Eurorack Computer

Ein Eurorackmodul mit python™-Anbindung
Bachelorarbeit aus Musikinformatik

Fabian Pfoser
Mat. Nr.: 1131376

2016/2017

Institut für Elektronische Musik und Akustik

Betreuung: Ao.Univ.Prof. Dipl.-Ing. **Winfried Ritsch**

Inhaltsverzeichnis

Die eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Abstract

Einen Computer in ein analoges Modularsystem zu integrieren mag zuerst paradox erscheinen, da die Verwendung von Analogsynthesizern meist im Kontrast zu Softwaresynthesizern steht und mit der Entscheidung der Verwendung von Analogsynthesizer bewusst nicht digital zu arbeiten oft einhergeht. Durch die Erfahrungen im Umgang mit Modularsystemen kam jedoch der Wunsch nach komplexeren Steuermöglichkeiten auf, welche nicht primär für den Analogsound zuständig sind, sondern das Verwalten und die Spielbarkeit komplexer Steuerungen ermöglichen sollte. Zusätzlich soll ein Audiointerface zur Wiedergabe und Aufnahme von gespielten dienen, um damit den Analogsynthesizer mit komplexen digitalen Audioprozessen zu erweitern. Als Linux-Minicomputer wurde ein “Olinuxino A20 Lime2“ der Firma Olimex mit Dual-Core ARM-Prozessor verwendet. Dieser ist für die Signalverarbeitung zuständig: Analog-Digital- und Digital-Analog-Wandlung von Steuersignalen und die Verarbeitung von Audiosignalen. Durch die Verwendung dieses vollwertigen Rechners mit vielerlei Interfaces, können die verschiedensten Anwendungen realisiert werden.
Es wurde ein Eurorackmodul entwickelt, welches aus einer Platine mit Userinterface, einem damit verbundenen Minicomputer und einer Frontplatte an der alles befestigt ist, besteht. Dieses Modul beinhaltet 10 Potentiometer, 12 Buchsen (6 Eingänge, 6 Ausgänge), 4 Kippschalter, 8 Silikondruckknöpfe mit RGB-Beleuchtung und 3 LEDs. Die Platine beherbergt noch 3 Analog-Digital-Wandler, 1 Digital Analogwandler und wurde noch mit einigen Operationsverstärkern ausgestattet, um die Signale der Eurorack-Norm anzupassen. Die Verarbeitung der Signale wurde mit Python realisiert, da diese Programmiersprache weit verbreitet ist und mit verhältnismäßig wenig Aufwand zu gewünschten Ergebnissen führt.

1 Einleitung

1.1 Übersicht über diese Arbeit

Zuerst wurde eine Recherche von existieren Modulen mit digitaler Anbindung gemacht, welche selbst programmierbar sind, um danach die Anforderungen des Moduls zu definieren. Im weiteren wurde die Hardware entwickelt, welche den Anforderungen für ein vielseitig verwendbares Eurorackmodul gerecht wird. Da jeder Eingang und Ausgang zwei Mal vorhanden ist, wurde bei der Anordnung der Bedienelemente auf einen symmetrischen Aufbau gesetzt. Somit können einfachere Anwendungen, wie zum Beispiel in Kapitel 6.1.1, doppelt ausgeführt werden ohne dass die Anwendung unübersichtlich wird.

Da kein Display integriert wurde, sind Silikon-Druckknöpfe mit RGB-LED verwendet worden. Durch verschiedene Farben kann so für den Benutzer ein konstruktives Feedback der Geschehnisse programmiert werden. Die Potentiometer wurden so angeordnet, dass sich die äußeren von den zwei mitlleren Reglern optisch unterscheiden, somit können zum Beispiel diese für eine Menüführung oder ähnliches verwendet werden, dies kann aber natürlich der Benutzer frei wählen. Im weiteren wurden alle Eingänge links und alle Ausgänge auf der rechten Seite platziert damit dies Ebenfalls schnell nachvollziehbar ist.

1.2 Einführung Eurorack

Der Eurorack-Standard wurde 1995 von der Firma DOEPFER-Musikelektronik eingeführt. Modulare Synthesizersysteme waren sehr große und teuere Instrumente welche kaum in einem privaten Studio platz fanden, doch durch den Eurorack Standard wurden die Systeme kleiner und auch leistbar. Im Eurorack-Standard passen alle Module in entsprechende 3HE (Höheneinheit, englisch: U - Unit) [1] Rahmen (Siehe 19-Zoll Norm: [1] bzw. das mechanische Konzept von Doepfer [2]) und haben rückseitig die gleiche Versorgungsspannung mit 12V, GND, -12V und optional 5V. Module verschiedenster Hersteller finden gemeinsam in einem Rahmen Platz. Die Steuerspannungen (Control Voltage - CV) betragen i.d.R. 0 bis 8V bzw 0 bis 10V. In den letzten Jahren sind immer mehr Firmen in das Eurorack-Geschäft eingestiegen und dadurch wurde fast ein ”Hype” ausgelöst.

1.3 Recherche digitaler, programmierbarer Eurorack Module

Anfangs war die Auswahl nur auf Analoge Module beschränkt, doch nach und nach wurden auch immer mehr digitale Module entwickelt. Es entstand eine Do-It-Yourself-Szene rund um modulare Synthesizer, welche selbst-gebaute Eurorack-Einschübe veröffentlichte, darunter gibt es auch selbst programmierbare Module. Beispielhaft werden folgende Module vorgestellt:
Eines dieser Module, welches voll funktionsfähig erworben werden kann, ist der OWL Modular, von Rebel Technology. Dieses Modul arbeitet mit einem ARM Cortex M4 Prozeessor. Da hier ebenfalls alles mit einer Open Source-Lizenz veröffentlicht wurde, gibt es schon sehr viele Anwendungen dafür, die frei zur Verfügung stehen.

[1] Eine Höheneinheit bei Elektronikbaugruppen entspricht 1,75 Zoll, also 44,45mm

Die anderen Module sind reine Do-It-Yourself-Projekte, wobei zwei davon als Basis einen Arduino [3] verwenden. Arduino sind meist Mikroprozessoren, welche eine Firmware benötigen und kein Computer-Betriebssystem. Dabei handelt es sich um Euro-Duino von Circuit Abbey und Ardcore von SnazzyFX. Diese beiden haben aber keine Audioschnittstelle und können somit nur für Logik-Anwendungen oder Steuerung verwendet werden.
Ein weiteres Open-Source-Projekt basiert auf einem Raspery Pi [4], welcher vermutlich der meist verwendetste Minicomputer ist. Das Projekt heißt Terminal Tedium, ein Breakoutboard mit Audiocodec für das Euroracksystem.

1.4 Open Source

Open Source ist ein internationaler Begriff für öffentlich zugängliche Projekte. Grundprinzip ist, dass alle Daten eines Projektes frei zur Verfügung stehen und jeder damit bzw. daran arbeiten kann. (Nähere Informationen über Open-Source-Hardware findet man unter folgendem Link: [5]
Somit können zum Beispiel Entwicklungskosten gespart oder ein Projekt schneller verbreitet werden. Einige Hersteller von Modulen kommen aus dem DIY-Bereich, bzw. kann durch eine Open Source-Community die professionelle Fertigung eines Projektes ermöglicht werden.
Da diese Arbeit sowohl Hardware- als auch Softwareseitig erweitert werden kann ist dies von Vorteil.

1.4.1 Projektverwaltung

Bei diesem Projekt wird die Hardware in einen Repository, ein verwaltetes Online-Verzeichnis zur Speicherung und Beschreibung von digitalen Objekten, dokumentiert und der Quellcode der Software als Open Source dort zur Verfügung gestellt. Änderungen werden von zugehörigen Versionssystem, in diesen Fall Git, aufgezeichnet und Archiviert.
Die Struktur des Repository von dieser Arbeit ist in zwei Bereiche aufgeteilt:

- Hardware
  - Beinhaltet die Stückliste, den Schaltplan und das Layout der Platine.
- Software
  - Beinhaltet das Pythonmodul mit den in Kapitel 3.4 beschriebenen Skripten, Testprogramme und den realisierten Anwendungen.

Dieses öffentliche Verzeichnis ist unter folgender Adresse zu finden:
`https://git.iem.at/mi/Eurorack_Software_Interface`

2 Entwicklung der Hardware

2.1 Bedienelemente

Bei analogen Synthesizern gibt es Steuersignale und Audiosignale. Damit verschiedenste Anwendungen realisiert werden können, wurde der Eurorack Computer mit jeweils zwei "gate" Ein- und Ausgängen, welche unter anderem als Trigger oder "Note-on/off" zum Beispiel in Sequenceranwednungen verwendet werden können, ausgestattet. Weiters wurden jeweils zur Spannungssteuerung zwei CV- Ein- und Ausgänge und als Audiointerface zwei Audio Ein- und Ausgänge implementiert. Als Bedienelemente des Moduls wurden zehn Potentiometer, vier Kippschalter und, als Taster und Schalter verwendbar, 8 Silikon-Druckknöpfe verwendet, welche je nach Anwendung frei programmierbar sind. Für die Rückmeldungen im Schnittstellen-Design können, zusätzlich zu den drei einfarbigen LEDs in der Mitte des Moduls, die in den Silikon-Druckknöpfen integrierten RGB-Leds verwendet werden.

Bei der Auswahl der Bedienelemente wurde auf gute Qualität Wert gelegt, dazu wurden Potentiometer von der Firma Alpha [6] und Kippschalter von der Firma Salecom [7] verwendet. Die verwendeten Buchsen von der Firma Thonk [8] wurden speziell für den Eurorack-DIY Bereich entwickelt. Viele namhafte Hersteller verwenden diese Buchsen da sie qualitativ sehr gut sind. Die Silikon-Druckknöpfe werden über Sparkfun [9] vertrieben.

2.2 Erstellung des Schaltplans

Um den Eurorack-Standard gerecht zu werden, ist es notwendig die Ausgangssignale an den gewünschten Pegel anzupassen. Ebenso ist es wichtig, die Eingangssignale so zu beschränken, dass der Olinuxino nicht beschädigt oder gar zerstört wird.

Die Schaltung wurde mit der Open Source Software KiCad [10] gezeichnet. Der Schaltplan wurde auf zwei Seiten aufgeteilt, wobei die zweite Seite die Ein- und Ausgänge und die Potentiometer beinhaltet. Für die Verschaltung der Analog-Digital- und Digital-Analog-Wandler wurden die jeweiligen Referenzdesigns aus den Datenblättern und Anwendungsmanuals des Herstellers verwendet und auf diese Anwendung angepasst. Als Vorlage für die Schaltungen der Ein- und Ausgänge wurden jene der Firma Mutable Instruments [11] verwendet. Dies ist möglich, da sie als Unterstützer der Open Source Community, jedes ihrer Module unter anderem mit einer Open-Software bzw. Open Hardware Lizenz veröffentlicht und damit alle Quellcodes und Schaltungen für weitere Entwicklungen frei zur Verfügung stehen.

2.2.1 Versorgung

Um dem Eurorack-Standard gerecht zu werden, wurde ein 16-Poligen Anschluss verbaut. Über ein Flachbandkabel wird das Modul dann mit dem Versorgungsbus eines beliebigen Eurorack-Systems verbunden. Die meisten Netzteile der Hersteller von Gehäusen haben zusätzlich eine 5V Versorgung. Damit kann der Benutzer wählen mit welcher Versorgung das Modul gespeist wird. Weiters wurde noch eine 2,1mm-Buchse verbaut, sodass der Olinuxino direkt von der 5V Versorgung des Bussystems versorgt werden kann. Dabei ist zu beachten, dass diese Versorgung auch genug

Strom liefern kann. Zur Sicherheit sind noch zwei Dioden als Verpolungsschutz hinzugefügt worden, welche für erfahrene Anwender nicht zwingend notwendig sind.

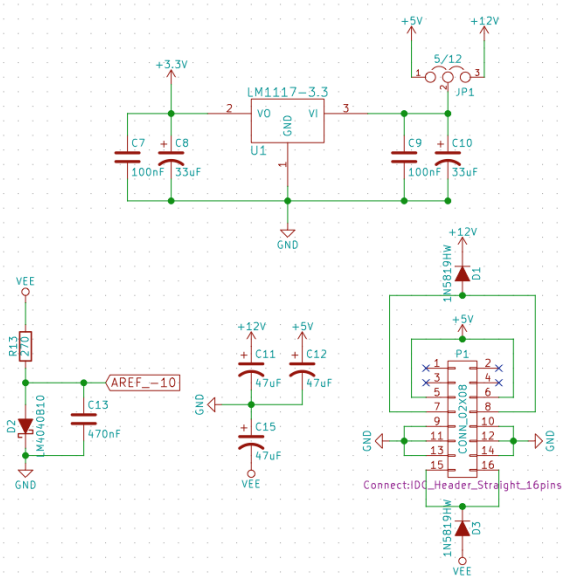


Abbildung 1: *Schaltung der Versorgung*

2.2.2 Gate Eingang

Dieser Eingang schaltet mit Hilfe von der für das Eurorack vorgesehenen Eingangsspannung von 0V - 12V einen Transistor, welcher dann 3,3V an den Olinuxino weitergibt. Wichtig war hier dass die 3,3V vom Entwicklungsboard und nicht die der Versorgung der eigenen Schaltung verwendet wurde, da diese Spannungen sich gering unterscheiden und somit Probleme bei den GPIOs auftreten können.

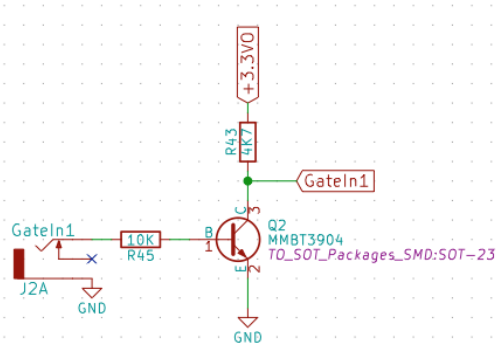


Abbildung 2: *Schaltung eines Gate Eingangs*

2.2.3 ControlVoltage Eingang

Dieser Eingang wandelt Eingangsspannungen von -5V - 12V in 3,3V - 0V um, wobei es keinen Schaden verursacht wenn -12V anliegen. Ein höherer Pegel sollte auf keinen Fall angelegt werden, was aber ohnehin in einem Euroracksystem nicht möglich ist.

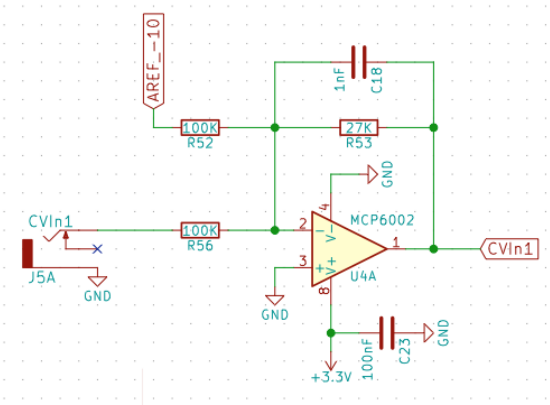


Abbildung 3: *Schaltung eines CV Eingangs*

2.2.4 Audio Eingang

Hier wurde als Verstärkerschaltung ein Operationsverstärker als invertierender Verstärker verwendet. Vor der Verstärkerschaltung wurde eine Ferrit (Spule) geschaltet damit möglichst wenig hochfrequente Störsignale in das Modul gespeist werden.

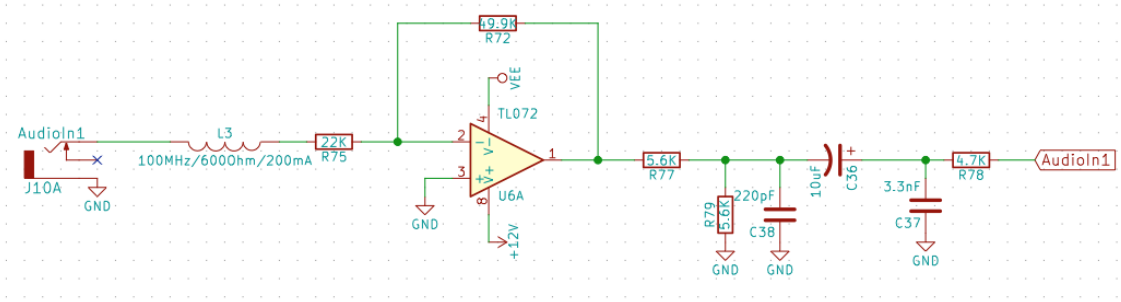


Abbildung 4: *Schaltung eines Audio Eingangs*

2.2.5 Gate Ausgang

Bei diesem Ausgang wurde analog zum Eingang, ebenso ein Transistor verwendet, der beim anliegen einer 3,3V Spannung ein 5V Signal an die Ausgangsbuchse schaltet.

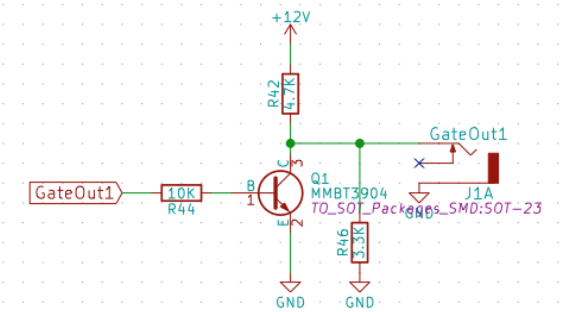


Abbildung 5: *Schaltung eines Gate Ausgangs*

2.2.6 ControlVoltage Ausgang

Dieser Ausgang wurde ebenfalls mit einer Operationsverstärkerschaltung realisiert welche die Ausgangsspannung des Digital-Analog-Wandlers von 0V - 3,3V auf 0V - 8V verstärkt.

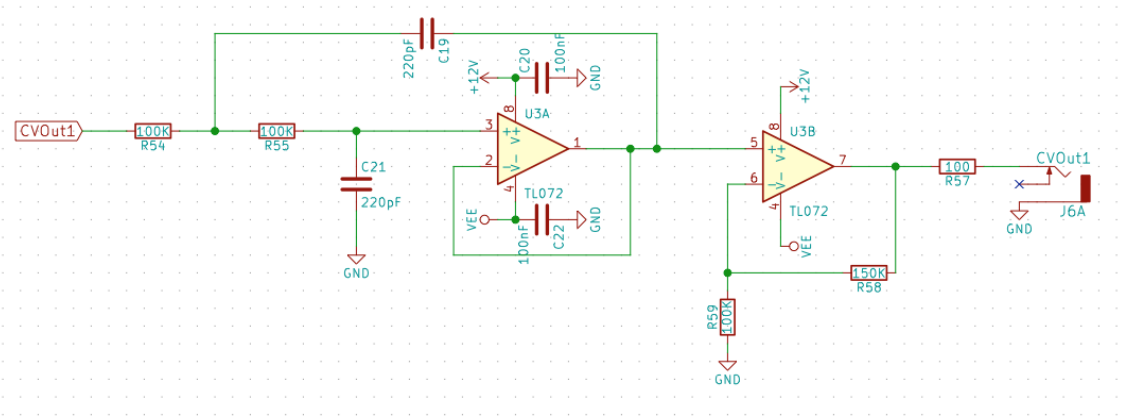


Abbildung 6: *Schaltung eines CV Ausgangs*

2.2.7 Audio Ausgang

Hier wurde der Kopfhörerausgang des Olimex-Boards als Audioausgang gewählt. Da hier eine eigene Audio-Masse vorhanden ist, sind die Ausgänge des Moduls auch von der gemeinsamen Masse getrennt. Auch hier wurden vor die Verstärkerschaltung Ferrite geschaltet damit keine Hochfrequenten Störsignale in das System übertragen werden.

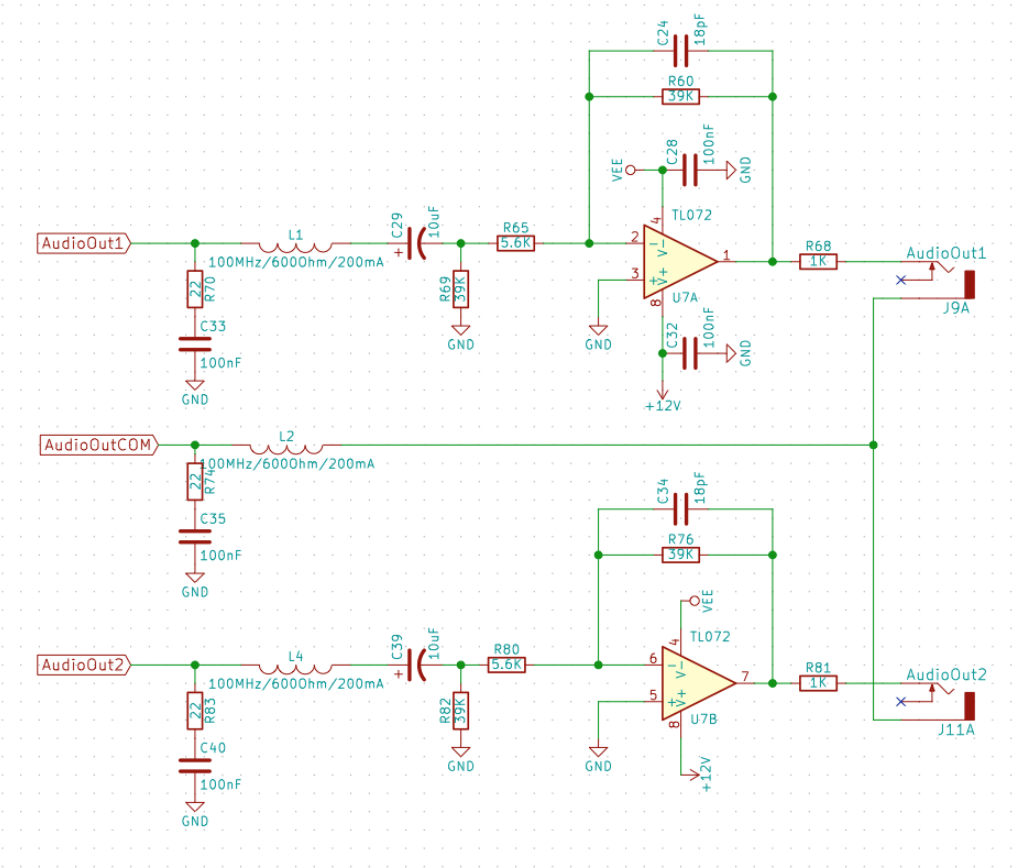


Abbildung 7: *Schaltung der Audio Ausgangs*

2.2.8 Analog Digital /Digital Analogwandler

Für die Analog/Digital-Wandlung wurde ein ADS1115 von Texas Instruments [12] verwendet. Wobei ein ADS1115 vier Eingänge mit jeweils 16-Bit Auflösung beinhaltet. Die Digital/Analogwandlung wird von einem AD5667R von Analog Devices [13] realisiert. Dieser Wandler hat ebenfalls eine Auflösung von 16-Bit und ist mit einer internen Referenzspannung versehen. Die Programmierung bzw. Datenverarbeitung wird im Kapitel 3.4.1 (ADS1115) und 3.4.2 (AD5667R) erläutert. Digital-Analog-Wandler wird nachfolgend mit DAC und Analog-Digital-Wandler mit ADC abgekürzt.

2.3 Prototyping der Breakoutschaltungen

Diese Schaltungen aus Kapitel 2.2 wurden im Vorfeld einzeln auf Breadboards aufgebaut. Dabei wurden verschiedene Variationen ausgeführt und diese verschiedenen Kombinationen getestet. Die Versorgung wurde auf einer Lochrasterplatine aufgebaut und um vier Potentiometern und zwei Schalter für die Tests der GPIOs bzw. der Analog-Digital-Wandler erweitert. In der folgenden Abbildung 8 sieht man das Entwicklungsboard, das Netzteil und die Steckbretter für die ADC/DAC-Tests sowie der Audioausgänge.

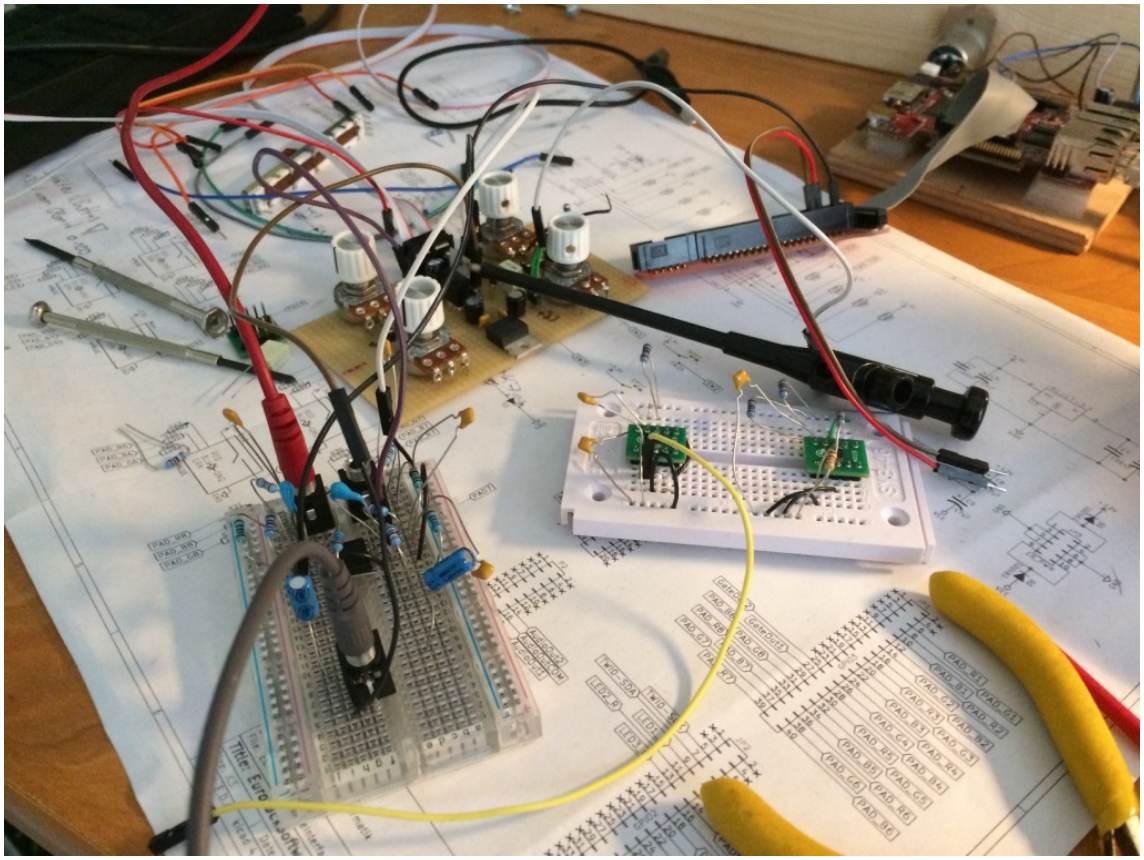


Abbildung 8: *Aufbau der Testschaltungen*

2.4 Layout der Platine

Nachdem die Funktion der einzelnen Schaltungen sichergestellt und der Schaltplan eingegeben wurde, konnte mit dem Layout der Leiterplatte begonnen werden. Zuerst wurde nach Anfertigung einer Skizze die Größe der Platine festgelegt, wobei die Gesamtbreite des Moduls zu beachten war, da auch diese Maße standardisiert sind. Die Platine hat vier Lagen, wobei eine innere Lage für die Masse verwendet wurde. Eine weitere Lage für eine Versorgungsspannung war nicht sinnvoll da fünf verschiedene Spannungen benötigt werden. (+12V, -12V, +5V, +3,3V und +3,3V des Entwicklungsboards). Danach sind alle Bedienelemente auf der Oberseite platziert worden. Anschließend wurden alle Schaltungsteile separat möglichst platzsparend angeordnet und erst danach auf der Platine platziert.

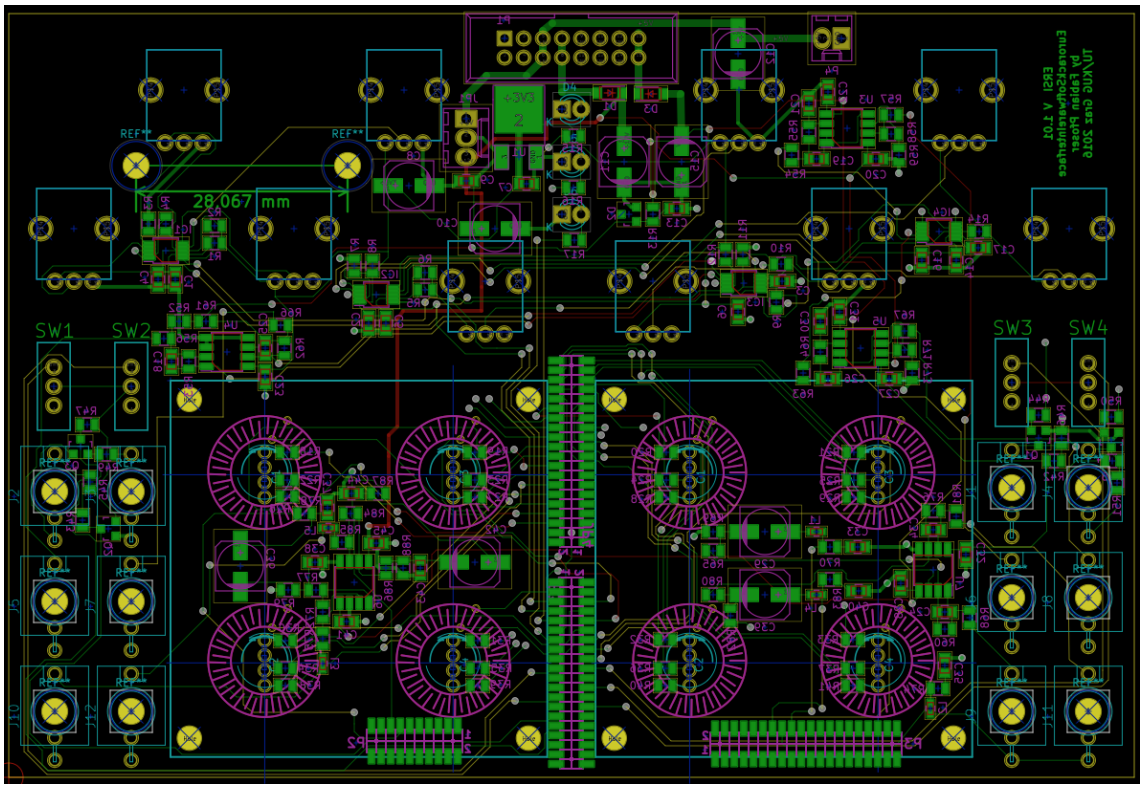


Abbildung 9: *Layout des Moduls*

2.5 Entwurf/Fertigung der Frontplatte und Assembling

Die Frontplatte wurde mit der freien Software FrontDesign von Schaeffer AG [14] gezeichnet, die für Frontplattendesigns konzipiert ist. Gefertigt wurde die Frontplatte im FabLab [15] der TU Graz . Die Dimension der Frontplatte wurde ebenfalls an den Eurorack-Standard angeglichen, woraus sich eine breite von 30 TE (Teilungseinheiten, englisch: HP- hoizontal pitch) [2] ergab. Um dem Modul ein einzigartiges Design zu verpassen, welches sich von anderen ähnlichen unterscheidet, wurde es nicht aus Aluminium sondern aus Holz bzw. Nuss-Furnier gefertigt.

Zuerst wurden alle SMD-Teile platziert und mittels eines umgebauten Backofens vom Institut 10 gelötet. Anschließend wurden die Bedienelemente manuell auf der Vorderseite verlötet. Mit Hilfe von zwei Schrauben wird das Olimex-Board an der Hauptplatine befestigt und mit Flachbandkabeln verbunden. Ein Bild vom fertigen Aufbau ist im Kapitel 4 zu sehen.

[2] Eine Teilungseinheit bei Elektronikbaugruppen entspricht 1/5 Zoll, also 5,08 mm

Abbildung 10: *Platine im Lötofen*

3 Entwicklung der Software

3.1 Auswahl des Minicomputers

Die Auswahl von Mini-Computern auf dem Markt ist schon sehr groß da auch die Verwendung in Projekten immer häufiger wird. Speziell für Prototypen ist es oft einfacher auf gut verbreitete Varianten zurückzugreifen. Durch die Wahl eines Minicomputers mit einem Betriebssystem ergeben sich auch wesentlich mehr Möglichkeiten anstatt der Verwendung eines Mikrocontrollers.
Da am Institut für Elektronische Musik und Akustik der Olinuxino A20 Lime2 (Siehe 11) schon für mehrere Projekte verwendet wurde und dieser dort zur Verfügung stand, wurde auch dieses Projekt damit realisiert. Ein wesentlicher Vorteil ist, dass sich die Firma Olimex ebenfalls zur Open Hardware Community bekennt und jegliche Informationen, Schaltungen, Layouts und Daten der Produkte frei zur Verfügung stellt und darauf achtet keine closed-software zu verwenden.
Das Board besitzt einen Allwinner A20 dual core Cortex-A7 Prozessor mit einer Taktfrequenz von 1 GHz pro Kern, 1 GB DDR3 RAM und sehr viel General-Purpose-Input/Output-Ports (Nachfolgend mit GPIO abgekürzt). Damit ist dieses Entwicklungsboard ist im ersten Augenschein überdimensioniert für dieses Projekt, lässt sich aber damit gut erweitern.

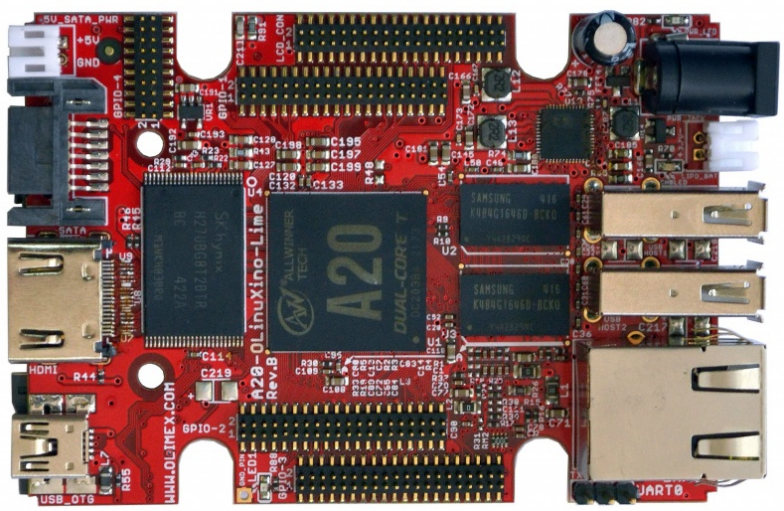


Abbildung 11: *Olinuxino A*20 *Lime*2

3.2 Aufsetzten des Betriebssystems/Arbeitsumgebung

Als Basis wurde der von Olimex empfohlene Debian OS-System mit einem von Olimex angepassten Kernel, welcher auf den sunxi-Kernel [16] basiert, verwendet. Der darin verwendete gut getestete Linux-Kernel wurde speziell für den Olinuxino Lime2 mit vier Gigabyte EMMC-Falsh Speicher konfiguriert und kompiliert.

3.3 Programmierung und Debugging

Alle weiteren Installationen und die Programmierung werden über SSH (Secure Shell) [17] realisiert somit können alle Aktionen von einem anderen Rechner ausgeführt werden. Die Installationen der Programme funktioniert wie bei jeder Linuxumgebung, auf diese Vorgänge werde ich aber in dieser Arbeit nicht weiter eingehen.

3.3.1 Signalverarbeitung mit Python

Olimex stellt für jedes Entwicklungsboard ein eigenes Python-Modul zur Verfügung. Damit wurde die Handhabung der General-Purpose-Input/Output-Ports (Nachfolgend mit GPIO abgekürzt) und dem I2C-Bus vereinfacht. Die GPIO-Ports werden für die LEDs, Schalter, Druckknöpfe sowie Gate Ein- und Ausgänge verwendet. Der I2C-Bus wird bei der Kommunikation zwischen den Analog-Digital- und Digital-Analogwandlern benötigt. Wie dieses Python-Modul verwendet wird wird in den Quellcodes der Wandler in Kapitel 6.1.3 und 6.1.4 gezeigt.

3.4 Projektbezogene Python-Skripte

Die Verarbeitung der Daten von den Bedienelementen sollte möglichst einfach sein und auch Benutzer mit wenig Erfahrung in der Programmierung von Hardware ermöglicht werden. Der Kern der Datenverarbeitung beschränkt sich auf drei Python-Skripte welche in diesem Kapitel beschrieben werden.

3.4.1 eupyADS1115 - Analog-Digital-Wandlung

Die Firma Adafruit Industries [18] bietet ein Breakoutboard mit diesem ADC an(Siehe [19] und stellt auch ein Python-Skript für den Raspery Pi, welcher dem Olinuxino sehr ähnlich ist, zur Verfügung. Dieses Skript wurde als Vorlage verwendet und dementsprechend angepasst um den Anforderungen dieses Projektes gerecht zu werden.
Der ADS1115 hat verschiedene Betriebsarten, die alle implementiert wurden, wobei hier auf die Projektrelevanten Funktionen eingegangen wird. Alle drei ADS1115 werden im Single-Ended-Input-Modus (Siehe Datenblatt, Seite 12: [20]), das bedeuted dass simultan insgesamt zwölf Werte von den jeweiligen Bedienelementen ausgelesen werden können. Als Ergebnis erhält man einen Integerwert zwischen 0 und 26480 da die Rückgabewerte bei der gewählten Verstärkung über den gesamten Bereich linear verteilt sind. Außerdem erhält man ohnehin, im Vergleich zum Werteumfang von MIDI-Daten, welche von 7-bit 0 bis 127 bzw. 14-bit definiert sind, eine über 200-fache bzw. 4-fache Auflösung. (MIDI-Standard: [21]

```
    def read_adc(self, address="GND", i2cbus=0, channel=0, gain=1, data_rate=None):
        """Read a single ADC channel and return the ADC value as a signed integer
        result.  Channel must be a value within 0-3.
        """
        validbus = False
        validaddress = False

        if i2cbus == 0:
            openbus = I2CBUS_0
            validbus = True
        if i2cbus == 1:
            openbus = I2CBUS_1
            validbus = True
        if i2cbus == 2:
            openbus = I2CBUS_2
            validbus = True
        elif validbus == False:
            print "I2C-Bus have to be '0', '1' or '2'!"
            exit()

        if address == "GND":
            dacaddr = ADS1115_ADDR_GND
            validaddress = True
        if address == "VDD":
            dacaddr = ADS1115_ADDR_VDD
            validaddress = True
        if address == "SDA":
            dacaddr = ADS1115_ADDR_SDA
            validaddress = True
        if address == "SCL":
            dacaddr = ADS1115_ADDR_SCL
            validaddress = True

        if validaddress == False:
            print "Adress have to be 'GND', 'VDD' ,'SDA' or 'SCL'!"
            exit()

        i2c.init(openbus)  #Initialize module to use /dev/i2c-0
```

```
39          i2c.open(dacaddr)

41          assert 0 <= channel <= 3, 'Channel must be a value within 0-3!'
            # Perform a single shot read and set the mux value to the channel plus
43          # the highest bit (bit 3) set.
            return self._read(channel + 0x04, dacaddr, gain, data_rate,
    ADS1115_CONFIG_MODE_SINGLE)
```

Listing 1: Funktion zum initialisieren des AD-Wandlers

Folgende Funktion wird von der vorherigen aufgerufen, konfiguriert den ADC für die aktuelle Wandlung und speichert den Wert im Register ab:

```
    def _read(self, mux, dacaddr, gain, data_rate, mode):
2           """Perform an ADC read with the provided mux, gain, data_rate, and mode
            values.  Returns the signed integer result of the read.
4           """

6           config = ADS1115_CONFIG_OS_SINGLE  # Go out of power-down mode for
    conversion.
            # Specify mux value.
8           config |= (mux & 0x07) << ADS1115_CONFIG_MUX_OFFSET
            # Validate the passed in gain and then set it in the config.
10          if gain not in ADS1115_CONFIG_GAIN:
                raise ValueError('Gain must be one of: 2/3, 1, 2, 4, 8, 16')
12          config |= ADS1115_CONFIG_GAIN[gain]
            # Set the mode (continuous or single shot).
14          config |= mode
            # Get the default data rate if none is specified
16          if data_rate is None:
                data_rate = self._data_rate_default()
18          # Set the data rate
            config |= self._data_rate_config(data_rate)
20          config |= ADS1115_CONFIG_COMP_QUE_DISABLE  # Disble comparator mode.
            # Send the config value to start the ADC conversion.
22          # Explicitly break the 16-bit value down to a big endian pair of bytes.
            i2c.write([((dacaddr << 8) | ADS1115_POINTER_CONFIG), (config >> 8) & 0xFF,
     config & 0xFF])
24          # Wait for the ADC sample to finish based on the sample rate plus a
            # small offset to be sure (0.1 millisecond).
26          time.sleep(1.0/data_rate+0.0001)

28          # Retrieve the result.
            i2c.write([ADS1115_POINTER_CONVERSION])
30          result = i2c.read(2)
            i2c.close()
32
            return self._conversion_value(result[1], result[0])
```

Listing 2: Konfiguration der aktuellen Wandlung

Anschließend wird der Wert noch beschränkt bzw skaliert um einen korrekten Rückgabewert zu erhalten.

```
    def _conversion_value(self, low, high):
        # Convert to 16-bit signed value.
        value = ((high & 0xFF) << 8) | (low & 0xFF)
        # Check for sign bit and turn into a negative value if set.
        if value & 0x8000 != 0:
            value -= 1 << 16

        #scale all pots/cv in range 1-26480
        if value <= 0:
            value = 1
        if value > 26480:
            value = 26480
        value = (-1)*(value - 26481)
        return value
```

Listing 3: Aufbereiten des Rückgabewertes

3.4.2 eupyAD5667R - Digital-Analog-Wandlung

Die Programmierung des AD5667R von Analog Devices ist aufgrund des wesentlich geringeren Funktionsumfang als beim ADS1115, deutlich einfacher. Damit der Ausgang auf 0-8V beschränkt ist wurde der Wertebereich auf 0-48300 verwendet. Dadurch ergibt sich eine theoretische Auflösung von 0,16 mV.

Hier wurde ebenfalls die Deklaration der Variablen für die Initialisierung und Konfiguration des DA-Wandlers dem Datenblatt entnommen welche in (6.1.4) (Zeile 25-55) zu sehen ist. Für eine Wandlung muss folgende Funktion aufgerufen werden, welche den jeweiligen Kanal initialisiert und den gewünschten Integer-Wert als Spannung ausgibt.

```
    def write_dac(self, address = "GND", i2cbus = 0, channel= "a", value = 0):
        # 24bit Input Shift Register
        # 8bit-command, 8bit-MSB, 8bit-LSB

        validbus = False
        validaddress = False

        if i2cbus == 0:
            openbus = I2CBUS_0
            validbus = True
        elif i2cbus == 1:
            openbus = I2CBUS_1
            validbus = True
        elif i2cbus == 2:
            openbus = I2CBUS_2
            validbus = True
        if validbus == False:
            print "I2C-Bus have to be '0', '1' or '2'!"
            exit()

        if address == "GND":
            dacaddr = AD5667R_ADDR_GND
            validaddress = True
        elif address == "VDD":
            dacaddr = AD5667R_ADDR_VDD
            validaddress = True
        elif address == "NC":
            dacaddr = AD5667R_ADDR_NC
            validaddress = True
        if validaddress == False:
            print "Adress have to be 'GND', 'VDD' or 'NC'!"
            exit()

        validchannel = False

        if channel == "a":
            dacout = AD5667R_DACA
            validchannel = True
        elif channel == "b":
            dacout = AD5667R_DACB
            validchannel = True
        elif channel == "ab":
            dacout = AD5667R_DACAB
            validchannel = True
        if validchannel == False:
            print "Channel have to be 'a', 'b' or 'ab' for both channels!"
            exit()

        i2c.init(openbus)  #Initialize module to use /dev/i2c-0
        i2c.open(dacaddr)

        # init before conversion / datasheet page 27
        #set PowerUP both DACs
        i2c.write([AD5667R_PWRUPDWN, 0b00000000, dacout])
```

```
56          #set internal reference ON
            i2c.write([AD5667R_INTREF, 0b00000000, 0b00000001])
58          #i2c.close()

60          #limit output to 8V, full range: 8,28V
            if value > 48300:
62              value = 48300
            if value <= 0:
64              value = 1

66          Highint = (value >> 8) & 0xFF
            Lowint = value & 0xFF
68          #cvout, write data, Bit 19:17 DAC Select DAC1, 00 000 000 +16DATA-Bits
            i2c.write([(AD5667R_WUDAC | dacout), Highint, Lowint])
70          #close i2c connection
            i2c.close()
```

Listing 4: Funktion für Initialisierung und DA-Wandlung

3.4.3 eupyControls

Diese Klasse wurde implementiert um die Programmierung möglichst einfach zu gestalten. Sie enthält alle Initialisierungen der GPIOs und das Management der Analog-Digital- bzw. Digital-Analog-Wandler. Der vollständige Quellcode ist im Kapitel 6.1.2 zu sehen. Der folgende Code demonstriert den einfachen Zugriff auf die einzelnen Bedienelemente des Moduls:

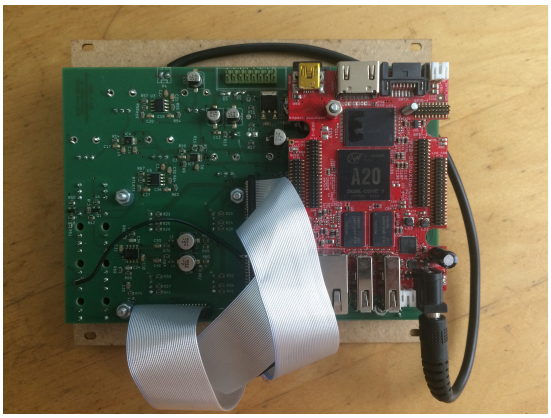
Diese Klasse enthält alle Initialisierungen der GPIOs und das Management der Analog-Digital- bzw. Digital-Analog-Wandler. Der vollständige Quellcode ist im Kapitel 6.1.2 zu sehen.
Der folgende Code demonstriert den einfachen Zugriff auf die einzelnen Bedienelemente des Moduls:
Zuerst wird der Zugriff auf die GPIOs demonstriert was die Silikondruckknöpfe und die dazugehörigen LEDs, die einzelnen LEDs, die Schalter und die "gateEin- und Ausgänge umfasst. Die unteren drei Beispiele demonstrieren den Zugriff auf den DAC bzw. die ADCs.

```
from eupyControls import Controls

c = Controls()

#Beim druecken von Pad 1, leuchtet Pad 1 rot
if c.pad(1):
    c.padled(1, 'r', 1)

#Beim druecken von Pad 3, leuchtet Pad 5 blau
if c.pad(3):
    c.padled(5, 'b', 1)

#Beim Betaetigen von Kippschalter 1 leuchtet LED 1 (gruen)
if c.switch(1):
    c.led(1,1)

#Beim Anliegen eines Signals am Gate-Input 1, Ausgabe eines Strings
if c.gatein(1):
    print "gate in channel 1"

#Setzt Gate-Output Kanal 1 auf "high"
c.gateout(1, 1)

#Gibt am CV-Output 1 den Integer-Wert 30000 aus
c.cvout(1, 30000)

#Speichert den aktuellen Wert von Potentiometer 4 in Variable x
x = c.pot(4)

#Speichert den aktuellen Wert von CV-Input 2 in Variable x
x = c.cvin(2)
```

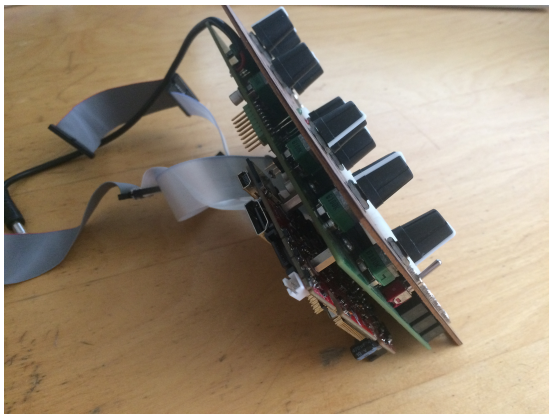
Listing 5: Zugriff auf die Bediehnelemente mit der Klasse eupyControls.py

4 Eurorack Computer

Abbildung 12: *Voderansicht des Moduls*



(a) Rückseite



(b) Seite

Abbildung 13: Rück- und Seitenansicht des Moduls

4.1 Eurorack Computer Anwendungen

Im folgendem werden als Beispiel ein „Dual-Slope-LFO" und ein „Dual-ADSR"gezeigt:

Wie vorher schon erwähnt, wurde eine Python-Anbindung realisiert, damit man beliebige Anwendungen in das Modularsystem integrieren kann. Funktionsgeneratoren gibt es viele verschiedene und bilden den Kern für Modulationen in jedem Synthesizer. Der LFO (Low-Frequency-Oscillator) [22] und die ADSR-Hüllkurve [23] sind die gängisten und praktisch in jedem Synthesizer verbaut. Der Unterschied zwischen einem LFO zu einem Slope-Generator ist, dass man Zugriff auf steigende- und fallende Flanke hat.

4.1.1 Anwendungsbeispiel: Dual-Slope-LFO

Abbildung 14: *Belegung der Bedienelemente : DualSlopeLFO*

Bei diesem digitalen Dual-Slope-Generator hat man Zugriff auf drei Parameter je Einheit. Die Form der Hüllkurve lässt sich mit einen Potentiometer von einem Sägezahn zu einem umgekehrten Sägezahn überblenden, somit ergibt sich in der Mitte eine Dreieckform. Der zweite Regler ist für die Geschwindigkeit verantwortlich, wobei sich insgesamt eine Bandbreite von 2,45Hz (0,4078s) bis 0,00037Hz (45min) ergibt. Der dritte Potentiometer beeinflusst die Auflösung der Hüllkurve, was mit einer analogen Schaltung eher schwierig zu realisieren ist. Unter Auflösung versteht sich hier die Anzahl der Abtastpunkte pro Periode der Hüllkurve, welche von 20 - 5000 regelbar sind. Der sehr geringe Wert ist interessant wenn man eine stufige Hüllkurve benötigt, die Obergrenze wurde so hoch gewählt damit auch bei sehr langen

Hüllkurven stufenlose Ergebnisse realisierbar sind.
Weiters lässt sich jede der beiden Ausgangsspannungen von 0V - 8V auf 6V, 4V oder 2V beschränken. Dieses Untermenü erreicht man durch betätigen des jeweiligen Kippschalters, anschließend kann man über die Silikondruckknöpfe den Spannungsbereich wählen.
Anschließend wurde noch zu jeder Einheit ein Sinus-LFO hinzugefügt, wodurch noch jeweils zwei weitere veränderliche Parameter dazukommen. Mit dem ersten Potentiometer lässt sich die Modulationsstärke regeln, welche von 0% - 30% einstellbar ist. Der zweite regelt die Geschwindigkeit des Modulations-LFOs welche von der vierfachen Frequenz der Gesamtperiode bis zur 80-fachen Frequenz variiert werden kann. Jede Hüllkurve lässt sich über den jeweiligen Gate-Eingang zurücksetzten, über die äusseren blauen Pads kann man dies auch manuell veranlassen. Zu guter Letzt wurde noch jeder Einheit eine LED zugeordnet.

Für die Umsetzung dieser Anwendung wurden nur Standardbibliotheken von Python verwendet. Scipy und Numpy, welche für die Signalverarbeitung in Python sehr häufig verwendet werden.
Die Benötigten Module, das Hauptprogramm welches die Threads startet und die Limitierung der Ausgangsspannung sind dem vollständigen Quellcode im Kapitel zu entnehmen. Im folgenden wird nur der Code des eigentlichen Slope-Generators erläutert:

Zuerst werden die Werte der Potentiometer festgehaten, wobei bei jedem Zugriff auf den I2C-Bus ein Thread-Lock durchgeführt wird um Kollisionen zu vermeiden. Anschließend die Abfrage für das Untermenü zur Spannungsbegrenzung am Ausgang. Im nächsten Schritt wird das Array mit der Wellenform erzeugt, abhängig von der eingestellten Form und der Auflösung. In den verschachtelten For-Schleifen wird das Array mit der Wellenform über den DAC ausgegeben, wobei die zweite Schleife die Wartezeit bis zum nächsten Ausgabewert kontrolliert.

```
def writeCV1(threadname):
    global outrange1

    while True:
        # get values from potentiometers
        lockMe.acquire()
        speed1 = int(c.pot(1) / 4)
        resolution1 = c.pot(3)
        shape1 = c.pot(2)
        lfospeed1 = c.pot(4)
        lfoamount1 = c.pot(9)
        lockMe.release()
        reset1 = False
        # first switch to enter range menu from left slope
        if c.switch(1):
            rangeone()

        # Scale shape-value betweeen 0 and 1
        # shape from sawtooth to reversed-sawtooth, middle position: triangle
        scalshape1 = format(shape1/26480.0, '.2f')
        # list of 1 periode of values of the wave
        # resolution from 20 - 5000
        t = np.linspace(0, 1, int(20+resolution1/5.317))
        wave1 = (signal.sawtooth(2 * np.pi * 1 * t, float(scalshape1)) + 1)/2
        #LFO frequency from 4times to 80times / maximal amount 30%
        lfowave1 = ((np.sin(8 * int(lfospeed1/1393 + 1) * np.pi * t))+1)/2 * (0.3 *
    lfoamount1/26480.0)
```

```
27
        for i in range(len(wave1) - 1):
29          # LFO-LED
            if i < (len(wave1)/5):
31              c.led(1, 1)
            else:
33              c.led(1,0)
            # write cv output - base slope plus lfo-mod
35          lockMe.acquire()
            c.cvout(1, int(((wave1[i]+lfowave1[i]) * 0.77) * 48299.0 * outrange1))
37          lockMe.release()
            # time from 0.4078s - 2700s --> 2,45Hz - 0,00037Hz (45min)
39          for j in range(int(6620/speed1)):
                time.sleep((2700.0/speed1/int(6620/speed1))/(len(wave1) - 1))
41              #gate / reset
                if c.gatein(1) or c.pad(1):
43                  i = 0
                    j = 0
45                  reset1 = True
                    break
47          if reset1:
                reset1 = False
49              break
```

Listing 6: writeCV1-Thread: Berechnung und Ausgabe der Hüllurve

4.1.2 Anwendungsbeispiel: Dual-ADSR

Abbildung 15: *Belegung der Bedienelemente : DualADSR*

Bei dieser digitalen Hüllkurve hat man Zugriff auf die vier Phasen einer vereinfachten Hüllkurve eines Klangs -Attack (Anstieg), Decay (Abfall), Sustain (Halten) und Release (Freigeben). Die Vier Phasen werden durch die blaue LED unter den Druckknöpfen dargestellt wobei die obere Reihe für die erste Hüllkurve und die untere Reihe für die zweite Hüllkurve zuständig ist. Solange ein Signal am Gate-Eingang

anliegt, wird die Hüllkurve an den CV-Ausgängen von 0V - 8V ausgegeben. Ist diese fertig durchlaufen, leuchtet der letzte Druckknopf weiss und wartet auf ein neues Gate-Signal am Eingang. Die maximale Dauer einer Hüllkurve beträgt (wird noch genau gemessen).
Für die Umsetzung dieser Anwendung wurden ebenfalls nur die beiden Standardbibliotheken, Scipy und Numpy, verwendet.
Die Benötigten Module und das Hauptprogramm welches die Threads startet sind dem vollständigen Quellcode zu entnehmen. Im folgenden wird nur der Code der eigentlichen Hüllkurve erläutert:

Zuerst werden wieder die Werte der Potentiometer festgehalten, wobei auch bei jedem Zugriff auf den I2C-Bus ein Thread-Lock durchgeführt wird um Kollisionen zu vermeiden. Anschließend werden die Werte skaliert und in den For-Schleifen die vier Phasen abgearbeitet und am DAC bzw. den CV-Ausgängen ausgegeben.

```
def adsr1(threadname):
    while True:
        global fullenv1
        # LEDs
        c.padled(1, 'g', 1)
        c.padled(3, 'g', 1)
        c.padled(5, 'g', 1)
        c.padled(7, 'g', 1)

        if c.gatein(1) and not fullenv1:
            lockMe.acquire()
            attack = int(c.pot(1) / 26.506 + 1) #1-1000
            decay = int(c.pot(2) / 26.506 + 1)
            sustain = int(c.pot(3) / 26.506 + 1)
            release = int(c.pot(4) / 26.506 + 1)
            lockMe.release()

            dur = attack + decay + sustain + release
            m_a = (1. / attack)
            m_d = (sustain/1000.0 - 1.) / decay
            m_r = - sustain/1000.0 * 1. / release

            len_a = int(attack + .5)
            len_d = int(decay + .5)
            len_r = int(release + .5)
            len_s = int(dur + .5) - len_a - len_d - len_r

            #attack
            for sample in xrange(len_a):
                if not c.gatein(1):
                    break
                lockMe.acquire()
                c.cvout(1, int((sample * m_a) * 48299.0))
                lockMe.release()
                c.padled(1, 'b', 1)
                c.padled(3, 'b', 0)
                c.padled(5, 'b', 0)
                c.padled(7, 'b', 0)
                c.padled(1, 'g', 0)
                c.padled(3, 'g', 1)
                c.padled(5, 'g', 1)
                c.padled(7, 'g', 1)
                time.sleep(0.001)
            #decay
            for sample in xrange(len_d):
                if not c.gatein(1):
                    break
                lockMe.acquire()
                c.cvout(1, int((1. + sample * m_d) * 48299.0))
                lockMe.release()
```

```
                c.padled(1, 'b', 0)
                c.padled(3, 'b', 1)
                c.padled(5, 'b', 0)
                c.padled(7, 'b', 0)
                c.padled(1, 'g', 1)
                c.padled(3, 'g', 0)
                c.padled(5, 'g', 1)
                c.padled(7, 'g', 1)
                time.sleep(0.001)
            #sustain
            for sample in xrange(len_s):
                if not c.gatein(1):
                    break
                lockMe.acquire()
                c.cvout(1, int(sustain/1000.0 * 48299.0))
                lockMe.release()
                c.padled(1, 'b', 0)
                c.padled(3, 'b', 0)
                c.padled(5, 'b', 1)
                c.padled(7, 'b', 0)
                c.padled(1, 'g', 1)
                c.padled(3, 'g', 1)
                c.padled(5, 'g', 0)
                c.padled(7, 'g', 1)
                time.sleep(0.001)
            #release
            for sample in xrange(len_r):
                if not c.gatein(1):
                    break
                lockMe.acquire()
                c.cvout(1, int((sustain/1000.0 + sample * m_r) * 48299.0))
                lockMe.release()
                c.padled(1, 'b', 0)
                c.padled(3, 'b', 0)
                c.padled(5, 'b', 0)
                c.padled(7, 'b', 1)
                c.padled(1, 'g', 1)
                c.padled(3, 'g', 1)
                c.padled(5, 'g', 1)
                c.padled(7, 'g', 0)
                time.sleep(0.001)

                # Boolean for not looping the envelope if gate is long pressed
                fullenv1 = True
        if not c.gatein(1):
            c.padled(1, 'b', 0)
            c.padled(3, 'b', 0)
            c.padled(5, 'b', 0)
            c.padled(7, 'b', 0)
            fullenv1 = False
            lockMe.acquire()
            c.cvout(1, 0)
            lockMe.release()
```

Listing 7: adsr1-Thread: Berechnung und Ausgabe der Hüllurve

Referenzen

[1] Internationale Norm für 19-Zoll-Systeme (DIN 41494 / IEC 297-3 / IEEE 1001.1).

[2] Doepfer Musikelektronik, mechanisches Konzept A-100 System. http://www.doepfer.de/a100_man/a100m_d.htm.

[3] ARDUINO. https://www.arduino.cc/.

[4] Raspery Pi. https://www.raspberrypi.org/.

[5] Open Source Hardware. http://freedomdefined.org/OSHW.

[6] Alpha Potentiometers. http://alphapotentiometers.net/.

[7] Salecom Switches. "https://www.salecom.com/de/index/index.html.

[8] Thonk Synth DIY. https://www.thonk.co.uk/.

[9] Sparkfun Electronics. https://www.sparkfun.com/.

[10] KiCad EDA. http://kicad-pcb.org/.

[11] Mutable Instruments. https://mutable-instruments.net/.

[12] Texas Instruments. https://www.ti.com/.

[13] Analog Devices. http://www.analog.com/en/index.html.

[14] Schaeffer AG. https://www.schaeffer-ag.de/downloads/frontplatten_designer/?no_cache=1.

[15] FabLab TU Graz. http://fablab.tugraz.at/.

[16] Linux Sunxi. https://linux-sunxi.org/Main_Page.

[17] Secure Shell. https://de.wikipedia.org/wiki/Secure_Shell.

[18] Adafruit Industries. https://www.adafruit.com/.

[19] Adafruit ADS1x15 Breakoutboard. https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/overview.

[20] ADS1115 Datasheet. https://cdn-shop.adafruit.com/datasheets/ads1115.pdf.

[21] MIDI Standard. https://www.midi.org/specifications/item/the-midi-1-0-specification.

[22] LFO. https://de.wikipedia.org/wiki/Low_Frequency_Oscillator.

[23] ADSR. https://de.wikipedia.org/wiki/ADSR.

5 Abbildungsverzeichnis

Abbildungsverzeichnis

6 Anhang

6.1 Quellcodes

6.1.1 eupyDualSlopeGenerator.py

```
#!/usr/bin/env python

__author__ = "Fabian Pfoser"
__copyright__ = "Copyright 2016"
__credits__ = ["Fabian Pfoser"]
__license__ = "GPL"
__version__ = "1.0"
__maintainer__ = __author__
__email__ = "f.pfoser@student.tugraz.at"

import os
import sys
import subprocess
from eupyControls import Controls
import time
from scipy import signal
import numpy as np
import threading

if not os.getegid() == 0:
    sys.exit('Script must be run as root')

def rangeone():
    global outrange1
    ledReset()
    while c.switch(1):
        range2V = c.pad(3)
        range4V = c.pad(4)
        range6V = c.pad(5)
        range8V = c.pad(6)
        if range2V:
            c.padled(3, 'g', 1)
            c.padled(4, 'g', 0)
            c.padled(5, 'g', 0)
            c.padled(6, 'g', 0)
            outrange1 = 0.25

        elif range4V:
            c.padled(3, 'g', 1)
            c.padled(4, 'g', 1)
            c.padled(5, 'g', 0)
            c.padled(6, 'g', 0)
            outrange1 = 0.5

        elif range6V:
            c.padled(3, 'g', 1)
            c.padled(4, 'g', 1)
            c.padled(5, 'g', 1)
            c.padled(6, 'g', 0)
            outrange1 = 0.75

        elif range8V:
            c.padled(3, 'g', 1)
            c.padled(4, 'g', 1)
            c.padled(5, 'g', 1)
            c.padled(6, 'g', 1)
            outrange1 = 1.0

        if outrange1 == 0.25:
            c.padled(3, 'g', 1)
            c.padled(4, 'g', 0)
            c.padled(5, 'g', 0)
            c.padled(6, 'g', 0)

        elif outrange1 == 0.5:
```

```
66              c.padled(3, 'g', 1)
                c.padled(4, 'g', 1)
68              c.padled(5, 'g', 0)
                c.padled(6, 'g', 0)
70
            elif outrange1 == 0.75:
72              c.padled(3, 'g', 1)
                c.padled(4, 'g', 1)
74              c.padled(5, 'g', 1)
                c.padled(6, 'g', 0)
76
            elif outrange1 == 1.0:
78              c.padled(3, 'g', 1)
                c.padled(4, 'g', 1)
80              c.padled(5, 'g', 1)
                c.padled(6, 'g', 1)
82      c.padled(3, 'g', 0)
        c.padled(4, 'g', 0)
84      c.padled(5, 'g', 0)
        c.padled(6, 'g', 0)
86      c.padled(1, 'b', 1)
        c.padled(7, 'b', 1)
88      c.padled(4, 'b', 1)
        c.padled(6, 'b', 1)
90
def rangetwo():
92      global outrange2
        ledReset()
94      while c.switch(2):
            range2V = c.pad(3)
96          range4V = c.pad(4)
            range6V = c.pad(5)
98          range8V = c.pad(6)
            if range2V:
100             c.padled(3, 'r', 1)
                c.padled(4, 'r', 0)
102             c.padled(5, 'r', 0)
                c.padled(6, 'r', 0)
104             outrange2 = 0.25

106         elif range4V:
                c.padled(3, 'r', 1)
108             c.padled(4, 'r', 1)
                c.padled(5, 'r', 0)
110             c.padled(6, 'r', 0)
                outrange2 = 0.5
112
            elif range6V:
114             c.padled(3, 'r', 1)
                c.padled(4, 'r', 1)
116             c.padled(5, 'r', 1)
                c.padled(6, 'r', 0)
118             outrange2 = 0.75

120         elif range8V:
                c.padled(3, 'r', 1)
122             c.padled(4, 'r', 1)
                c.padled(5, 'r', 1)
124             c.padled(6, 'r', 1)
                outrange2 = 1.0
126
            if outrange2 == 0.25:
128             c.padled(3, 'r', 1)
                c.padled(4, 'r', 0)
130             c.padled(5, 'r', 0)
                c.padled(6, 'r', 0)
132
            elif outrange2 == 0.5:
134             c.padled(3, 'r', 1)
                c.padled(4, 'r', 1)
136             c.padled(5, 'r', 0)
                c.padled(6, 'r', 0)
138
```

```
        elif outrange2 == 0.75:
            c.padled(3, 'r', 1)
            c.padled(4, 'r', 1)
            c.padled(5, 'r', 1)
            c.padled(6, 'r', 0)

        elif outrange2 == 1.0:
            c.padled(3, 'r', 1)
            c.padled(4, 'r', 1)
            c.padled(5, 'r', 1)
            c.padled(6, 'r', 1)
    c.padled(3, 'r', 0)
    c.padled(4, 'r', 0)
    c.padled(5, 'r', 0)
    c.padled(6, 'r', 0)
    c.padled(1, 'b', 1)
    c.padled(7, 'b', 1)
    c.padled(4, 'b', 1)
    c.padled(6, 'b', 1)

#
    ##############################################################################################################

### Read Potentiometers
#
    ##############################################################################################################

# def read(threadname):
#     global speed1
#     global speed2
#     global resolution1
#     global resolution2
#     global shape1
#     global shape2
#     global lfospeed1
#     global lfospeed2
#     global lfoamount1
#     global lfoamount2
#
#     while True:
#         # get values from potentiometers
#         lockMe.acquire()
#         speed1 = c.pot(1)
#         resolution1 = c.pot(3)
#         shape1 = c.pot(2)
#         lfospeed1 = c.pot(4)
#         lfoamount1 = c.pot(9)
#         speed2 = c.pot(8)
#         resolution2 = c.pot(6)
#         shape2 = c.pot(7)
#         lfospeed2 = c.pot(5)
#         lfoamount2 = c.pot(10)
#         lockMe.release()

#
    ##############################################################################################################

### No 1
#
    ##############################################################################################################

def writeCV1(threadname):
    global outrange1
    # global speed1
    # global resolution1
    # global shape1
    # global lfospeed1
    # global lfoamount1
    while True:
        # get values from potentiometers
        lockMe.acquire()
        speed1 = int(c.pot(1) / 4)
        resolution1 = c.pot(3)
```

```
        shape1 = c.pot(2)
        lfospeed1 = c.pot(4)
        lfoamount1 = c.pot(9)
        lockMe.release()
        reset1 = False
        # first switch to enter range menu from left slope
        if c.switch(1):
            rangeone()

        # Scale shape-value betweeen 0 and 1
        # shape from sawtooth to reversed-sawtooth, middle position: triangle
        scalshape1 = format(shape1/26480.0, '.2f')
        # list of 1 periode of values of the wave
        # resolution from 20 - 5000
        t = np.linspace(0, 1, int(20+resolution1/5.317))
        wave1 = (signal.sawtooth(2 * np.pi * 1 * t, float(scalshape1)) + 1)/2
        #LFO frequency from 4times to 80times / maximal amount 30%
        lfowave1 = ((np.sin(8 * int(lfospeed1/1393 + 1) * np.pi * t))+1)/2 * (0.3 *
    lfoamount1/26480.0)

        for i in range(len(wave1) - 1):
            # LFO-LED
            if i < (len(wave1)/5):
                c.led(1, 1)
            else:
                c.led(1,0)
            # write cv output - base slope plus lfo-mod
            lockMe.acquire()
            c.cvout(1, int(((wave1[i]+lfowave1[i]) * 0.77) * 48299.0 * outrange1))
            lockMe.release()
            # time from 0.4078s - 2700s --> 2,45Hz - 0,00037Hz (45min)
            for j in range(int(6620/speed1)):
                time.sleep((2700.0/speed1/int(6620/speed1))/(len(wave1) - 1))
                #gate / reset
                if c.gatein(1) or c.pad(1):
                    i = 0
                    j = 0
                    reset1 = True
                    break
            if reset1:
                reset1 = False
                break

#
    ##########################################################################################################

### No 2
#
    ##########################################################################################################

def writeCV2(threadname):
    global outrange2
    # global speed2
    # global resolution2
    # global shape2
    # global lfospeed2
    # global lfoamount2
    while True:
        # get values from potentiometers
        lockMe.acquire()
        speed2 = int(c.pot(8) / 4)
        resolution2 = c.pot(6)
        shape2 = c.pot(7)
        lfospeed2 = c.pot(5)
        lfoamount2 = c.pot(10)
        lockMe.release()
        reset2 = False
        # second switch to enter range menu from right slope
        if c.switch(2):
            rangetwo()

        # Scale shape-value betweeen 0 and 1
        # shape from sawtooth to reversed-sawtooth, middle position: triangle
```

```
272         scalshape2 = format(shape2/26480.0, '.2f')
            # list of 1 periode of values of the wave
274         # resolution from 20 - 5000
            t = np.linspace(0, 1, int(20+resolution2/5.317))
276         wave2 = (signal.sawtooth(2 * np.pi * 1 * t, float(scalshape2)) + 1)/2
            #LFO frequency from 4times to 80times / maximal amount 30%
278         lfowave2 = ((np.sin(8 * int(lfospeed2/1393 + 1) * np.pi * t))+1)/2 * (0.3 *
     lfoamount2/26480.0)

280         for k in range(len(wave2) - 1):
                # LFO-LED
282             if k < (len(wave2)/5):
                    c.led(3, 1)
284             else:
                    c.led(3,0)
286             # write cv output - base slope plus lfo-mod
                lockMe.acquire()
288             c.cvout(2, int(((wave2[k] + lfowave2[k]) * 0.77) * 48299.0 * outrange2)
    )
                lockMe.release()
290
                # time from 0.4078s - 2700s --> 2,45Hz - 0,00037Hz (45min)
292             for l in range(int(6620/speed2)):
                    time.sleep((2700.0/speed2/int(6620/speed2))/(len(wave2) - 1))
294                 #gate / reset
                    if c.gatein(2) or c.pad(7):
296                     k = 0
                        l = 0
298                     reset2 = True
                        break
300             if reset2:
                    reset2 = False
302                 break

304 def ledReset():
    c = Controls()
306 c.led(1, 0)
    c.led(2, 0)
308 c.led(3, 0)
    c.padled(1, 'r', 0)
310 c.padled(1, 'g', 0)
    c.padled(1, 'b', 0)
312 c.padled(2, 'r', 0)
    c.padled(2, 'g', 0)
314 c.padled(2, 'b', 0)
    c.padled(3, 'r', 0)
316 c.padled(3, 'g', 0)
    c.padled(3, 'b', 0)
318 c.padled(4, 'r', 0)
    c.padled(4, 'g', 0)
320 c.padled(4, 'b', 0)
    c.padled(5, 'r', 0)
322 c.padled(5, 'g', 0)
    c.padled(5, 'b', 0)
324 c.padled(6, 'r', 0)
    c.padled(6, 'g', 0)
326 c.padled(6, 'b', 0)
    c.padled(7, 'r', 0)
328 c.padled(7, 'g', 0)
    c.padled(7, 'b', 0)
330 c.padled(8, 'r', 0)
    c.padled(8, 'g', 0)
332 c.padled(8, 'b', 0)

334 #if __name__ == '__main__':
    # Global variables
336 outrange1 = 1.0
    outrange2 = 1.0
338 # speed1 = 0
    # speed2 = 0
340 # resolution1 = 0
    # resolution2 = 0
342 # shape1 = 0
```

```
# shape2 = 0
# lfospeed1 = 0
# lfospeed2 = 0
# lfoamount1 = 0
# lfoamount2 = 0
# start Controls()
c = Controls()

ledReset()
c.padled(1, 'b', 1)
c.padled(7, 'b', 1)
c.padled(4, 'b', 1)
c.padled(6, 'b', 1)

# init and start Threads
lockMe = threading.Lock()
#lockMe2 = threading.Lock()
#Tread = threading.Thread(target=read, args=("Thread-1", ))
TwriteCV1 = threading.Thread(target=writeCV1, args=("Thread-2", ))
TwriteCV2 = threading.Thread(target=writeCV2, args=("Thread-3", ))
#Twrite = threading.Thread(target=write, args=("Thread-4", ))
#Tread.start()
TwriteCV1.start()
TwriteCV2.start()
```

eupyDualSlopeLFO.py

6.1.2 eupyControls.py

```
#!/usr/bin/env python

#
    ##############################################################################################################

# class to fetch the actual values of the EuroPython User Interface
# 10 Potentiometers (ADC), 4 Switches (GPIO), 8 Buttons (GPIO), 2 GateIn/2 GateOut
    (GPIO), 2 CVIn (ADC), 2 CVOut (DAC), 2 AudioIn, 2 AudioOut
# with Olinuxino A20 LIME 2 development board
# by Fabian Pfoser September2016
#
#
    ##############################################################################################################


import os
import sys
# from time import sleep
from pyA20Lime2.gpio import gpio
from pyA20Lime2.gpio import port
from eupyAD5667R import eupyAD5667R
from eupyADS1115 import eupyADS1115


if not os.getegid() == 0:
   sys.exit('Script must be run as root')

__author__ = "Fabian Pfoser"
__copyright__ = "Copyright 2016"
__credits__ = ["Fabian Pfoser"]
__license__ = "GPL"
__version__ = "1.0"
__maintainer__ = __author__
__email__ = "f.pfoser@student.tugraz.at"

##############################################
# INPUTS
##############################################
gpioled = port.PH2

# GPIO2
pad1 = port.PE0
pad2 = port.PE1
pad3 = port.PE2
pad4 = port.PE3
pad5 = port.PE4
pad6 = port.PE5
pad7 = port.PE6
pad8 = port.PE7
sw1 = port.PE8
sw2 = port.PE10
sw3 = port.PE9
sw4 = port.PE11
gatein1 = port.PI21
gatein2 = port.PI20

##############################################
# OUTPUTS
##############################################

# GPIO2
led1 = port.PI0
led2 = port.PI1
led3 = port.PI2
pad2b = port.PI16
# GPIO3
pad1r = port.PB6
pad1g = port.PB5
pad1b = port.PB4
pad2r = port.PB7
pad2g = port.PB8
```

```
#pad2b = port.PB9 #not working gpio :-(
pad3r = port.PB10
pad3g = port.PB11
pad3b = port.PB12
pad4r = port.PB13
pad4g = port.PB14
pad4b = port.PB15
pad5r = port.PB16
pad5g = port.PB17
pad5b = port.PH24
pad6r = port.PH25
pad6g = port.PH26
pad6b = port.PH27
pad7r = port.PH23
pad7g = port.PH22
pad7b = port.PH21
pad8r = port.PH20
pad8g = port.PH19
pad8b = port.PH18
gateout1 = port.PH17
gateout2 = port.PH16


# GPIO test section
gpio.init() # Initialize module. Always called first

gpio.setcfg(gpioled, gpio.OUTPUT)

gpio.setcfg(pad1, gpio.INPUT)   # Configure pad1 as input
gpio.setcfg(pad2, gpio.INPUT)
gpio.setcfg(pad3, gpio.INPUT)
gpio.setcfg(pad4, gpio.INPUT)
gpio.setcfg(pad5, gpio.INPUT)
gpio.setcfg(pad6, gpio.INPUT)
gpio.setcfg(pad7, gpio.INPUT)
gpio.setcfg(pad8, gpio.INPUT)
gpio.setcfg(sw1, gpio.INPUT)
gpio.setcfg(sw2, gpio.INPUT)
gpio.setcfg(sw3, gpio.INPUT)
gpio.setcfg(sw4, gpio.INPUT)
gpio.setcfg(gatein1, gpio.INPUT)
gpio.setcfg(gatein2, gpio.INPUT)

gpio.setcfg(led1, gpio.OUTPUT)  # Configure LED1 as output
gpio.setcfg(led2, gpio.OUTPUT)
gpio.setcfg(led3, gpio.OUTPUT)
gpio.setcfg(pad1r, gpio.OUTPUT)
gpio.setcfg(pad1g, gpio.OUTPUT)
gpio.setcfg(pad1b, gpio.OUTPUT)
gpio.setcfg(pad2r, gpio.OUTPUT)
gpio.setcfg(pad2g, gpio.OUTPUT)
gpio.setcfg(pad2b, gpio.OUTPUT)
gpio.setcfg(pad3r, gpio.OUTPUT)
gpio.setcfg(pad3g, gpio.OUTPUT)
gpio.setcfg(pad3b, gpio.OUTPUT)
gpio.setcfg(pad4r, gpio.OUTPUT)
gpio.setcfg(pad4g, gpio.OUTPUT)
gpio.setcfg(pad4b, gpio.OUTPUT)
gpio.setcfg(pad5r, gpio.OUTPUT)
gpio.setcfg(pad5g, gpio.OUTPUT)
gpio.setcfg(pad5b, gpio.OUTPUT)
gpio.setcfg(pad6r, gpio.OUTPUT)
gpio.setcfg(pad6g, gpio.OUTPUT)
gpio.setcfg(pad6b, gpio.OUTPUT)
gpio.setcfg(pad7r, gpio.OUTPUT)
gpio.setcfg(pad7g, gpio.OUTPUT)
gpio.setcfg(pad7b, gpio.OUTPUT)
gpio.setcfg(pad8r, gpio.OUTPUT)
gpio.setcfg(pad8g, gpio.OUTPUT)
gpio.setcfg(pad8b, gpio.OUTPUT)
gpio.setcfg(gateout1, gpio.OUTPUT)
gpio.setcfg(gateout2, gpio.OUTPUT)
```

```
#Clear pullups/pulldowns
gpio.pullup(pad1, 0)
gpio.pullup(pad2, 0)
gpio.pullup(pad3, 0)
gpio.pullup(pad4, 0)
gpio.pullup(pad5, 0)
gpio.pullup(pad6, 0)
gpio.pullup(pad7, 0)
gpio.pullup(pad8, 0)
gpio.pullup(sw1, 0)
gpio.pullup(sw2, 0)
gpio.pullup(sw3, 0)
gpio.pullup(sw4, 0)

# Enable pull-downs
gpio.pullup(pad1, gpio.PULLDOWN)
gpio.pullup(pad2, gpio.PULLDOWN)
gpio.pullup(pad3, gpio.PULLDOWN)
gpio.pullup(pad4, gpio.PULLDOWN)
gpio.pullup(pad5, gpio.PULLDOWN)
gpio.pullup(pad6, gpio.PULLDOWN)
gpio.pullup(pad7, gpio.PULLDOWN)
gpio.pullup(pad8, gpio.PULLDOWN)
gpio.pullup(sw1, gpio.PULLDOWN)
gpio.pullup(sw2, gpio.PULLDOWN)
gpio.pullup(sw3, gpio.PULLDOWN)
gpio.pullup(sw4, gpio.PULLDOWN)


class Controls(object):

    # return boolean value of choosen pad
    def pad(self, select=1):
        if (select < 1) or (select > 8):
            print "Only eight pads on ersi, please select a pad from 1 to 8"
        else:
            if select == 1:
                return gpio.input(pad1)
            elif select == 2:
                return gpio.input(pad2)
            elif select == 3:
                return gpio.input(pad3)
            elif select == 4:
                return gpio.input(pad4)
            elif select == 5:
                return gpio.input(pad5)
            elif select == 6:
                return gpio.input(pad6)
            elif select == 7:
                return gpio.input(pad7)
            elif select == 8:
                return gpio.input(pad8)

    # return boolean value of choosen switch
    def switch(self, select=1):
        if (select < 1) or (select > 4):
            print "Only four switches on ersi, please select a switch from 1 to 4"
        else:
            if select == 1:
                return gpio.input(sw1)
            elif select == 2:
                return gpio.input(sw2)
            elif select == 3:
                return gpio.input(sw3)
            elif select == 4:
                return gpio.input(sw4)

    # switch led on(1) or off(0)
    def led(self, select, onoff=0):
        if (select < 1) or (select > 3):
            print "Only three leds on ersi, please select a led from 1 to 3"
        else:
            if select == 1:
```

```
                gpio.output(led1, onoff)
            elif select == 2:
                gpio.output(led2, onoff)
            elif select == 3:
                gpio.output(led3, onoff)

    # switch leds of pads on(1) or off(0)
    def padled(self, pad, led, onoff=0):
        if (pad < 1) or (pad > 8):
            print "Only eight pads on ersi, please select a pad from 1 to 8"
        if (led != 'r') and (led != 'g') and (led != 'b'):
            print "Led have to be 'r', 'g', or 'b'"
        else:
            if (pad == 1) and (led == 'r'):
                gpio.output(pad1r, onoff)
            elif (pad == 2) and (led == 'r'):
                gpio.output(pad2r, onoff)
            elif (pad == 3) and (led == 'r'):
                gpio.output(pad3r, onoff)
            elif (pad == 4) and (led == 'r'):
                gpio.output(pad4r, onoff)
            elif (pad == 5) and (led == 'r'):
                gpio.output(pad5r, onoff)
            elif (pad == 6) and (led == 'r'):
                gpio.output(pad6r, onoff)
            elif (pad == 7) and (led == 'r'):
                gpio.output(pad7r, onoff)
            elif (pad == 8) and (led == 'r'):
                gpio.output(pad8r, onoff)
            elif (pad == 1) and (led == 'g'):
                gpio.output(pad1g, onoff)
            elif (pad == 2) and (led == 'g'):
                gpio.output(pad2g, onoff)
            elif (pad == 3) and (led == 'g'):
                gpio.output(pad3g, onoff)
            elif (pad == 4) and (led == 'g'):
                gpio.output(pad4g, onoff)
            elif (pad == 5) and (led == 'g'):
                gpio.output(pad5g, onoff)
            elif (pad == 6) and (led == 'g'):
                gpio.output(pad6g, onoff)
            elif (pad == 7) and (led == 'g'):
                gpio.output(pad7g, onoff)
            elif (pad == 8) and (led == 'g'):
                gpio.output(pad8g, onoff)
            elif (pad == 1) and (led == 'b'):
                gpio.output(pad1b, onoff)
            elif (pad == 2) and (led == 'b'):
                gpio.output(pad2b, onoff)
            elif (pad == 3) and (led == 'b'):
                gpio.output(pad3b, onoff)
            elif (pad == 4) and (led == 'b'):
                gpio.output(pad4b, onoff)
            elif (pad == 5) and (led == 'b'):
                gpio.output(pad5b, onoff)
            elif (pad == 6) and (led == 'b'):
                gpio.output(pad6b, onoff)
            elif (pad == 7) and (led == 'b'):
                gpio.output(pad7b, onoff)
            elif (pad == 8) and (led == 'b'):
                gpio.output(pad8b, onoff)

    # return boolean value if there knocks a fuckin gatein
    # INVERSE
    def gatein(self, select):
        if (select < 1) or (select > 2):
            print "Only two gateIn-jacks on ersi, please select jack 1 or 2"
        else:
            if select == 1:
                return not gpio.input(gatein1)
            elif select == 2:
                return not gpio.input(gatein2)
```

```
286     # switch gateout, sends gate or triggers
        # INVERSE
288     def gateout(self, select, onoff=0):
            if (select < 1) or (select > 2):
290             print "Only two gateOut-jacks on ersi, please select jack 1 or 2"
            else:
292             if select == 1:
                    gpio.output(gateout1, onoff)
294             elif select == 2:
                    gpio.output(gateout2, onoff)
296
298     ##################################################
        # DAC things, 0-48300
300     #      0  --> 0V
        # 48300  --> 3,3V --> 8V
302     ##################################################
        def cvout(self, select=1, value=0):
304         dac = eupyAD5667R()

306         if (select < 1) or (select > 12):
                print "Only two CVOut-jacks on ersi, please select jack 1, 2 or 12 for
    both channels"
308         else:
                if select == 1:
310                 channel = 'a'
                elif select == 2:
312                 channel = 'b'
                elif select == 12:
314                 channel = 'ab'

316             dac.write_dac("GND", 0, channel, value)

318     ##################################################
        # ADC things, return a signed integer
320     # CVins:
        # -5V ->   3,3V   -->
322     #  0V -> 2,727V   -->
        # 12V ->     0V   -->
324     #
        # POTS:
326     # 0V (full CCW)   -->   26480
        # 3.3V (full CW)  -->       0
328     ##################################################
        def cvin(self, select=1):
330         if (select < 1) or (select > 2):
                print "Only two CV inputs on ersi, please select 1 or 2"
332         else:
                if select == 1:
334                 currentcv = 0
                elif select == 2:
336                 currentcv = 1
                adc = eupyADS1115()
338             return adc.read_adc("SDA", 0, currentcv, gain=1, data_rate=128)

340     def pot(self, select=1):
            if (select < 1) or (select > 10):
342             print "Only ten potentiometers on ersi, please select one from 1 to 10"
            else:
344             if select == 1:
                    currentadc = "GND"
346                 currentch = 0
                elif select == 2:
348                 currentadc = "GND"
                    currentch = 1
350             elif select == 3:
                    currentadc = "GND"
352                 currentch = 2
                elif select == 4:
354                 currentadc = "GND"
                    currentch = 3
356             elif select == 5:
                    currentadc = "VDD"
```

```
358                 currentch = 0
            elif select == 6:
360                 currentadc = "VDD"
                    currentch = 1
362             elif select == 7:
                    currentadc = "VDD"
364                 currentch = 2
            elif select == 8:
366                 currentadc = "VDD"
                    currentch = 3
368             elif select == 9:
                    currentadc = "SDA"
370                 currentch = 2
            elif select == 10:
372                 currentadc = "SDA"
                    currentch = 3
374         adc = eupyADS1115()
            return adc.read_adc(currentadc, 0, currentch, gain=1, data_rate=128)
```

eupyControls.py

6.1.3 eupyADS1115.py

```
#
    ##########################################################################################

# ORIGINAL FILE by: (Author:) Tony DiCola
# Copyright (c) 2016 Adafruit Industries
#
# modified to use ADS1115 on Olinuxiono A20 Lime 2 by Fabian Pfoser September 2016
#
    ##########################################################################################

#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
import time
import os
import sys
from pyA20Lime2 import i2c

if not os.getegid() == 0:
   sys.exit('Script must be run as root')

# I2C Bus on Olimex
I2CBUS_0                  = "/dev/i2c-0"
I2CBUS_1                  = "/dev/i2c-1"
I2CBUS_2                  = "/dev/i2c-2"

# Register and other configuration values:
ADS1115_ADDR_GND                = 0x48
ADS1115_ADDR_VDD                = 0x49
ADS1115_ADDR_SDA                = 0x4A
ADS1115_ADDR_SCL                = 0x4B

ADS1115_POINTER_CONVERSION      = 0x00
ADS1115_POINTER_CONFIG          = 0x01
ADS1115_POINTER_LOW_THRESHOLD   = 0x02
ADS1115_POINTER_HIGH_THRESHOLD  = 0x03
ADS1115_CONFIG_OS_SINGLE        = 0x8000
ADS1115_CONFIG_MUX_OFFSET       = 12
# Maping of gain values to config register values.
ADS1115_CONFIG_GAIN = {
    2/3: 0x0000,
    1:   0x0200,
    2:   0x0400,
    4:   0x0600,
    8:   0x0800,
    16:  0x0A00
}
ADS1115_CONFIG_MODE_CONTINUOUS  = 0x0000
ADS1115_CONFIG_MODE_SINGLE      = 0x0100
# Mapping of data/sample rate to config register values for ADS1115 (slower).
ADS1115_CONFIG_DR = {
    8:    0x0000,
    16:   0x0020,
    32:   0x0040,
    64:   0x0060,
    128:  0x0080,
    250:  0x00A0,
    475:  0x00C0,
    860:  0x00E0
}
ADS1115_CONFIG_COMP_WINDOW      = 0x0010
ADS1115_CONFIG_COMP_ACTIVE_HIGH = 0x0008
ADS1115_CONFIG_COMP_LATCHING    = 0x0004
```

```
ADS1115_CONFIG_COMP_QUE = {
    1: 0x0000,
    2: 0x0001,
    4: 0x0002
}
ADS1115_CONFIG_COMP_QUE_DISABLE = 0x0003


class eupyADS1115(object):

    def _data_rate_default(self):
        # Default from datasheet page 16, config register DR bit default.
        return 128

    # def _data_rate_config(self, data_rate):
    #     """Subclasses should override this function and return a 16-bit value
    #     that can be OR'ed with the config register to set the specified
    #     data rate.  If a value of None is specified then a default data_rate
    #     setting should be returned.  If an invalid or unsupported data_rate is
    #     provided then an exception should be thrown.
    #     """
    #     raise NotImplementedError('Subclass must implement _data_rate_config
    function!')

    def _data_rate_config(self, data_rate):
        if data_rate not in ADS1115_CONFIG_DR:
            raise ValueError('Data rate must be one of: 8, 16, 32, 64, 128, 250,
    475, 860')
        return ADS1115_CONFIG_DR[data_rate]

    # def _conversion_value(self, low, high):
    #     """Subclasses should override this function that takes the low and high
    #     byte of a conversion result and returns a signed integer value.
    #     """
    #     raise NotImplementedError('Subclass must implement _conversion_value
    function!')

    def _conversion_value(self, low, high):
        # Convert to 16-bit signed value.
        value = ((high & 0xFF) << 8) | (low & 0xFF)
        # Check for sign bit and turn into a negative value if set.
        if value & 0x8000 != 0:
            value -= 1 << 16

        #scale all pots/cv in range 1-26480
        if value <= 0:
            value = 1
        if value > 26480:
            value = 26480
        value = (-1)*(value - 26481)
        return value


    def _read(self, mux, dacaddr, gain, data_rate, mode):
        """Perform an ADC read with the provided mux, gain, data_rate, and mode
        values.  Returns the signed integer result of the read.
        """

        config = ADS1115_CONFIG_OS_SINGLE  # Go out of power-down mode for
    conversion.
        # Specify mux value.
        config |= (mux & 0x07) << ADS1115_CONFIG_MUX_OFFSET
        # Validate the passed in gain and then set it in the config.
        if gain not in ADS1115_CONFIG_GAIN:
            raise ValueError('Gain must be one of: 2/3, 1, 2, 4, 8, 16')
        config |= ADS1115_CONFIG_GAIN[gain]
        # Set the mode (continuous or single shot).
        config |= mode
        # Get the default data rate if none is specified (default differs between
        # ADS1015 and ADS1115).
        if data_rate is None:
            data_rate = self._data_rate_default()
        # Set the data rate (this is controlled by the subclass as it differs
```

```
137         # between ADS1015 and ADS1115).
            config |= self._data_rate_config(data_rate)
139         config |= ADS1115_CONFIG_COMP_QUE_DISABLE  # Disble comparator mode.
            # Send the config value to start the ADC conversion.
141         # Explicitly break the 16-bit value down to a big endian pair of bytes.
            i2c.write([((dacaddr << 8) | ADS1115_POINTER_CONFIG), (config >> 8) & 0xFF,
     config & 0xFF])
143         # Wait for the ADC sample to finish based on the sample rate plus a
            # small offset to be sure (0.1 millisecond).
145         time.sleep(1.0/data_rate+0.0001)

147         # Retrieve the result.
            i2c.write([ADS1115_POINTER_CONVERSION])
149         result = i2c.read(2)
            i2c.close()
151
            return self._conversion_value(result[1], result[0])
153


155
    def read_adc(self, address="GND", i2cbus=0, channel=0, gain=1, data_rate=None):
157         """Read a single ADC channel and return the ADC value as a signed integer
            result.  Channel must be a value within 0-3.
159         """
            validbus = False
161         validaddress = False

163         if i2cbus == 0:
                openbus = I2CBUS_0
165             validbus = True
            if i2cbus == 1:
167             openbus = I2CBUS_1
                validbus = True
169         if i2cbus == 2:
                openbus = I2CBUS_2
171             validbus = True
            elif validbus == False:
173             print "I2C-Bus have to be '0', '1' or '2'!"
                exit()
175
            if address == "GND":
177             dacaddr = ADS1115_ADDR_GND
                validaddress = True
179         if address == "VDD":
                dacaddr = ADS1115_ADDR_VDD
181             validaddress = True
            if address == "SDA":
183             dacaddr = ADS1115_ADDR_SDA
                validaddress = True
185         if address == "SCL":
                dacaddr = ADS1115_ADDR_SCL
187             validaddress = True

189         if validaddress == False:
                print "Adress have to be 'GND', 'VDD' ,'SDA' or 'SCL'!"
191             exit()

193         i2c.init(openbus)  #Initialize module to use /dev/i2c-0
            i2c.open(dacaddr)
195
            assert 0 <= channel <= 3, 'Channel must be a value within 0-3!'
197         # Perform a single shot read and set the mux value to the channel plus
            # the highest bit (bit 3) set.
199         return self._read(channel + 0x04, dacaddr, gain, data_rate,
    ADS1115_CONFIG_MODE_SINGLE)

201
    def start_adc(self, address="GND", i2cbus=0, channel=0, gain=1, data_rate=None)
    :
203         """Start continuous ADC conversions on the specified channel (0-3). Will
            return an initial conversion result, then call the get_last_result()
205         function to read the most recent conversion result. Call stop_adc() to
            stop conversions.
```

```
207         """
209         validbus = False
            validaddress = False
211
            if i2cbus == 0:
213             openbus = I2CBUS_0
                validbus = True
215         if i2cbus == 1:
                openbus = I2CBUS_1
217             validbus = True
            if i2cbus == 2:
219             openbus = I2CBUS_2
                validbus = True
221         if validbus == False:
                print "I2C-Bus have to be '0', '1' or '2'!"
223             exit()

225         if address == "GND":
                dacaddr = ADS1115_ADDR_GND
227             validaddress = True
            if address == "VDD":
229             dacaddr = ADS1115_ADDR_VDD
                validaddress = True
231         if address == "SDA":
                dacaddr = ADS1115_ADDR_SDA
233             validaddress = True
            if address == "SCL":
235             dacaddr = ADS1115_ADDR_SCL
                validaddress = True
237         if validaddress == False:
                print "Adress have to be 'GND', 'VDD' ,'SDA' or 'SCL'!"
239             exit()

241         i2c.init(openbus)  #Initialize module to use /dev/i2c-0
            i2c.open(dacaddr)  # open ADC (IC3) Pot1-4, ADDR=GND=0b1001000, 7Bit Slave
    Adress
243
            assert 0 <= channel <= 3, 'Channel must be a value within 0-3!'
245
            # Start continuous reads and set the mux value to the channel plus
247         # the highest bit (bit 3) set.
            return self._read(channel + 0x04, gain, data_rate,
    ADS1115_CONFIG_MODE_CONTINUOUS)
249

251     def stop_adc(self):
            """Stop all continuous ADC conversions (either normal or difference mode).
253         """
            # Set the config register to its default value of 0x8583 to stop
255         # continuous conversions.
            config = 0x8583
257         i2c.write([ADS1115_POINTER_CONFIG])
            i2c.write([(config >> 8) & 0xFF, config & 0xFF])
259         i2c.close()

261     def get_last_result(self):
            """Read the last conversion result when in continuous conversion mode.
263         Will return a signed integer value.
            """
265         # Retrieve the conversion register value, convert to a signed int, and
            # return it.
267
            i2c.write([ADS1115_POINTER_CONVERSION])
269         result = i2c.read(2)
            return self._conversion_value(result[1], result[0])
271

273     def read_adc_difference(self, differential, gain=1, data_rate=None):
            """Read the difference between two ADC channels and return the ADC value
275         as a signed integer result.  Differential must be one of:
              - 0 = Channel 0 minus channel 1
277           - 1 = Channel 0 minus channel 3
```

```
          - 2 = Channel 1 minus channel 3
          - 3 = Channel 2 minus channel 3
        """
        assert 0 <= differential <= 3, 'Differential must be a value within 0-3!'
        # Perform a single shot read using the provided differential value
        # as the mux value (which will enable differential mode).
        return self._read(differential, gain, data_rate, ADS1115_CONFIG_MODE_SINGLE
    )


    def _read_comparator(self, mux, gain, data_rate, mode, high_threshold,
                         low_threshold, active_low, traditional, latching,
                         num_readings):
        """Perform an ADC read with the provided mux, gain, data_rate, and mode
        values and with the comparator enabled as specified.  Returns the signed
        integer result of the read.
        """
        assert num_readings == 1 or num_readings == 2 or num_readings == 4, 'Num
    readings must be 1, 2, or 4!'
        # Set high and low threshold register values.
        i2c.write(ADS1115_POINTER_HIGH_THRESHOLD, [(high_threshold >> 8) & 0xFF,
    high_threshold & 0xFF])
        i2c.write(ADS1115_POINTER_LOW_THRESHOLD, [(low_threshold >> 8) & 0xFF,
    low_threshold & 0xFF])
        # Now build up the appropriate config register value.
        config = ADS1115_CONFIG_OS_SINGLE  # Go out of power-down mode for
    conversion.
        # Specify mux value.
        config |= (mux & 0x07) << ADS1115_CONFIG_MUX_OFFSET
        # Validate the passed in gain and then set it in the config.
        if gain not in ADS1115_CONFIG_GAIN:
            raise ValueError('Gain must be one of: 2/3, 1, 2, 4, 8, 16')
        config |= ADS1115_CONFIG_GAIN[gain]
        # Set the mode (continuous or single shot).
        config |= mode
        # Get the default data rate if none is specified (default differs between
        # ADS1015 and ADS1115).
        if data_rate is None:
            data_rate = self._data_rate_default()
        # Set the data rate (this is controlled by the subclass as it differs
        # between ADS1015 and ADS1115).
        config |= self._data_rate_config(data_rate)
        # Enable window mode if required.
        if not traditional:
            config |= ADS1115_CONFIG_COMP_WINDOW
        # Enable active high mode if required.
        if not active_low:
            config |= ADS1115_CONFIG_COMP_ACTIVE_HIGH
        # Enable latching mode if required.
        if latching:
            config |= ADS1115_CONFIG_COMP_LATCHING
        # Set number of comparator hits before alerting.
        config |= ADS1115_CONFIG_COMP_QUE[num_readings]
        # Send the config value to start the ADC conversion.
        # Explicitly break the 16-bit value down to a big endian pair of bytes.
        i2c.write([(config >> 8) & 0xFF, config & 0xFF])
        # Wait for the ADC sample to finish based on the sample rate plus a
        # small offset to be sure (0.1 millisecond).
        time.sleep(1.0/data_rate+0.0001)
        # Retrieve the result.
        result = i2c.read(2)
        return self._conversion_value(result[1], result[0])


    def start_adc_difference(self, differential, gain=1, data_rate=None):
        """Start continuous ADC conversions between two ADC channels. Differential
        must be one of:
          - 0 = Channel 0 minus channel 1
          - 1 = Channel 0 minus channel 3
          - 2 = Channel 1 minus channel 3
          - 3 = Channel 2 minus channel 3
        Will return an initial conversion result, then call the get_last_result()
        function continuously to read the most recent conversion result.  Call
```

```
        stop_adc() to stop conversions.
347     """
        assert 0 <= differential <= 3, 'Differential must be a value within 0-3!'
349     # Perform a single shot read using the provided differential value
        # as the mux value (which will enable differential mode).
351     return self._read(differential, gain, data_rate,
    ADS1115_CONFIG_MODE_CONTINUOUS)


353
    def start_adc_comparator(self, channel, high_threshold, low_threshold,
355                          gain=1, data_rate=None, active_low=True,
                             traditional=True, latching=False, num_readings=1):
357     """Start continuous ADC conversions on the specified channel (0-3) with
        the comparator enabled.  When enabled the comparator to will check if
359     the ADC value is within the high_threshold & low_threshold value (both
        should be signed 16-bit integers) and trigger the ALERT pin.  The
361     behavior can be controlled by the following parameters:
          - active_low: Boolean that indicates if ALERT is pulled low or high
363                     when active/triggered.  Default is true, active low.
          - traditional: Boolean that indicates if the comparator is in traditional
365                       mode where it fires when the value is within the threshold
    ,
                       or in window mode where it fires when the value is
    _outside_
367                    the threshold range.  Default is true, traditional mode.
          - latching: Boolean that indicates if the alert should be held until
369                   get_last_result() is called to read the value and clear
                      the alert.  Default is false, non-latching.
371       - num_readings: The number of readings that match the comparator before
                        triggering the alert.  Can be 1, 2, or 4.  Default is 1.
373     Will return an initial conversion result, then call the get_last_result()
        function continuously to read the most recent conversion result.  Call
375     stop_adc() to stop conversions.
        """
377     assert 0 <= channel <= 3, 'Channel must be a value within 0-3!'
        # Start continuous reads with comparator and set the mux value to the
379     # channel plus the highest bit (bit 3) set.
        return self._read_comparator(channel + 0x04, gain, data_rate,
381                                  ADS1115_CONFIG_MODE_CONTINUOUS,
                                     high_threshold, low_threshold, active_low,
383                                  traditional, latching, num_readings)

385 def start_adc_difference_comparator(self, differential, high_threshold,
    low_threshold,
                                     gain=1, data_rate=None, active_low=True,
387                                  traditional=True, latching=False,
    num_readings=1):
        """Start continuous ADC conversions between two channels with
389     the comparator enabled.  See start_adc_difference for valid differential
        parameter values and their meaning.  When enabled the comparator to will
391     check if the ADC value is within the high_threshold & low_threshold value
        (both should be signed 16-bit integers) and trigger the ALERT pin.  The
393     behavior can be controlled by the following parameters:
          - active_low: Boolean that indicates if ALERT is pulled low or high
395                     when active/triggered.  Default is true, active low.
          - traditional: Boolean that indicates if the comparator is in traditional
397                       mode where it fires when the value is within the threshold
    ,
                       or in window mode where it fires when the value is
    _outside_
399                    the threshold range.  Default is true, traditional mode.
          - latching: Boolean that indicates if the alert should be held until
401                   get_last_result() is called to read the value and clear
                      the alert.  Default is false, non-latching.
403       - num_readings: The number of readings that match the comparator before
                        triggering the alert.  Can be 1, 2, or 4.  Default is 1.
405     Will return an initial conversion result, then call the get_last_result()
        function continuously to read the most recent conversion result.  Call
407     stop_adc() to stop conversions.
        """
409     assert 0 <= differential <= 3, 'Differential must be a value within 0-3!'
        # Start continuous reads with comparator and set the mux value to the
411     # channel plus the highest bit (bit 3) set.
```

```
        return self._read_comparator(differential, gain, data_rate,
413                                      ADS1115_CONFIG_MODE_CONTINUOUS,
                                         high_threshold, low_threshold, active_low,
415                                      traditional, latching, num_readings)
```

eupyADS1115.py

6.1.4 eupyAD5667R.py

```
#!/usr/bin/env python

#
    ##############################################################################################################

# class for using AD5667R Digital-Analog-Converter with Olinuxino A20 LIME 2
    development board
# by Fabian Pfoser September2016
#
# LDAC Function not implemented yet!
#
    ##############################################################################################################


import os
import sys
from pyA20Lime2 import i2c

if not os.getegid() == 0:
   sys.exit('Script must be run as root')

__author__ = "Fabian Pfoser"
__copyright__ = "Copyright 2016"
__credits__ = ["Fabian Pfoser"]
__license__ = "GPL"
__version__ = "1.0"
__maintainer__ = __author__
__email__ = "f.pfoser@student.tugraz.at"


# I2C Bus on Olimex
I2CBUS_0                    = "/dev/i2c-0"
I2CBUS_1                    = "/dev/i2c-1"
I2CBUS_2                    = "/dev/i2c-2"

# AD5667R 7bit-I2C-Slave Adresses
AD5667R_ADDR_GND            = 0b0001111
AD5667R_ADDR_VDD            = 0b0001100
AD5667R_ADDR_NC             = 0b0001110

# AD5667R 1bit-Read/Write
AD5667R_WRITE               = 0b00000000
AD5667R_READ                = 0b00000001

# AD5667R 8bit-COMMAND Register (DB23(bit7) is reserved, should always be 0!
# Bit 6
AD5667R_MULTBYTE_ON         = 0b01000000
# Bit 5:3
AD5667R_WIR                 = 0b00000000    # Write to input register n (n=DAC)
AD5667R_UDACR               = 0b00001000    # Update DAC register n
AD5667R_WIRUA               = 0b00010000    # Write to input register n, update all
AD5667R_WUDAC               = 0b00011000    # Write to and update DAC channel n
AD5667R_PWRUPDWN            = 0b00100000    # Power up/Power down
AD5667R_RESET               = 0b00101000    # RESET
AD5667R_LDAC                = 0b00110000    # LDAC register setup
AD5667R_INTREF              = 0b00111000    # Internal reference setup (on/off)
# Bit 1:0
AD5667R_DACA                = 0b00000000
AD5667R_DACB                = 0b00000001
AD5667R_DACAB               = 0b00000111


class eupyAD5667R(object):

    def write_dac(self, address = "GND", i2cbus = 0, channel= "a", value = 0):
        # 24bit Input Shift Register
        # 8bit-command, 8bit-MSB, 8bit-LSB

        validbus = False
        validaddress = False

```

```
        if i2cbus == 0:
68          openbus = I2CBUS_0
            validbus = True
70      elif i2cbus == 1:
            openbus = I2CBUS_1
72          validbus = True
        elif i2cbus == 2:
74          openbus = I2CBUS_2
            validbus = True
76      if validbus == False:
            print "I2C-Bus have to be '0', '1' or '2'!"
78          exit()

80      if address == "GND":
            dacaddr = AD5667R_ADDR_GND
82          validaddress = True
        elif address == "VDD":
84          dacaddr = AD5667R_ADDR_VDD
            validaddress = True
86      elif address == "NC":
            dacaddr = AD5667R_ADDR_NC
88          validaddress = True
        if validaddress == False:
90          print "Adress have to be 'GND', 'VDD' or 'NC'!"
            exit()
92
        #print channel
94      validchannel = False

96      if channel == "a":
            dacout = AD5667R_DACA
98          validchannel = True
        elif channel == "b":
100         dacout = AD5667R_DACB
            validchannel = True
102     elif channel == "ab":
            dacout = AD5667R_DACAB
104         validchannel = True
        if validchannel == False:
106         print "Channel have to be 'a', 'b' or 'ab' for both channels!"
            exit()
108
        i2c.init(openbus)  #Initialize module to use /dev/i2c-0
110     i2c.open(dacaddr)

112     # init before conversion / datasheet page 27
        #set PowerUP both DACs
114     i2c.write([AD5667R_PWRUPDWN, 0b00000000, dacout])

116     #set internal reference ON
        i2c.write([AD5667R_INTREF, 0b00000000, 0b00000001])
118     #i2c.close()

120     #limit output to 8V, full range: 8,28V
        if value > 48300:
122         value = 48300
        if value <= 0:
124         value = 1

126     Highint = (value >> 8)
        Lowint = value
128     #cvout, write data, Bit 19:17 DAC Select DAC1, 00 000 000 +16DATA-Bits
        i2c.write([(AD5667R_WUDAC | dacout), Highint, Lowint])
130     #close i2c connection
        i2c.close()
132
    def start_dac(self, address = "GND", i2cbus = 0):
134
        validbus = False
136     validaddress = False

138     if i2cbus == 0:
            openbus = I2CBUS_0
```

```
140             validbus = True
        elif i2cbus == 1:
142             openbus = I2CBUS_1
                validbus = True
144         elif i2cbus == 2:
                openbus = I2CBUS_2
146             validbus = True
            if validbus == False:
148             print "I2C-Bus have to be '0', '1' or '2'!"
                exit()
150
            if address == "GND":
152             dacaddr = AD5667R_ADDR_GND
                validaddress = True
154         elif address == "VDD":
                dacaddr = AD5667R_ADDR_VDD
156             validaddress = True
            elif address == "NC":
158             dacaddr = AD5667R_ADDR_NC
                validaddress = True
160         if validaddress == False:
                print "Adress have to be 'GND', 'VDD' or 'NC'!"
162             exit()

164         i2c.init(openbus)  #Initialize module to use /dev/i2c-0
            i2c.open(dacaddr)
166
            # init before conversion / datasheet page 27
168         #set software reset
            i2c.write([0b00101000, 0b00000000, 0b00000001])
170         #set PowerUP both DACs
            i2c.write([AD5667R_PWRUPDWN, 0b00000000, 0b00000011])
172         #i2c.write([0b00011110, 0b00100000, 0b00000000, 0b00000010])
            #set internal reference ON
174         i2c.write([AD5667R_INTREF, 0b00000000, 0b00000001])

176     def stop_dac(self):
            # set both outputs to 0V:
178         i2c.write([AD5667R_WUDAC | AD5667R_DACAB , 0b00000000, 0b00000000])
            #close i2c connection
180         i2c.close()
```

eupyAD5667R.py

6.2 Schaltplan