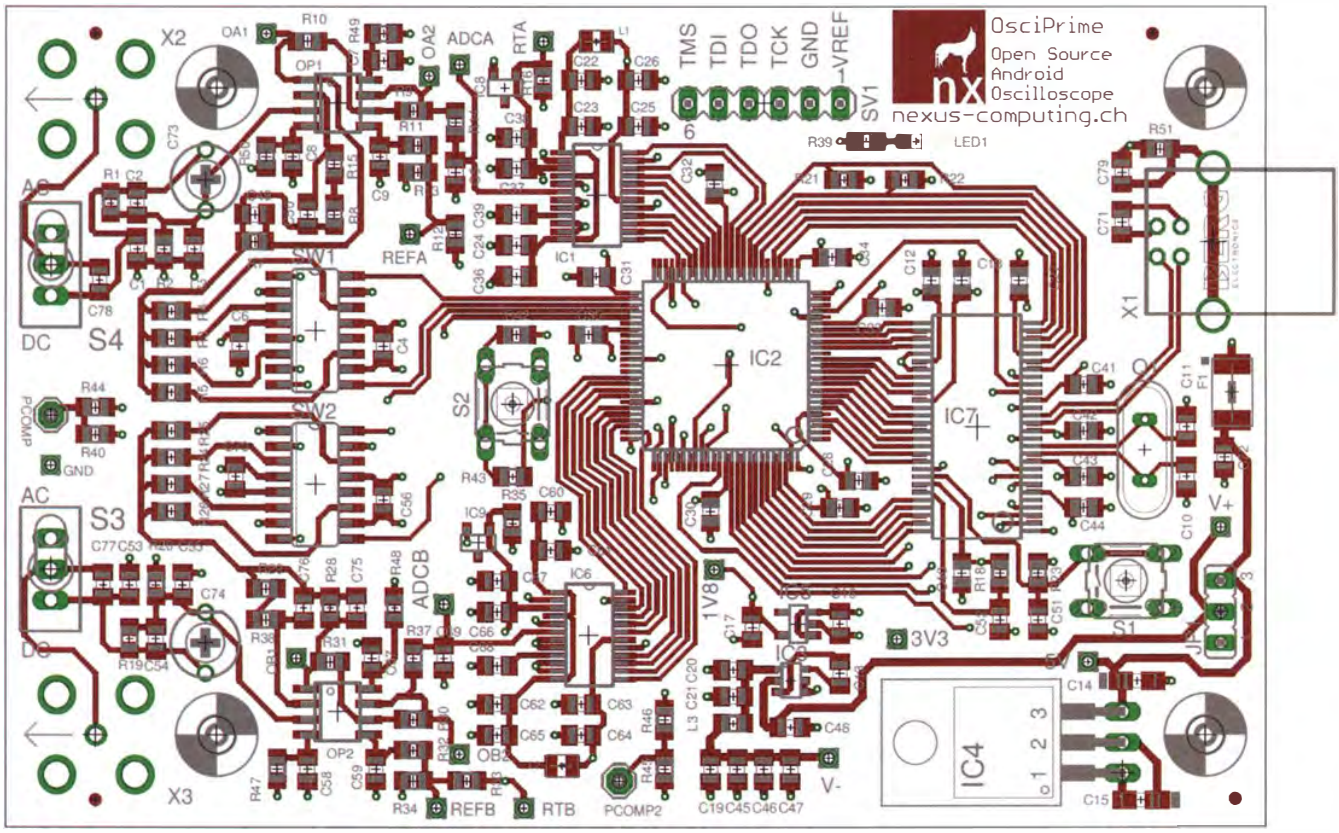


Bild 8. Die Platine des JellyScopes.

Datenrate liegt deshalb nur bei 96 MBit/s, was problemlos über USB übertragen werden kann. Im Hardware-getriggerten Modus wartet das CPLD, bis eine definierte ansteigende oder fallende Flanke im Strom des Samples auftritt. Erst dann wird der SLWR-Pin aktiviert und auch das nur für 1.024 Takte. Dabei werden folglich genau 2.048 Byte in die FIFO-Queue übertragen.

Die Daten werden vom FX2 im internen Speicher gepuffert und dann ohne Gefahr eines Datenverlustes über USB an den Host übertragen. Alle relevanten CPLD-Parameter wie der Modus (kontinuierlich oder getriggert), ansteigende oder fallende Flanke, Trigger-Pegel etc. werden vom FX2 via I²C an das CPLD übertragen.

CPLD-Firmware

Die CPLD-Firmware wurde in VHDL mit der freien ISEWeb-Pack-Software von Xilinx erstellt. Sie besteht aus drei unterschiedlichen Modulen: I²C-Slave, Controller und Sampling. Das I²C-Slave-Modul empfängt Daten und informiert den Controller, wenn ein neues Byte verfügbar ist. Diese Daten werden vom Controller eingelesen. Der Controller steuert auch z.B. die Verstärkung des analogen Frontend und zusätzlich interne Parameter des Sampling-Moduls.

Das Sampling-Modul besteht aus dem Teil mit dem kontinuierlichen Sample-Modus und dem Teil mit dem Hardware-getriggerten Modus mit Flankendetektion. Im kontinuierlichen Modus wird der Endpoint 2 des FX2 adressiert. Im Hardware-getriggerten Modus wird Endpoint 6 genutzt. Am Anfang des Hardware-getriggerten Modus wird auf das PKTEND-Signal zugegriffen. Dies wiederum erzeugt ein kurzes Paket um sicher zu stellen, dass die Buffer des FX2 geleert sind, bevor 2.048 Byte in die Queue geschrieben werden.

Ein Datenflussdiagramm des internen VHDL-Moduls des CPLD und die entprechenden Signale liegen zusammengefasst unter http://www.elektor.de/mc7/Download zum Download bereit. Der VHDL-Code wird zwar nicht detailliert erläutert, doch zusammen mit dem Flussdiagramm der Daten, dem Daten-Lexikon und dem Source-Code der Module Controller und Sampling sollte er nachvollziehbar sein. Das I²C-Slave-Modul ist wohl nicht

so leicht verständlich. Es empfiehlt sich, hierfür auch die I²C-Specification von NXP zu Rate zu ziehen. Das I²C-Slave-Modul empfängt lediglich Daten und bestätigt sie, schickt sie aber nicht zurück zum I²C-Master.

Um die internen Parameter des CPLD zu setzen, muss ein empfangenes Byte ein spezielles Muster aufweisen. Die ersten zwei Bits (die signifikantesten Bits) entscheiden darüber, welche Parameter durch die restliche 6 Bit an Nutzdaten geändert werden sollen. Eine Zusammenfassung dieser Muster kann unter http://www.elektor.de/mc7/download abgerufen werden.

Vielleicht fragen Sie sich, warum das Signal probeTuning vom Modul Sampling statt vom Controller oder einem anderen Modul stammt? Um das benötigte Rechtecksignal mit einer Frequenz von 5 kHz zu generieren, ist ein relativ großer interner Counter erforderlich. Da das Modul Sampling im Hardware-getriggerten Modus exakt 2.048 Byte schreibt und dies einen Counter implementiert, kann der gleiche Counter auch für die Erzeugung des Rechtecksignals genutzt werden. Andernfalls würden dem CPLD die Makrozellen ausgehen und man kann es dann nicht erfolgreich synthetisieren.

Der VHDL-Source-Code liegt gemeinsam mit den Source-Dateien dieses Beitrags unter der oben angeführten Adresse bereit. Es sind auch Testbenches im VHDL-Source-Code enthalten. Beim CPLD werden 112 von 128 Makrozellen genutzt, davon allein 95 Makrozellen durch das Design. Das CPLD wird via JTAG programmiert. Wir haben hierfür das Platform Cable USB für eine USB-JTAG-Verbindung benutzt. Unglücklicherweise ist dieses Tool ziemlich teuer. Es gibt aber auch preiswertere Alternativen wie das XUP USB-JTAG Programming Cable.

FX2-Firmware

Die Aufgabe des Mikrocontrollers FX2 besteht darin, auf Anfrage die vom CPLD erhaltenen Daten an den Android-Host weiterzuleiten. Der FX2 ist auch dafür zuständig, die Befehle vom Host über I²C an das CPLD zu übertragen. Der Chip enthält nützliche Funktionseinheiten, die sich um die Datenübertragung kümmern (die serielle Hardware beispielsweise und einen 8051-Mikrocontroller, auf dem die Firmware läuft). Durch seinen Slave-FIFO-Mode ist ein FX2 in der Lage, die Daten von 16 Eingangs-Pins parallel einzulesen und direkt in den gewünschten Ziel-Buffer zu schreiben, sobald der entsprechende Write-Pin durch externe Hardware aktiviert wird. In diesem Modus wird die Firmware des 8051-Mikrocontrollers inaktiv und er muss also nicht ins Geschehen eingreifen. Um den Slave-FIFO-Mode zu nutzen, muss die Firmware lediglich das Interface entsprechend konfigurieren und die Endpoint-FIFOs bei der Initialisierung festlegen.

Auch der Source Code für diesen Teil des JellyScopes liegt zum Download bereit.

Layout

Für das Platinen-Layout kam Eagle zum Einsatz. Es wurde sehr viel Mühe darauf verwendet, eine möglichst kleine Platine zu kreieren, so dass dazu sogar die freie Eagle-Light-Version ausreicht.

Man muss große Sorgfalt bei der Trennung des analogen vom digitalen Teil walten lassen – es gibt nämlich keine getrennten Masseführungen für den analogen und digitalen Teil. Andere Verfahren, mit denen man Einstreuungen digitaler Signale in die analogen Stufen verhindern kann, ist die hier verwendete einfachste Maßnahme der räumlichen Trennung der beiden Teile. Außerdem dient der komplette Bottom-Layer als Massefläche für analoge und digitale Schaltungsteile.

Bild 8 zeigt die Eagle-Platine. Weitere Details findet man in der Schaltung und sonstigen Layout-Dateien, die sich im Datei-Verzeichnis zu diesem Projekt befinden. Die Eagle-Library oscione.lbr beispielsweise enthält alle relevanten Bauteile in Symbol- und Bestückungsform.

Android-Applikation

Die Aufgabe der Android-Applikation ist es, die Daten auf dem Bildschirm anzuzeigen. Darüber hinaus kann der Anwender die Applikation ganz ähnlich wie ein Hardware-Oszilloskop bedienen. Die Applikation steuert die Datenquellen wie die USB- oder Audio-Daten. Von daher verfügt sie über einen Prozess, der UI-Requests und das Grafik-Display behandelt, und zusätzlich über einen Dienst, der die Datenquellen und die Flusskontrolle steuert.

Um Daten anzeigen zu können, muss natürlich eine Datenquelle vorhanden sein. Das Frontend zapft eine Signalquelle an und übergibt die resultierenden Samples via USB an das Android-Gerät, dient also selbst wiederum als Signalquelle. Auch wenn wir andere Entwickler dazu ermutigen wollen, auch weitere Datenquellen zu implementieren, geht es hier (wenn nicht explizit erwähnt) vor allen Dingen um USB-Daten. Diese stellen die zeitkritischsten Daten in diesem Projekt dar. Sie müssen mit größerer Aufmerksamkeit als andere Datenquellen bedacht werden. Bild 9 zeigt die Schritte, die für die Anzeige von Samples im kontinuierlichen Modus notwendig sind. Da die Applikation insgesamt als Oszilloskop arbeitet, ist die Anzahl an darstellbaren Datenpunkten begrenzt. Man muss also eine vernünftige Auswahl für die Anzeige treffen.

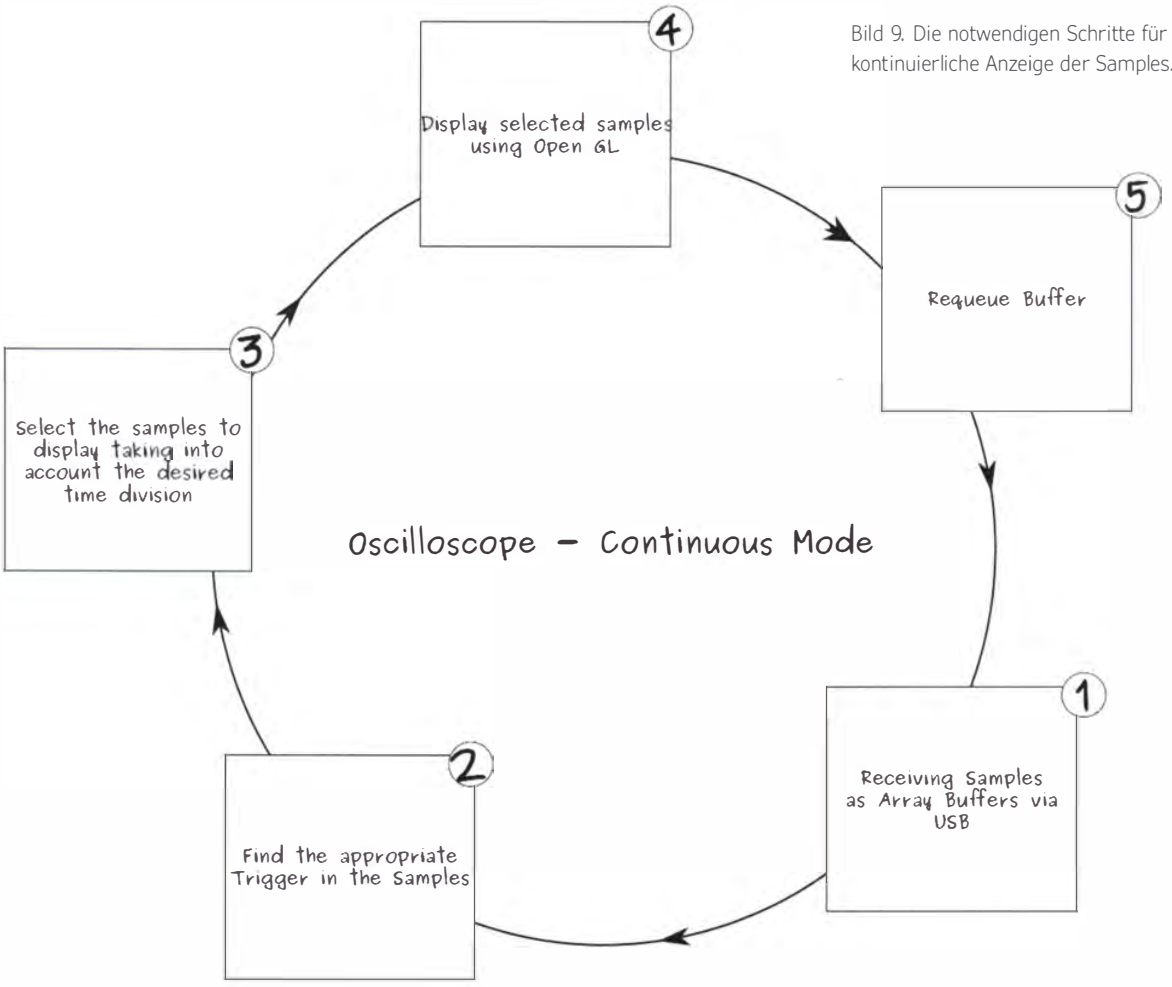


Bild 9. Die notwendigen Schritte für die kontinuierliche Anzeige der Samples.

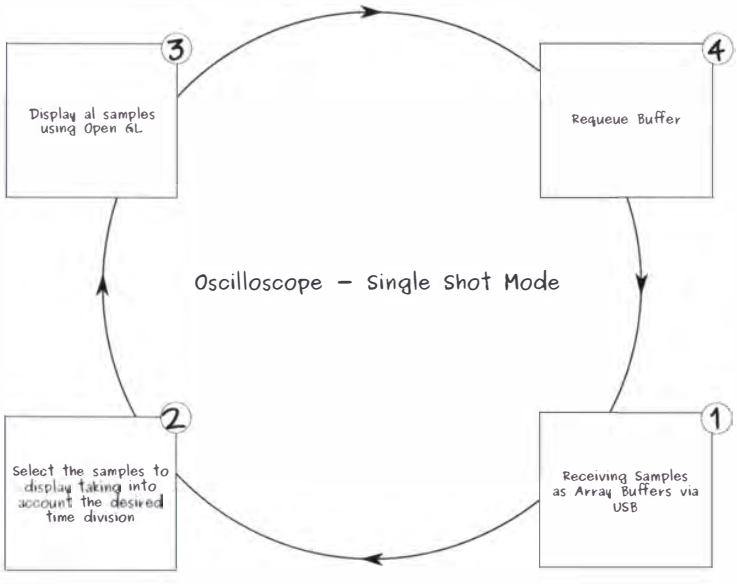


Bild 10. Die manuelle Triggerung entfällt: Den optimalen Trigger-Punkt findet die CPLD-Hardware selbstständig.

Diese Auswahl besteht aus einem Trigger, der durch die Detektion einer ansteigenden oder fallenden Signalflanke ausgelöst wird und einem Zeitbereich um den Trigger-Punkt, in dem die Samples angezeigt werden. Im Single-Shot-Mode übernimmt die Hardware die Triggerung und schickt nur die relevanten Samples an die Applikation. Software-Triggerung ist in diesem Fall also nicht notwendig.

Wie in Bild 9 gezeigt besteht die Applikation aus einem zyklischen Prozess. Das Hardware-Front-End schickt gesampelte Daten. Die Applikation kümmert sich um die USB-Transfers. Die Transfer-Strukturen beinhalten Daten-Buffer, die von den vom

Frontend erfassten Daten belegt werden. Nach der Datenerfassung werden die Daten von der Host-Applikation verarbeitet, so dass die gewünschten Samples ausgewählt und angezeigt werden können. Die Verarbeitung (Schritte 2 und 3) besteht aus dem Finden des Triggers und der Auswahl der Daten um den Trigger-Zeitpunkt herum, um diese dann anzuzeigen. Am Ende dieses Prozesses muss der initiale USB-Transfer wieder angereiht werden, so dass der nächste Buffer mit Sample-Daten transferiert werden kann. Wie bei Schritt 4 zu sehen, nutzt die Applikation OpenGL, um die ausgewählten Samples auf dem Bildschirm darzustellen. OpenGL sorgt für schnelles Rendering, was eine ausreichend Rate an Frame-Redraws erlaubt.
Der zyklische Prozess beinhaltet zwei wichtige Herausforderungen:

- **Timing:** Die mittlere Zeit zwischen den Punkten 1 bis 5 darf nicht länger als die Zeit zwischen zwei Transfers sein. Lösung: Operationen mit Zeitbedarf werden in nativem C geschrieben.
- **Flusskontrolle:** Bei einem einzigen Daten-Buffer gehen die in der Zeit von Schritt 1 bis 5 gesammelten Daten verloren, da keine USB-Pakete für neue Daten verfügbar sind. Aus diesem Grund ist eine Art von Muli-Buffering obligatorisch. Lösung: Verwendung mehrfacher Transfer-Strukturen und Daten-Buffer.

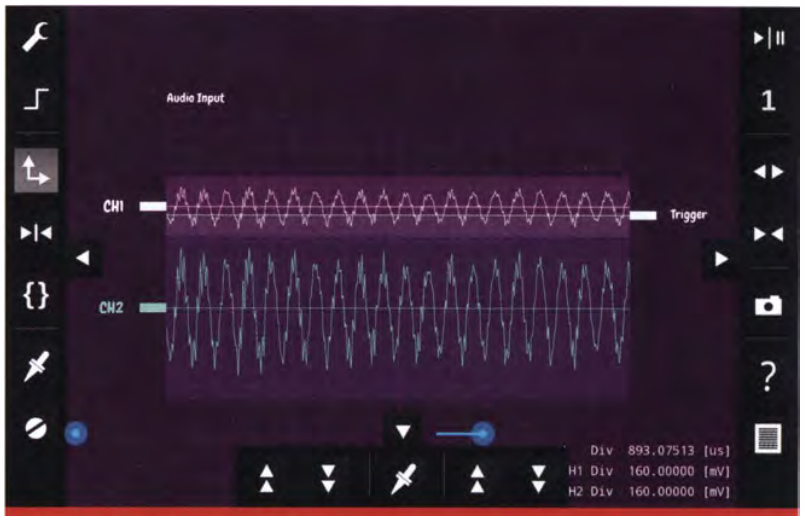
Die obere Grenze der Datenauflösung ist lediglich durch das Hardware-Frontend limitiert. Die untere Grenze ist hingegen durch die Applikation festgelegt. Mit High-Speed-USB-Bulk-Transfers liegt die praktische Grenze bei 40 MB/s. Im Single-Shot-Mode zeigt die Applikation nur eine Auswahl an Samples an, die nach einem Trigger-Ereignis erfasst wurden. Das Front-End entdeckt solche Trigger-Ereignisse und schickt die auf ein solches Ereignis folgenden Daten an die Android-Applikation.

Wenn man Bild 9 mit Bild 10 vergleicht, kann man sehen, dass keine Notwendigkeit zum Suchen von Trigger-Punkten im Signal besteht. Der Trigger-Punkt wird durch Hardware mit dem CPLD detektiert. Der Hauptgrund für einen Hardware-Trigger liegt in der erzielbaren hohen Sample-Rate von bis zu 48 MS/s pro Kanal (für eine kurze Zeit). Es ist leider nicht möglich, über längere Zeit eine Datenrate von 2 x 48 MS/s = 96 MB/s über USB aufrecht zu erhalten. Mit den Endpoint-Buffern des USB-Mikrocontrollers kann solch eine hohe Datenrate nur über kurze Zeit erreicht werden. Bei Single-Shot-Mode gibt es noch eine weitere heftige Herausforderung zu meistern: Pre-Trigger. Da beim CPLD kein RAM eingesetzt wird, ist ein Pre-Trigger alles andere als trivial.

Die Samples sind in einem FX2-Endpoint-Buffer abgelegt. Da diese Buffer nicht als Ringpuffer organisiert werden können, kann man auch keine konstante Anzahl an Pre-Trigger-Samples erreichen. Es ist allerdings möglich, Samples kontinuierlich an einen anderen Endpoint zu übertragen und sie zu verwerfen, wenn der Endpoint-Buffer voll ist, sowie wieder neue Samples in den Buffer zu schreiben. Das führt zu einem Pre-Trigger mit undefiniertem Umfang.

Und alles Open Source

Zum Schluss sei noch einmal darauf hingewiesen, dass alle Design-Unterlagen, weitere tiefgehendere Erläuterungen sowie Hinweise auf Zusatzinformationen zu diesem Projekt über http://www.elektor.de/mc7/download zu haben sind.



JellyScope im Playstore

Wie in der Anleitung erwähnt, ist der Markenname dieses Projekts OsciPrime. Mit diesem Wort als Suchanfrage gibt Googles Playstore die beiden für dieses Projekt verfügbaren Android-Anwendungen aus. Die kostenfreie Version ist auch eine Vollversion, das heißt, sie läuft mit dem hier vorgestellten Frontend und visualisiert das Signal des Mikrofoneingangs - beides allerdings nur auf einem Kanal und nur mit Grundfunktionen. Die vollausgestattete Version bietet den Funktionsumfang, den man von modernen Scopes her kennt und kostet weniger als 5 Euro.