Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások

Szintetizátor- és effektmodulok megvalósítása LV2 környezetben

Szakdolgozat

Készítette
Nagy Jenő

Konzulens
dr. Rucz Péter

2018. december 7.

Tartalomjegyzék

Kivonat

A technológia fejlődésének és a digitális audiotartalmak terjedésének köszönhetően sokakban merül fel igény az ilyen tartalom előállítására, szerkesztésére. Ezekre az egyik legalkalmasabb szoftverek csoportja a Digital Audio Workstion (röviden DAW) programok. Nagyon sok kereskedelmi szoftver létezik, ugyanakkor léteznek ingyenes, nyílt alternatívák is. Sokak számára az utóbbi szimpatikusabb lehet, akár az ingyenessége, akár a velejáró szabadság miatt.

Ezek a programok azonban nem lennének teljesek az általuk betölthető pluginok nélkül, amelyekkel a DAW szoftvercsomagok képességei igazán kihasználhatóak. Egyik ilyen nyílt plugin formátum az LV2, amely az LADSPA-nak utódja. A keretrendszer gyakorlatilag bármilyen plugin megvalósítását lehetővé teszi a bővíthetősége miatt.

Szakdolgozatom elején a Linux audio környezetet mutatom be. Ezt követően az LV2 alapjait ismertetem, végezetül három általam elkészített pluginon keresztül bemutatom, hogyan lehet LV2 keretrendszerben hangszer-, illetve effektmodult készíteni, továbbá milyen lehetőség van egy modul grafikus felületének megírására.

Abstract

Due to the development of technology and the spread of digital audio contents, people wanted to start producing, editing audio content using their own personal computer. One of the best software for this purpose is called Digital Audio Workstion (DAW). There are plenty of commercial DAWs, but free, open-source alternatives also already exist. Because of the price or the freedom it offers, some people prefer these software over over the commercial ones.

DAWs wouldn't be complete without plugins. Plugins help us to get most out of the sofwares. One of the free plugin frameworks is called LV2, the successor of LADSPA. The framework allows us to implement various kinds of plugins thanks to the extensions.

In my thesis first, I will explain the Linux audio system. After that I will explain the basics of LV2. Finally I present how an instrument, and an effect plugin can be created in LV2, and how to implement a graphical interface for a plugin.

Bevezető

A digitális audiotartalmak térhódításával egyre jobban terjednek a Digital Audio Workstation (röviden DAW) szoftverek is. Ezek az eszközök lehetővé teszik a digitális audiofelvételek készítését, szerkesztését illetve lejátszását. A mai DAW szoftverek nem lennének teljesek a különböző pluginarchitektúrák nélkül. Ezek teszik lehetővé, hogy külső forrásból, vagy akár saját fejlesztésű modulokat használhassunk. Ezek közül a legismertebb a Steinberg VST (Virtual Studio Technology), az Apple-féle Audio Unit, illetve az elsősorban linuxos környezetben használt LV2 (Linux Audio Developer's Simple Plugin API version 2).

Szakdolgozatomban elsősorban az utóbbival fogok foglalkozni. Mivel a Linuxot használók meglehetősen kis részét teszik ki az asztali gépek piacának, továbbá a Linux alapú disztribúciót használók gyakran előnyben részesítik az ingyenes, nyílt szoftvereket, ezért kevés szoftvergyártó cég foglalkozik az ezt a platformot megcélzó fejlesztésekkel. Ez nem volt másképp az audio szoftverekkel sem, ezért a közösség tagjai elkezdték elkészíteni a saját szoftvereiket, azonban 2000-ig nem volt egy egységes alkalmazásprogramozási felület (Application programming interface, API), ami segítette volna az alkalmazások egymás közötti kommunikációját. Azonban ennek az évnek az elején a Linux Audio Developer levelezőlistán megjelent egy új API-nak a prototípusa. Ez az API volt az LADSPA, vagyis a Linux Audio Developer's Simple Plugin API, amely az LV2 elődjének tekinthető.

A téma ötletét két érdeklődési köröm metszete adta, a zene, illetve a szabad szoftverek világa. Már általános iskolás korom óta foglalkozom hangszeres zenével, illetve használok otthon valamilyen Linux disztribúciót. Eleinte csak felhasználója voltam a szabad szoftvereknek, majd későbbiekben kezdtem el utánajárni a mögötte álló filózófiának, mozgalomnak. Ezzel párhuzamosan folytattam a zenei tanulmányaimat, majd 2014-ben egy zenész ismerősöm által megismerkedtem a digitális zeneszerkesztés alapjaival. Ekkor jött az ötlet, hogy elkezdjem tanulmányozni, hogy milyen FLOSS (Free/Libre and Open Source Software) megoldások léteznek ebben a témakörben. Ahogy az egyetemi tanulmányaimat folytattam, egyre inkább kezdett érdekelni nem csak ezeknek a szoftvereknek a használata, hanem működése is. Így jutottam el arra a pontra, hogy szeretnék megismerkedni ezeknek a fejlesztésével. Több pluginformátum létezett, mikor belekezdtem a munkába. Választhattam az LADSPA, a DSSI (Disposable Soft Synth Interface), illetve az LV2, továbbá a VST között. A VST-t elvetettem, mert akkoriban még nem volt hivatalosan támogatva,

továbbá zárt megoldás volt.[1] Így választhattam az előző háromból. Mivel az LV2 az előző kettőnek a leváltása céljából jött létre, továbbá ezt fejlesztik aktívabban, így emelett döntöttem. Célomul tűztem ki, hogy mind egy hangszer plugint (szintetizátort), mind egy effekt plugint létrehozzak, továbbá megismerkedjek a grafikus felületük programázásának alapjaival.

Dolgozatomban először a Linux audio környezetét fogom bemutatni, annak rétegeit, kiemelve a JACK-et (vagy teljes néven JACK Audio Connection Kit), ami jelenleg az egyik legelterjettebb eszköz a tartalomkészítéshez, illetve az élő előadások megvalósításához. A Linux audiokörnyezetet a dolgozat 1. fejezete mutatja be. Szakdolgozatom 2. fejezetében az LV2 API-t fogom részletezni. Először a történetéről fogok említést tenni, ezt követően bemutatom, hogy milyen fájlok szükségesek egy működő pluginhoz. Ebben a fejezetben ismertetem a pluginok két alapvető fajtáját az effekt, illetve hangszer pluginokat, valamint az ezek közti különbségeket. A 3. fejezetben pedig bemutatom az általam elkészített pluginokat, azoknak felépítését, a fejlesztés menetét, továbbá a használt külső forrásokat.

[1] 2017 márciusától kezdve a VST SDK duál licenszelés alatt GPL(GNU General Public License) alatt is elérhető, továbbá a Linux is hivatalasan támogatott platform

1. fejezet

A Linux audio környezet bemutatása

Ha valaki Linux környezetben audioval szeretne foglalkozni (akár csak zenehallgatás céljából), akkor gyakran szembe találhatja magát különböző három vagy négy betűből álló mozaikszavakkal, mint például ALSA, JACK, OSS. Ebben a fejezetben bemutatom ezeknek kapcsolatát, illetve azt, hogy milyen célt szolgálnak. Ezek a kifejezések általában a Linux audio egyes rétegeit takarják. Leegyszerűsítve: közvetlenül a hardver fölött helyezkedik el az úgy nevezett kernel réteg (pl. ALSA, OSS), efölött pedig a különböző hangszerverek (JACK, PulseAudio), amelyekkel az alkalmazások kommunikálnak.

1.1. Kernel réteg

Kezdetben a Linux hanggal kapcsolatos feladatait a kernel rétegben az OSS (Open Sound System) látta el, ami több más Unix, illetve Unix-szerű operációs rendszerhez próbált egy egységes API-t biztosítani az audioalkalmazások megírásához. Kezdetben minden UNIX vendornak megvolt a saját API-ja ami az audiofeldolgozásért felelt. Így minden alkalmazást meg kellett volna írni külön-külön az adott API-val kompatibilis módon, hogy működjön az adott környezetben [1]. Ezt próbálta meg kiküszöbölni az OSS, egységes felületet biztosítva az alkalmazások számára. Linux alatt nagyon sokáig ez volt az alapértelmezett hangkártyakezelő rendszer, egészen 2002-ig. Kezdetben ez nyílt szoftver volt, azonban 2002-ben a 4Front Technologies az OSS 4-es verzióját zárt szoftverként adta ki, ezért vehette át a szerepét a kernelben az ALSA (Advanced Linux Sound Architecture).

Az ALSA projekt azért jött létre, mert a hangkártya meghajtóprogramok nem voltak aktívan fejlesztve a Linux kernelben, továbbá elmaradtak képességeikben az akkori új technológiákhoz képest. Jaroslav Kysela kezdte el a projektet, aki korábban már írt egy hangkártya meghajtóprogramot. Idővel több fejlesztő csatlakozott a projekthez, akik további eszközök támogatását tették lehetővé. Az ALSA-t úgy tervezték, hogy támogasson bizonyos funkciókat, amit akkoriban még az OSS nem támogatott. Ilyen például a hardveralapú többcsatornás keverés, vagy a hardveralapú MIDI szintézis. Mivel sok korábbi alkalmazás az OSS API-t használta, ami eltért az ALSA API-jától, ezért a disztribúciók egy része az ALSA-t úgy szállította, hogy az egy emulációs réteget biztosítson a régebbi alkalmazások számára [2].

1.2. Hangszerverek

Az OSS-nek azonban voltak hiányosságai. Többek között képtelen volt egyidejűleg több audiofolyam lejátszására, valamint több hangkártya kezelésére. A hiányosságok pótlása miatt megjelentek a különböző hangszerverek. Ezek szolgáltatásként (daemonként) futva biztosítják ezeket a funkciókat. Általában a hangszerver a különböző alkalmazásoktól megkapja a hívásokat, majd ezeknek az audiofolyamát összekeveri, majd kiküldi az így összeállított folyamot az eszközre. Ma leginkább két elterjedt hangszerver van, az egyik a PulseAudio, a másik a JACK. A két hangszervert más-más funkciók miatt használják. Hétköznapi használatra a disztribúciók általában a PulseAudiot tartalmazzák. A PulseAudio az általános felhasználókat célozta meg, akik számára az egyszerű használat volt a szempont.

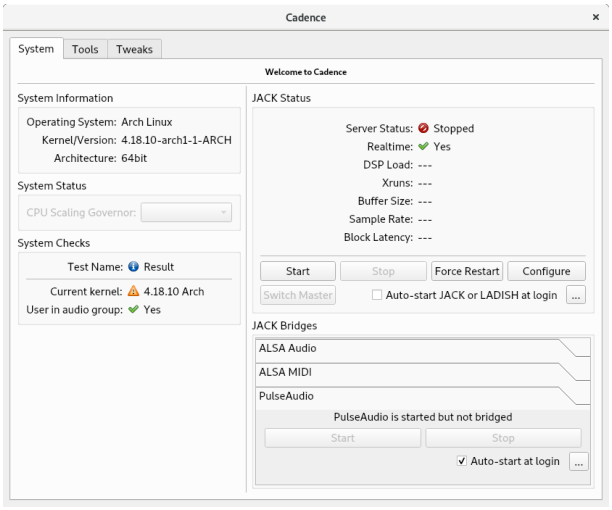
A JACK azonban más szempontot tartott szem előtt. A fejlesztők egy olyan hangrendszert akartak kidolgozni, amely a különböző audioprogramok kimenetét egy másik program bemenetébe képes küldeni, mindezt alacsony késleltetés mellett. Így megadja a felhasználónak a lehetőséget arra, hogy mely program kimenetét, mely másik bemenetébe, esetleg közvetlenül a hangkártyához küldje. Így leginkább egy hardveres stúdióhoz hasonlít a legjobban, ahol az egyes eszközök ki- illetve bemenetét, kábelekkel kötjük össze. A JACK támogatja, hogy egy kimenet több bemenethez, illetve hogy egy bemenethez több kimenet csatlakozzon.

Bár a két hangszerver más felhasználó csoportot céloz meg, egyesek számára fontos lehet, hogy párhuzamosan tudja használni mind a PulseAudio mind a JACK API-t támogató alkalmazásokat. Bár ez több módon lehetséges, a tapasztaltabb felhasználók megpróbálják ezt elkerülni, az úgynevezett „xrun”-ok minimálisra csökkentésének érdekében. Az xrun puffer alul-, illetve túlcsordulás is lehet, mely akkor jön létre ha nem sikerül az alkalmazásnak időben továbbítania az adatot az ALSA audio pufferébe, illetve ha nem sikerült onnan időben feldolgoznia. Ezek az xrunok általában recsegésként hallhatóak a kimeneten [3]. Ezért a gyakorlottabb felhasználók, hogy ezeket elkerüljék, megpróbálják csökkenteni az általuk használt alkalmazásokat, szolgáltatásokat, így például a PulseAudiot letiltják vagy akár törlik a rendszerből. Akik mégis párhuzamosan használnák a kettőt, azok számára is biztosítottak teret a fejlesztők. Egyik lehetőség másik hangkártyát használni a JACK-hez és a PulseAudiohoz, másik lehetőség, pedig a PulseAudiot a JACK-en keresztül átvezetni. Ebben az esetben a PulseAudio kliensként jelenik meg a JACK-ben, így lehetőség nyílik a PulseAudiot használó alkalmazások ki- és bemenetére más JACK API-t használó alkalmazások ki- és bemeneti portjait kötni [4].

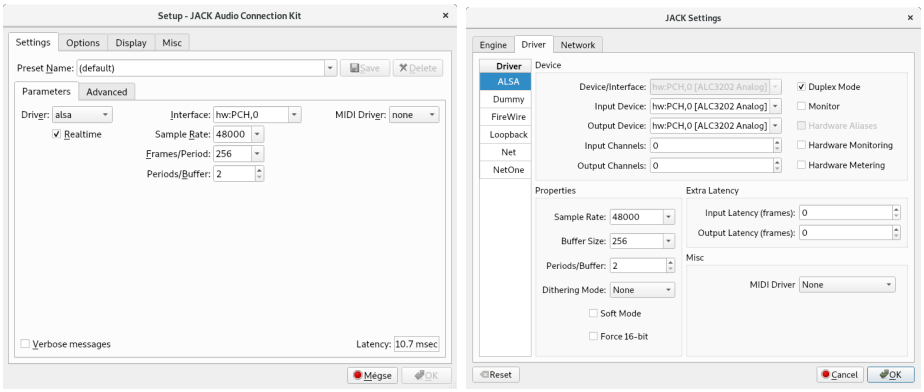
1.3. JACK konfigurálása, bemutatása a saját környezetemben

A JACK-et legkönnyebben a disztribúció csomagkezelőjéből lehet telepíteni, ezt követően több eszköz is rendelkezésre áll ennek konfigurálására. Lehetőség van shellben konfigurálni, azonban érdemes telepíteni valamilyen grafikus alkalmazást, amelyből szintén elindítható, illetve az alkalmazások közötti kapcsolatok is szervezhetőek. Ezek közül legismertebb a QjackCtl (1.1. ábra) illetve a Cadence (1.2. ábra). A JACK konfigurálását a

1.1. ábra. *A QjackCtl főablaka*

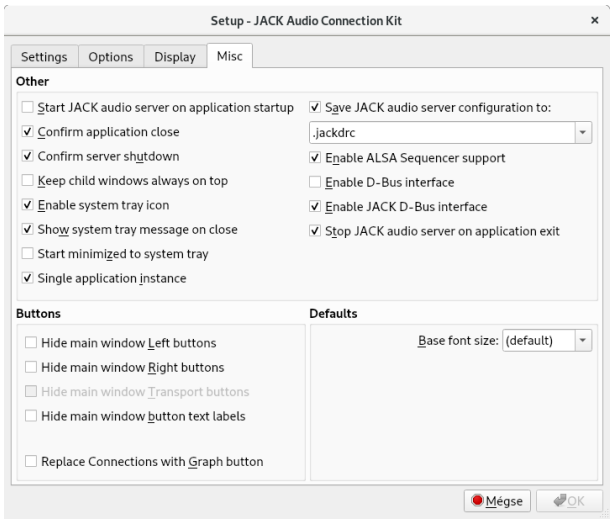


1.2. ábra. *A Cadence főablaka*

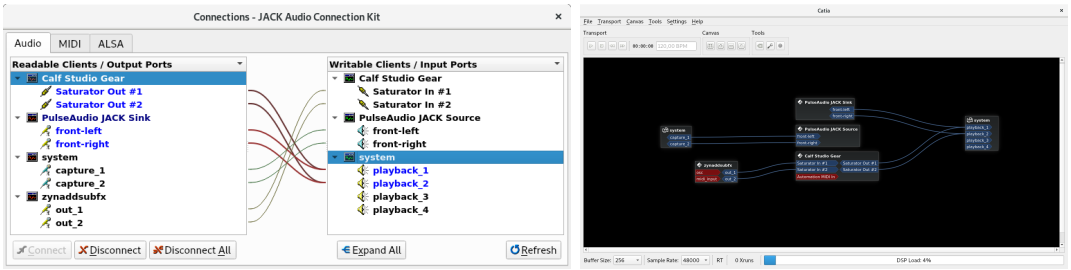


1.3. ábra. *A JACK paramétereinek beállítása, bal oldalon a QJackCtl, jobb oldalon a Cadence felülete látható*

QjackCtl alól a *Setup* gomb, Cadence alól a *Configure* gomb megnyomásával tudjuk elérni. Mindkettő felületén (1.3. ábra) hasonló paraméterek beállítására van lehetőség. Ezek közül a legfontosabbak a *Driver*, a *Device interface*. A *Driver* az általaunk használt meghajtóprogramot állítja be, a device interface-nél pedig a használni kívánt hangkártyát állíthatjuk be. Továbbá a késleltetés minimalizására a *Sample Rate*, a *Buffer size* (vagy QJackctl alatt: *frames/period*), illetve a *periods/buffer* paraméterek változtatásával van lehetőségünk. A *sample rate* fordítottan, a *buffer size* pedig egyenesen arányos a késleltetés mértékével. A *periods/buffer* ajánlott értéke 3, ha USB eszközt használunk, illetve 2, ha nem [5]. QJackCtl alatt további beállításra lehet szükségünk, ha párhuzamosan kívánjuk használni a JACK-et a PulseAudioval. Ekkor a QJackCTL *Setup*-jában a *Misc* fülön (1.4. ábra) engedélyeznünk kell a JACK D-BUS interfészt, melyet az *Enable JACK D-BUS*



1.4. ábra. *A qjacktl setupjának a misc füle*



1.5. ábra. *Ugyanazon JACK munkamenet kapcsolatai szervezése, bal oldalon QjacktCtl-ben jobbon pedig as Cadence-ből könnyen elérhető Catia-ban*

interface melleti jelölőnégyzet kipipálásával tehetünk meg.

A hangszerver elindítását követően lehet elkezdeni szervezni a JACK kliens alkalmazások közötti kapcsolatot. QJackCtl-t használva ez a felület a főablak (1.1. ábra) *Connect* gombjára kattintva érhető el. Amint az 1.5. ábra bal oldalán látható, itt három fül áll rendelkezésre. Mindegyik esetben az alkalmazások kimeneti portjai a bal oldalon találhatóak, míg a bemeneti portjai a jobbon. Az *Audio* fül alatt láthatóak az audio portok, a *MIDI* fül alatt a JACK MIDI portok, és az *ALSA* fül alatt az ALSA MIDI portok. Cadance-ben a főablak (1.2. ábra) *Tools* fülére kattintva a JACK-hez egyéb hasznos eszközöket érhetünk el. Ilyen például a Catia, ami az előbbihez hasonló módon segít az alkalmazások közötti kapcsolatok szervezésében, bár kicsit más felülettel (1.5. ábra baloldali képe) rendelkezik. Itt az alkalmazások portjai egy egységbe (téglalapba) vannak zárva. Az alkalmazásokat jelképező téglalap baloldalán láthatóak a bemeneti portok, jobbon pedig a kimenetiek. Kék szín az audio, piros a MIDI portokat ábrázolja. Az alkalmazás nem engedi hogy a különböző típusú portokat, illetve az azonos „irányú” (például 2 kimeneti, vagy 2 bemeneti) portokat egymáshoz kötni.

2. fejezet

Az LV2 keretrendszer és egy host program, a Qtractor

2.1. Az LV2 előzménye, születése

Mint azt a bevezetőben említettem, az LV2 elődjének az LADPSA tekinthető. Ennek az API-nak a prototípusát Richard W.E. Furse 2000. február 28-án mutatta be a linux-audio-dev levezőlistára írt levelében [6]. Ebben leírta, miért hozta létre az API-t, és miért ilyen felépítést alkotott hozzá. Az első stabil verziót azonban hetekkel később, április 2-án jelentette be ugyanezen levelezési listán [7]. A fejlesztés célja, hogy minél hatékonyabb, könnyen megtanulható és egyszerű API szülessen. Bár stabilnak mondta, ennek ellenére senkit sem biztatott arra, hogy VST helyett ezt az API-t kezdje el használni. Sok hiányossága ellenére nagyon sokáig ez volt a linux audio pluginok elterjedt szabványa.

A hiányosságok pótlására Chris Cannam 2004. április 27-én írt egy levelet [8] a linux-audio-dev listára, amiben egy új API-ra tett javaslatot, melyet Steve Haris-szel együtt hozott létre. Itt olyannak írja le a DSSI-t, mint a VSTi, vagyis egyfajta LADSPA hangszerek számára. A leveléhez csatolt egy RFC-t (Request For Comments), melyben az API-ra vonatkozó specifikáció szerepel, továbbá egy headert, melynek a dokumentációja magában a fájlban volt található.

Steve Harris egy 2006. áprilisi levelében [9] kifejtette, hogy újra át kell gondolni az LADSPA-t, amivel a két évvel korábbi LAC-on (Linux Audio Conference) már több ember egyetértett. Javasolta, hogy az új pluginok csomagokban (könyvtárakban) legyenek elérhetőek, melynek az ötletét az OpenStep projektből merítette. Ezek a csomagok tartalmazzák a pluginhoz tartozó bináris fájlt, illetve a szükséges adatokat, amelynek a leírásához a Turtle [10] szintaxist választotta. Hogy ezt bemutassa az új API headerrel, egy új (bár nem tesztelt) pluginnak az elérhetőségét közzétette levelében. Ezt követően többen is csatlakoztak a projekthez, bár kisebb vitát eredményezett a levelezési listán, Steve Harris végül az LV2 mellett döntött. A projekt első stabil verzióját 2008. január 9-én jelentették be [11], melyet az LADSPA egyszerű, de könnyen bővíthető utódjaként emlegettek, mely a bővítményekkel bármilyen funkciót lehetővé tesz az LADSPA-val ellentétben.

2.2. Egy plugin felépítése

Ahogy azt az előzményekben említettem, egy plugin egy csomagból áll, mely két dolgot tartalmaz, (1) a pluginhoz tartozó bináris fájl(oka)t, illetve (2) az őt leíró adatokat Turtle szintaxisban. Ezeknek a részletezéséhez a Programming LV2 Plugins „könyvet" [12] vettem segítségül.

Minden csomagnak tartalmaznia kell egy `manifest.ttl` fájlt, ami plugin belépési pontja. Mivel a Turtle fájlok nagyon sok URI-t (Uniform Resource Identifier)-t tartalmazhatnak, ezért az egyszerűbb olvashatóság, szerkesztés céljából lehetőségünk van ezekhez prefixeket rendelni, későbbiekben így a hosszú URI-k helyett a már sokkal rövidebb prefixeket használhatjuk. Ismertetőmben, a „könyvben" is használt prefixeket fogom használni, melyek az alábbiak:

```
@prefix atom: <http://lv2plug.in/ns/ext/atom#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix lv2:  <http://lv2plug.in/ns/lv2core#> .
@prefix midi: <http://lv2plug.in/ns/ext/midi#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix urid: <http://lv2plug.in/ns/ext/urid#> .
```

A manifest.ttl fájl a könyvtárban található pluginokat, erőforrásokat tartalmazza, illetve arról ad leírást, hogy a pluginról mely fájl tartalmaz további információkat. Ezt az alábbi módon tehetjük meg:

```
<http://example.org/lv2/plugin>
        a lv2:Plugin ;
        lv2:binary <plugin.so>  ;
        rdfs:seeAlso <plugin.ttl> .
```

Az első sor a plugin egyedi URI azonosítója. A következő sorban ennek az URI-nek a definíciója látható. Az `a` az angol „is a"-nek, magyarul „ez egy"-nek felel meg. A sor végén található egy pontosvessző, amely jelzi, hogy az előző tárgyhoz tartozó leírást szeretnénk folytatni. A következő sorban a pluginhoz tartozó bináris fájl elérhetőségét adhatjuk meg. Végül az `rdfs:seeAlso` adja meg, hol találhatunk további információt a pluginról. Bár a további leírást is megtehetnénk a manifest fájlban, azonban célszerű különválasztani, hogy a host programnak csak a rövidebb fájlt kelljen végigolvasni, megrövidítve a pluginok felderítésének idejét. Ebben a példában a specifikusabb információkat a (nemlétező) plugin.ttl tartalmazza:

```
<http://example.org/lv2/plugin>
        a lv2:Plugin ;
    doap:name "Example LV2 Plugin" ;
        doap:license <http://opensource.org/licenses/isc> ;
        lv2:project <http://example.org/lv2> ;
        lv2:requiredFeature feature1 ;
        lv2:optionalFeature feature2 ;
```

Ebben a fájlban szintén a manifest.ttl-ben definiált URI-t kell használni, különben a host alkalmazás nem tudja betölteni a plugint. Minden pluginhoz kell rendelni egy nevet, mely a `doap:name` sorban látható, rendelhetünk hozzá licenszet, amit a következő sor mutat, továbbá projektet, amit az `lv2:project` sor mutat. Az `lv2:requiredFeature` sorral megkövetelhetjük, hogy a host program támogassa a *feature1* funkciót. Ha támogatja, akkor

a hozzátartozó URI-t a továbbítania kell a plugin számára az `LV2_Descriptor::instantiate()` függvényben. Amennyiben a host nem támogatja, akkor a pluginnak nem szabad betölteni. Végül az `lv2:optionalFeature feature2` sor jelzi a host számára, hogy a plugin támogatja a feature2 funkciót. Továbbá definiálnunk kell a pluginban használt különböző portokat. Ezeknél a portoknál két dolgot kell meghatározni: az irányát (`lv2:InputPort` vagy `lv2:OutputPort`), továbbá az adattípust. Erre a szakdolgozatom 3. fejezetében fogok példát mutatni, a saját pluginjaimon keresztül.

A plugin kódjában definiálnunk kell, bizonyos függvényeket, struktúrákat. Az egyik legfontosabb struktúra `LV2_Descriptor`, amelyben definiáljuk a pluginunk URI-jét, amely meghatározza, mely plugint tölti be a host, ehhez a plugin `lv2_descriptor` függvényét hívja meg, amely a plugin könyvtár belépési pontja. Az `LV2_Descriptor` struktúrában továbbá meg kell határozni bizonyos függvényeket az alábbi sorrendben:

```
static LV2_Handle
instantiate(const LV2_Descriptor* descriptor, double rate,
            const char* bundle_path, const LV2_Feature* const* features)
static void connect_port(LV2_Handle instance, uint32_t port, void* data)
static void activate(LV2_Handle instance)
static void run(LV2_Handle instance, uint32_t n_samples)
static void deactivate(LV2_Handle instance)
static void cleanup(LV2_Handle instance)
static const void* extension_data(const char* uri)
```

Az `instantiate` függvény a plugin példányának létrehozásakor hívódik meg. A host a pluginnak átadja paraméterül a `descriptor`t, a mintavételi frekvenciát (`rate`), egy könyvtár elérhetőségét (`bundle_path`) amennyiben a további erőforrásokra lenne szüksége a pluginnak, és host által biztosított funkciók listájára mutató pointert (`features`), amit szükségesnek definiáltak a plugin Turtle fájljában. Ebben a függvényben van lehetőségünk a plugin számára szükséges memóriaterületet lefoglalni.

Ennek a párja a `cleanup`, melyben az előbb lefoglalt memória területet kell felszabadítani. Ez a plugin példány megsemmisülésekor hívódik meg.

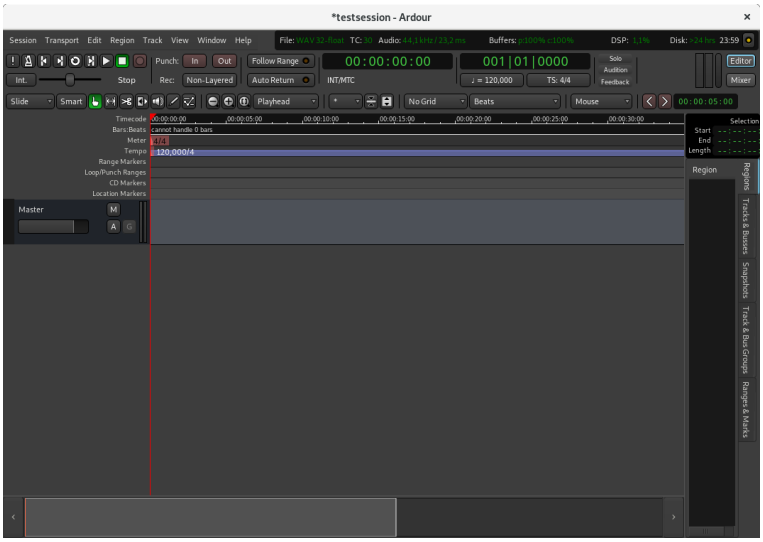
A `connect_port` függvényben tudjuk hozzárendelni a példányunkhoz (`instance`), Turtle fájlban definiált portokat az indexük (`port`) szerint. Az `activate` függvény előkészíti a plugint futásra, melynek a párja a `deactivate`, melyet a host a plugin futása utána hív meg.

A `run` függvény a plugin fő függvénye, amelyben a plugin funkcióját kell megvalósítani, itt lehet feldolgozni a bejövő audio puffert, illetve a kimenőt előállítani. Azonban ha a plugint `lv2:hardRTCapable` funkció biztosításával definiáltuk (`lv2:optionalFeature`), akkor ügyelni kell arra, hogy a blokkolás, illetve a memóriafoglalás tilos.

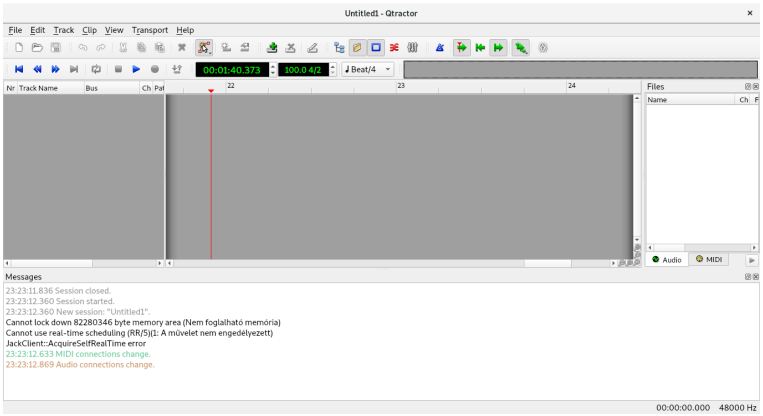
Az `extension_data` függvénynek a plugin által támogatott kiterjesztéseket kell visszatérítenie.

2.3. Qtractor

A pluginok használatához azonban szükséges egy host program, linux alatt több lehetőség közül választhatunk. Legismertebb ingyenes, nyíltforráskódú DAW-ok, melyek támogatják az LV2-t az Ardour (2.1. ábra), illetve a Qtractor (2.2. ábra). Ebben a részben a Qtractorban történő teszteléshez szükséges lépéseket mutatom be röviden.



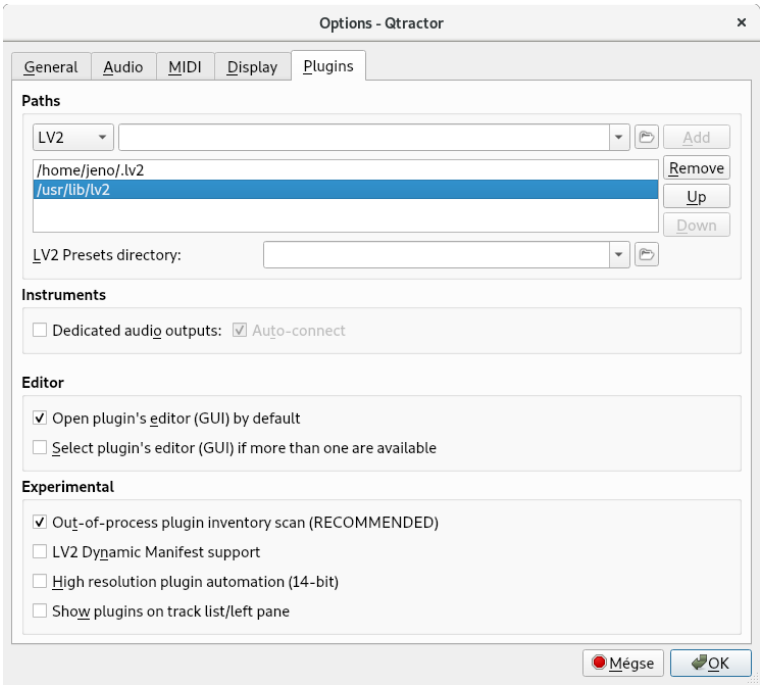
2.1. ábra. *Az Ardour felülete*



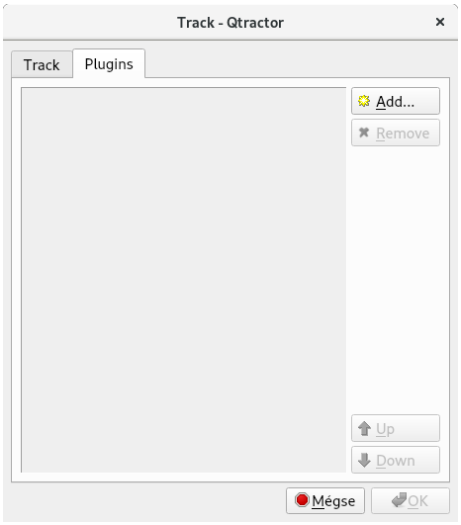
2.2. ábra. *A Qtractor felülete*

A Qtractor bár 2005-ben jelent meg először, a fejlesztője 2018-ban is még béta verzió néven jelenteti meg az újabb kiadásokat. Ennek ellenére a program stabilnak mondható, és funkcionalitásban is kielégítő lehet. A projekt oldala szerint az LV2 mellett más plugin formátumokat is támogat, mint például a régebbi LADSPA, DSSI, de a natív VSTi-t is támogatja. A különböző pluginformátumok mellett sokféle audioformátumot is támogat, többek között az OGG, az MP3, a WAV is szerepel a leírásában. A program nem korlátozza az audio/MIDI sávok használatát, továbbá a sávonkénti maximális pluginmennyiséget sem, így ezeknek gyakorlatilag csak a hardver szab határt. További képességeiről a projekt oldalán [13] lehet tájékozódni.

Ahhoz, hogy az LV2 pluginokat tesztelni lehessen, a Qtractor által használt könyvtárak egyikébe kell helyeznünk a pluginokat tartalmazó könyvtárakat. Ezeket a könyvárakat a Qtractor *View -> Options* menü (ez a az ablak az F12-es gomb megnyomásával is előhívható) *Plugins* füle alatt lehet megtekinteni, ahogy azt a 2.3. ábra mutatja. Itt a többi plugin által használt könyvtárakat is meg lehet tekinteni a legördülő menüből amelyen jelenleg az *LV2* felirat látható. További könyvtárak hozzáadására is van lehetőség, az *LV2* sorában lévő kis dosszié ikonra kattintva feljön egy fájlböngésző, amelyben ki lehet válasz-



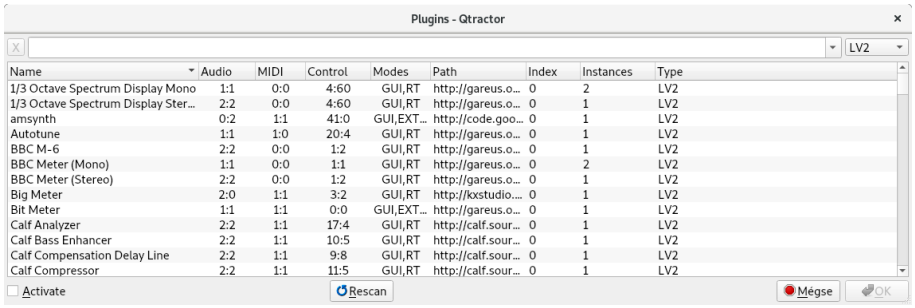
2.3. ábra. *Qtractor Options ablakának a plugin füle*



2.4. ábra. *Pluginok hozzáadása a sávhoz*

tani az új könyvtárat, melyet ezután az *Add* gomb használatával lehet hozzáadni. Ahhoz, hogy érvénybe lépjenek az új beállítások, az *Ok* gombra kell kattintani az elmentéséhez, majd újra kel indítani az alkalmazást.

A pluginok teszteléséhez egy MIDI/audio sáv szükséges, amelyet a főablak (2.2. ábra) *Track* menüje alatt található *Add new track*-kel létre is lehet hozni, vagy pedig importálni is lehet egy audio/MIDI fájlt ugyanezen menü *Import* almenüje alatt a *MIDI*-t vagy az *Audio*-t kiválasztva. Sávot létrehozva/importálva a beállításai a sávra jobb egérgombbal kattintva a *Track Properties*-ből érhetőek el. Itt a *Plugin* fül alatt (2.4. ábra) lehet hozzáadni a sávhoz a pluginokat. Az *Add* gombra kattintva feljön egy lista, amely a használható pluginokat tartalmazza.



2.5. ábra. *Pluginok listája*

3. fejezet

Az elkészített modulok

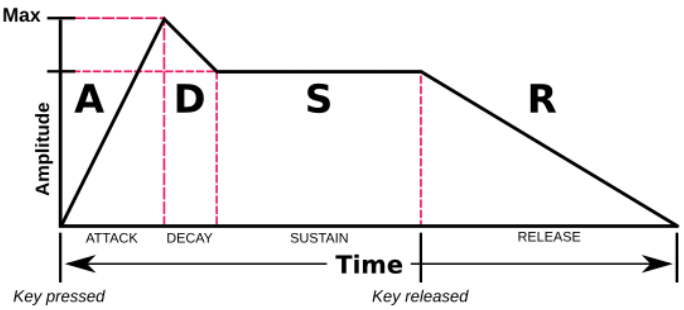
3.1. A modulok elkészítésének folyamata

A pluginok elkészítéséhez az LV2 oldalán fellelhető „könyvet” vettem segítségül [12]. Eleinte magával az API alapjaival ismerkedtem meg, úgy, hogy végignéztem a példa programokat, egyeseket lefordítottam és magam is leteszteltem. Mivel olyan URI-t tilos használni, amelynek tartománya felett nincs irányítása a szerzőnek, ezért programjaimban a `http://example.org/` domaint használtam, amely példák bemutatásának céljául szolgál.

Ezt követően kezdtem bele a szintetizátor fejlesztésébe. Az első cél itt egy folyamatos szinuszhang keltése volt. Miután ezt sikerült elérni, a következő lépés az volt, hogy MIDI billentyű lenyomására adja ki ezt a hangot, valamint a megfelelő magasságban. Hogy a mai virtuális szintetizátorok képességét jobban megközelítse a plugin a következő lépés volt az ADSR (Attack Decay Sustain Release) burkológörbe megvalósítása (3.1. ábra [14]). Az ADSR-nek négy paramétere van, az első paraméter (*Attack*) határozza meg, hogy a hang mennyi idő alatt éri el a maximális hangerősséget. A második paraméter, vagyis a *Decay* határozza meg, hogy a hang a maximális amplitúdó elérését követően mennyi idő alatt csökken le a harmadik (*Sustain*) paraméter által meghatározott szintre. A hang amlitúdója ezen a szinten marad, a *Decay* szakasz után, ha a billentyűt nyomva tartjuk. Végül az utolsó paraméter, a *Release* azt az időintervallumot határozza meg, ami alatt a hang elhalkul a billentyű elengedését követően. Miután az ADSR is megvalósításra került, egy új funkciót, a hullámformák közüli választást tettem lehetővé, ahol három hullámformát valósítottam meg: a szinusz-, a négyszög-, és a fűrészjelet.

Ezt követte a hangszínszabályozó elkészítése, melyhez Vinnie Falco *DSPFilters* szűrő gyűjteményét vettem segítségül [15]. Először egy egyszerű mono szűrőt készítettem el amely működőnek bizonyult. Ezt követően további szűrők beépítésével értem el a kívánt hangszínszabályozót, amelyet végül további egy kimeneti- és egy bemeneti porttal bővítettem, amivel a plugin már sztereó képessé vált.

Végezetül az LV2 grafikus felületének programozásával ismerkedtem meg. Itt egy egyszerű aluláteresztő szűrőnek (LPF vagy LowPass filter) a felületét készítettem el, melynek két változtatható paramétere volt. A fülelet programozására több widgetkészlet is rendelkezésre áll. Mivel az LV2 „könyv” példájában a GTK használata megtalálható, ezért



3.1. ábra. *Az ADSR burkológörbe ábrája*

úgy gondoltam, hogy egy másik widgetkészletet fogok használni. Így esett a választásom a Qt-ra, mely elérhető mind kereskedelmi, mind szabad licensz [16] alatt.

3.2. A szintetizátor plugin

A szintetizátor működéséhez különböző ki- és bemeneti portokat kell definiálni az őt leíró Turtle fájlban. Mivel MIDI billentyűzet támogatása volt a cél, így szükséges egy minimum egy bemeneti portot (`lv2:InputPort`), és a sztereó hanghoz két kimeneti portot (`lv2:OutputPort`) definiálni, továbbá a host programtól meg kell követelni (`lv2:requiredFeature`), hogy támogassa a (`urid:map`) funkciót. Így a manifest fájlból linkelt Turtle fájl eleje az alábbi módon néz ki:

```
<http://example.org/eg-sine>
        a lv2:Plugin ,
        lv2:InstrumentPlugin ;

        lv2:project <http://example.org/sine> ;
        lv2:requiredFeature urid:map ;
        doap:name "Simple Sine"  ;
        doap:license <http://opensource.org/licenses/isc> ;
        lv2:optionalFeature lv2:hardRTCapable ;
        lv2:port [
                a lv2:InputPort ,
                atom:AtomPort ;
                atom:bufferType atom:Sequence ;
                atom:supports midi:MidiEvent ;
                lv2:designation lv2:control ;
                lv2:index 0 ;
                lv2:symbol "in" ;
                lv2:name "In"
        ] , [
                a lv2:AudioPort ,
                lv2:OutputPort ;
                lv2:index 1 ;
                lv2:symbol "left" ;
                lv2:name "Left"
]
```

Mint látható, az `lv2:Plugin` típus mellett definiáltam a pluginnak egy második típust, hogy a host program számára jelezzem, hogy ez egy `lv2:InstrumentPlugin` is. A bemeneti por-

toknál most csak a MIDI és az egyik kimeneti portot írtam le, mert a másik hasonlít az utóbbihoz, csak eltérő indexeléssel, szimbólummal illetve névvel. Mint azt a 2.2. részben említettem, minden porthoz legalább két típust kell rendelni. Az irányon kívül (`lv2:InputPort` és `lv2:OutputPort`) szükséges még megadni az őt leíró adat típusát, audio esetén ez `lv2:AudioPort`, míg MIDI esetén az `atom:AtomPort` típus. `atom:AtomPort` egy `Atom`-ot tartalmaz, amely egy általános adattípus bármilyen adat számára. Mivel MIDI eseményeket szeretnék feldolgozni, ezért ebben az esetben a kötelező `atom:bufferType` típusa `atom:Sequence`, ami időbélyeggel ellátott események sorozatát jelenti.

Mivel ADSR funkciót is szerettem volna megvalósítani, ezért további `lv2:InputPort` portok felvételére volt szükségem, melyek típusához leginkább az `lv2:ControlPort` illeszkedett. Ennek használatát az alábbi példa szemlélteti:

```
[
        a lv2:InputPort ,
                lv2:ControlPort ;
        lv2:index 3 ;
        lv2:symbol "attack" ;
        lv2:name "Attack";
        lv2:default 0.0 ;
        lv2:minimum 0.0 ;
        lv2:maximum 2000.0 ;
        units:unit units:ms ;
        lv2:scalePoint [
                rdfs:label "+10" ;
                rdf:value 10.0
        ] , [
                rdfs:label "0" ;
                rdf:value 0.0
        ]
]
```

Mint látható, az `lv2:minimum`, illetve az `lv2:maximum` paraméterek segítségével egy tartományt lehet jelezni a host felé, amely azt adja meg, hogy az adott paramétert milyen értékek között lehessen változtatni, továbbá az `lv2:default` megadásával megadhatjuk az adott paraméter kezdőértéket a plugin betöltésekor. Az `lv2:scalePoint` segítségével pedig skála pontokat adhatunk meg, ami a hostnak elősegíti a megjelenítést. Ezt a hullámforma kiválasztásához használt port szemlélteti:

```
[

        a lv2:InputPort ,
        lv2:ControlPort ;
        lv2:index 7 ;
        lv2:symbol "form" ;
        lv2:name "Wave form";
        lv2:default 0 ;
        lv2:minimum 0 ;
        lv2:maximum 2 ;
        lv2:portProperty lv2:integer ;
        lv2:scalePoint [
                rdfs:label "Saw" ;
                rdf:value 1
        ] , [
                rdfs:label "Sine" ;
                rdf:value 0
```

```
                ] , [
                        rdfs:label "Square" ;
                        rdf:value 2
                ]
]
```

A programomban három hullámformát definiáltam, így ennek kezeléséhez egy ekkora tartományra volt szükségem. Hogy ebben a tartományban csak egész számértékeket vehessen fel ez a port, azt az `lv2:portProperty` segítségével állítottam be. Ezt követően a különböző számokhoz skálapontokat rendeltem, hogy a hullámformáknál ne 0, 1, 2 szerepeljenek, hanem a hullámformák nevei.

A plugint leíró kódban két struktúrát definiáltam, az egyik magát a plugint írja le, azoknak a puffereit, illetve a rendelkezésre álló „hangokat” (`Voices`), ezek közül az éppen használtak számát (`active_notes`), továbbá a mintavételi frekvenciát (`rate`), amely a megfelelő hangmagasság előállításához lesz szükséges. Ezt az alábbi struktúra írja le:

```
        typedef struct {
                // Port buffers
                const LV2_Atom_Sequence* in_port;
                float*       right;
                float*       left;
                const float* attack;
                const float* decay;
                const float* sustain;
                const float* release;
                const float* osc;
                Voice *voices;
                // Rate
                double rate;
                uint8_t active_notes;

                struct {
                        LV2_URID midi_MidiEvent;
                } uris;

                LV2_URID_Map*  map;
        } Sine;
```

A másik pedig egy adott billentyű lenyomáshoz tartozó hangnak a jellemzőit, aktuális állapotát tartalmazza. Ilyen például a lenyomott billentyű száma (`key`), a sebessége (`velocity`), az ADSR paraméterei, a hullámforma típusa (`form`) a billentyűlenyomás-üzenet érkezésének pillanatában. Ezeket az alábbi struktúra írja le:

```
typedef struct{
        uint32_t position;
        float freq;
        bool before_note_off;
        bool active;
        int8_t key;
        uint32_t pos_release;
        float value;
        uint8_t velocity;
        float attack;
        float decay;
        float sustain;
        float release;
        uint8_t form;
```

```
} Voice;
```

További fontos változói a `position`, amely a hullámforma megvalósításához szükséges. A `before_note_off`, amely igaz, ha lenyomva tartjuk a billentyűt, hamissá válik, ha felengedjük. A `pos_release` paraméter a felengedés utáni eltelt idő mérésére szolgál. Az `active` igaz, ha használatban van a hang, hamis egyébként.

A pluginom elején továbbá definiáltam egy makrót (`MAX_VOICES`), amely az egyszerre megszólaltatható hangok maximális számát jelöli.

Az `instantiate` függvényben lefoglalom a `Sine`, továbbá a `voices` számára a területet. Ezt követően megvizsgálom, hogy a host valóban támogatja-e a `urid:map` funkciót, amennyiben igen, az `instantiate` függvény sikeresen visszatér a létrehozott `Sine`-nal.

Ezt követően a `connect_port` függvényben, egy egyszerű esetszétválasztás segítségével párosítom a portokat a megfelelő pufferekhez.

Ezután az `activate` függvényben meghívom a `stopallnotes` függvényt, amely beállítja a `Sine active_voice` és a `voice`-ok paramétereit 0-ra, illetve false-ra, ez legyen a kiindulási állapot.

A `run`-ban pedig a plugin hangkeltésének működése található. Először megvizsgálja, hogy érkezett-e MIDI üzenet. Ha igen, akkor az üzenettől függően cselekszik. Amennyiben ez egy `NOTE_ON` üzenet, akkor megvizsgálja hogy kevesebb aktív hang van-e még (`active_voice`), mint maximális `MAX_NOTES`. Amennyiben több van, halad tovább a feldolgozással, ha pedig kevesebb, akkor megkeresi az első nem aktív `Voice`-t a `sine->voices` tömbjében, és meghívja a `keypressed` segédfüggvényt.

A `keypressed` függvénynek három paramétere van, melyek rendre `sine`, ami a pluginra mutat, `note`, ami a lista első nem aktív hangjának az indexe, végül a `msg`, vagyis a MIDI üzenet, amely alapján be tudja állítani a hang `key` és `velocity` paraméterét.

Ha `NOTE_OFF` üzenet érkezik akkor megkeresi a `sine->voices` tömbben azt az elemet, amelynek ugyanaz a billentyűkódja van és még le van nyomva. (`before_note_off==true`) és a `before_note_off` paraméterét false-ra állítja, jelezve, hogy ehhez a hanghoz már nem tartozik lenyomott billentyű.

`MSG_STOP` esetén pedig meghívja a fentebb már említett `stopalllnotes` függvényt.

A MIDI üzenetek feldolgozása után következik a hangkeltés. Az aktív hangok közül három esetet érdemes megkülönböztetni: (1) lenyomva tartott gomb, (2) elengedett gomb, de még a *release* időtartam vége előtt, (3) aktív hang, de már a *relese* időtartam lejárta után .

Az utolsó esetben `Voice active`-ját `false`-ra kell állítani és nem kell az audiopufferbe semmit sem másolni.

A másik két esetben azonban más-más módon kell a kimeneti jelet biztosítani. Általánosságban elmondható, hogy $f(t) = a(t) \cdot x(t)$, vagyis a kimeneti jel az amplitúdó és valamilyen periodikus függvénynek a szorzata. Mind az amplitúdó, és a periodikus tag meghatározására definiáltam egy-egy függvényt. Pontosabban az amplitúdóhoz kettőt, mert különválasztottam a nyomva tartott, illetve a felengedett billentyű esetét.

A periodikus részt a `voicegenerate` függvény határozza meg az alábbi módon:

```
float voicegenerate (Sine* sine,int i){
```

```
        float result=0.0f;
        float frequency = 440.0 * pow(2, (sine->voices[i].key-69) / 12.0f);
        float frequencyt=sine->voices[i].position/sine->rate*frequency;
        switch(sine->voices[i].form){
                case 1:
                        result=frequencyt-floorf(frequencyt);
                break;
                case 2:
                        result=round(frequenyt)-floorf(frequencyt);
                break;
                default:
                        result=sinf(2*M_PI*frequencyt);
                break;
}
        return result;
}
```

Mint látható, a hullámformától függően (`sine->voices[i].form`) választva téríti vissza a megfelelő értéket. A fűrészjelet az első, a négyszögjelet a második, és a szinuszjelet az alapértelmezett eset határozza meg.

Az amplitúdó meghatározására a `envelope` illetve a `releaseamp` függvényeket készítettem el. Előbbi a lenyomott, utóbbi a felengedett billentyű esetét határozza meg. Az első esetet az alábbi kód mutatja meg:

```
float envelope(Sine* sine, int i){
        uint32_t position = sine->voices[i].position;
        double rate =  sine->rate;
        float attack = sine->voices[i].attack/1000.0;
        float decay = sine->voices[i].decay/1000.0;
        float sustain = sine->voices[i].sustain;
        if(position/rate <= attack)
                return position/rate/attack;
        else if(position/rate<=decay+attack )
                return 1+(sustain-1)/decay*(position/rate-attack);
        else
                return sustain;
}
```

A paraméterek beállítása után először azt ellenőrzöm, hogy az *attack* ideje alatt történik-ea függvényhívás, emennyiben igen az ennek megfelelő értékkel térek vissza (lineáris esetben ez *eltelt_idő/maximális_idő*. Amennyiben nem, akkor megvizsgálom, hogy a billentyű lenyomása óta eltelt idő, elérte-e az $attack + decay$ időtartamot. Ebben az esetben a decay idő tartamra vonatkozó értéket térítem vissza. Ha pedig egyik eset sem igaz, akkor már a *sustain* értékkel térek vissza.

Az amplitúdó meghatázosához továbbá figyelembe vettem a *velocity*-t is, mivel ennek a maximális értéke 127, ezért a `voice`-hoz tartozó `velocity` értéket elosztottam 127-tel (ügyelve, hogy ne egész osztásként végezze el) és ezzel szoroztam meg a kimeneti értéket.

3.3. A hangszínszabályozó plugin

Ennél a pluginnál szerencsére könnyebb helyzetben voltam, köszönhetően Vinnie Falco DSPFilter gyűjteményének, mely MIT license alatt érhető el. Ez a gyűjtemény többféle filtert is tartalmaz, esetemben azonban ezek közül keveset használtam. Használatuk szeren-

csére nagyon egyszerű, a filter definiálása után, annak `setup` függvényével lehet beállítani a megfelelő paramétereit, melyek szűrőnként eltérőek lehetnek. Szerencsére a kódja nagyon jól van kommentezve, így ha valaminek a használatlában nem vagyunk biztosak, könnyen utána lehet járni. A szűrő beállítása után a `process` függvényével lehet feldolgozni az audiopuffert. Ennek a függvények minden esetben két paramétere van, első a minták, második a pufferek tömbje.

Ennél a feladatnál egy ötszűrős hangszínszabályozót valósítottam meg. Ezekhez a szűrőkhöz a korábban említett szűrőgyűjteményt használtam fel. A pluinonban összesen egy felülátereszto (`RBJ::HighPass`), három sáv- `RBJ::BandShelf` és egy aluláteresztő (`LowPass`) szűrőt használtam fel. Míg az elsőnek és utolsónak két-két változtatható paramétere volt, a három sávszűrőnek egyenként három. Hogy ezeket a paramétereket változtatni lehessen, összesen tizenkettő `lv2:ControlPort` szükséges. A sztereó működéshez további négy `lv2:AudioPort`-ot vettem fel, melyek közül kettő `lv2:InputPort` kettő pedig `lv2:OutputPort`.

Az hangszínszabályzót az alábbi struktúrába készítettem el:

```
typedef struct {
        const float* audio_in[2];
        float* audio_out[2];
        double sample_rate;
        const float* filter1par[2];
        const float* filter2par[3];
        const float* filter3par[3];
        const float* filter4par[3];
        const float* filter5par[2];
        Dsp::SimpleFilter <Dsp::RBJ::HighPass, 2> filter1;
        Dsp::SimpleFilter <Dsp::RBJ::BandShelf, 2> filter2;
        Dsp::SimpleFilter <Dsp::RBJ::BandShelf, 2> filter3;
        Dsp::SimpleFilter <Dsp::RBJ::BandShelf, 2> filter4;
        Dsp::SimpleFilter <Dsp::RBJ::LowPass, 2> filter5;
} Equalizer;
```

A kimeneti- (`audio_out[2]`) illetve bemeneti portok (`audio_in[2]`) mellett szerepelnek a szűrőkhöz tartozó portok. A `filter1par[2]` felülátereszto szűrő (`filter1`) portjainak felel meg. A `filter2par[3]` az egyik sávszűrő (`filter2`) portjainak felel meg, hasonlüan a 3-as illetve 4-eshez. Végezetül az aluláteresztő szűrőnek (`filter5`) portjait a `filter5par[2]` jelenti. A filterek működéséhez szükséges a mintavételi frekvencia (`sample_rate`) ismerete, melyet az `instantiate` függvényben állít be a plugin.

Miután az `instantiate` függvény lefoglalta a szükséges memóriaterületet a plugin számára, valamint beállította a mintavételi frekvenciát, a `connect_port` függvény a szintetizátorhoz hasonlóan index szerint összepárosítja a portok a megfelelő pufferekkel.

3.4. Ismerkedés az LV2 grafikus programozásával Qt keretrendszer használatával

Ebben az esetben az egyszerűség kedvéért egy egyszerű aluláteresztő szűrőhöz készítettem el egy GUI-t. Egy olyan felületet akartam létrehozni a számára, amelyben két tekerőgombhoz hasonló widgettel modullal lehet állítani a filter paramétereit, továbbá a

felület meg is jeleníti a változók aktuális értékeit.

Mivel nem csak egy binárisból áll a plugin, ezért ezt fel kell tüntetni a `manifest` fájlban:

```
<http://example.org/eg-filter>
        a lv2:Plugin ;
        lv2:binary <filter@LIB_EXT@>  ;
        rdfs:seeAlso <filter.ttl> .

<http://example.org/eg-filter#qt>
        a ui:Qt5UI ;
        ui:binary <filter_ui.so> ;
        rdfs:seeAlso <filter.ttl> .
```

Mint látható, a `http://example.org/eg-filter#qt` URI egy Qt5 felületet definiál, amelynek a binárisa `filter_ui.so` fájl, illetve további információt a `filter.ttl tartalmaz`. Ennek a (nem teljes) tartalma:

```
<http://example.org/eg-filter#qt>
        a ui:Qt5UI ;
        ui:binary <filter_ui.so> ;
        ui:requiredFeature ui:makeResident .

<http://example.org/eg-filter>
        a lv2:Plugin ;
        lv2:project <http://example.org/filter> ;
        lv2:requiredFeature urid:map ;
        doap:name "Example Filter"  ;
        doap:license <https://opensource.org/licenses/gpl-license> ;
        lv2:optionalFeature lv2:hardRTCapable ;
        ui:ui <http://example.org/eg-filter#qt> ;
```

Mint látható,a felület URI definíciója után, a plugin URI-jében, ezt a felületet az `ui:ui` kezdetű sorral rendelhetjük hozzá. Ezt követi a portok felsorolása. Ebben az esetben négy audioportot (két bemeneti, és két kimeneti), továbbá két `lv2:ControlPort`-ot definiáltam, melyek a filter paramétereit határozzák meg.

A filterplugint a `filter.cpp` fájlban az alábbi struktúra jelképez:

```
typedef struct {
const float* audio_in[2];
float* audio_out[2];
const float* filterpar[2];
double sample_rate;
Dsp::SimpleFilter <Dsp::RBJ::LowPass, 2> lpfilter;
} Filter;
```

A bemeneti- (`audio_in[2]`) illetve kimeneti portokon (`audio_out[2]`) kívül a két kontroll paraméter (`filterpar[2]`) található. A struktúrában továbbá egy egyszerű szűrő (`lpfilter`), valamint a szűrő megfelelő működéséhez szükséges mintavételi frekvencia (`sample_rate`) szerepel.

A portok megfelelő hozzárendelése után (`connect` függvény) a `run` függvényben megtörténhet a hang feldolgozása. A filter paramétereit az előző példához hasonló módon a `filter->lpfilter.setup` függvénnyel lehet megtenni. Ezt követően a bemeneti puffert átmásolom a kimeneti pufferbe, majd ezt a `filter->lpfilter.process` függvény feldolgozza.

A grafikus felület kódját a `filter_ui.cpp` tartalmazza.

Az UI készítéséhez két fejléc fájlt kell include-olni, egyik az ismert core, másik pedig az ui-t definiáló fejléc:

```
#include "lv2/lv2plug.in/ns/lv2core/lv2.h"
#include "lv2/lv2plug.in/ns/extensions/ui/ui.h"
```

Továbbá a Qt widgetek használatához is kell néhány fejléc fájl:

```
#include <QObject>
#include <QWidget>
#include <QDial>
#include <QLabel>
#include <QVBoxLayout>
#include <QString>
```

A core API-hoz hasonlóan itt is definiálni kell egy URI-t, ami azonosítja a plugint (interfészét). Egy ui nélküli pluginhoz hasonlóan itt is definiálni kell egy descriptort, mely az implementált, függvényeket tartalmazza, ez azonban eltér a core API-ban használtaktól.

Az `instantiate` illetve a `cleanup` itt is szükséges függvények azonban, szerepük hasonló a core API-ban látottakhoz. Ezt követően azonban egy `port_event` mely akkor hívódik meg ha plugin belső állapotában valami változás lépett fel (és ezt nem az ui kezdeményezte). Ilyen például ha egy host programban, valamilyen paramétert automatizálunk. Eze a függvény ahhoz szükséges hogy az ilyen változásokat az UI kezelje.

A szűrő grafikus felületét az alábbi osztály definiálja:

```
class FilterGui : public QWidget {
        Q_OBJECT
        public:
                FilterGui(QWidget* parent = 0);
                QDial* frequency_dial;
                QDial* qfactor_dial;
                QLabel* frequency_label;
                QLabel* qfactor_label;
                LV2UI_Controller controller;
                LV2UI_Write_Function write_function;

        public slots:
                void frequencyChanged(int value);
                void qfactorChanged(int calue);
};
```

Amint látható két QDialt definiáltam, egyik a szűrő frekvenciájáért (`frequency_dial`), a másik a jósági tényezőjéért (`qfactor_dial`) felel. Ezekhez egy-egy feliratot rendeltem amely a felületen kiírja az szűrő aktuális frekvenciájának (`frequency_label`), másik pedig a jósági tényezőnek (`qfactor_label`) értékét. A megfelelő működéshez továbbá definiálni kell a `write_function` függvényt amellyel az UI a plugin példányának bemeneti portjai `lv2:ControlPort` felé tudja küldeni az adatot. Továbbá egy `control`-ra is szükség van, amelyet az előbbi függvény meghívásakor első paramétereként kell átadni.

Ezeken felül definiáltam két `public slot`-ot. A `slot`ok olyan függvények amelyek bizonyos ui események által kiváltott, úgy nevezett `signal`ok válaszául szolgálnak. Ezáltal például a `QDial` tekerésekor frissíteni lehet a hozzátartozó címkét, így az aktuális érték jelenik meg felületen. Ezekről bővebb leírást Qt honlapja [17] ad.

Az előbbi osztály konstruktora a felületi elemeket (`QDial`, `QLabel`), egy rácsban helyezi el, úgyhogy felül legyenek a `QDial` elemek alattuk pedig a hozzájuk tartozó felirat:

```
FilterGui::FilterGui(QWidget* parent)
: QWidget(parent) {
        frequency_dial = new QDial();
        frequency_label = new QLabel();
        qfactor_dial = new QDial();
        qfactor_label = new QLabel();

        frequency_label->setText("400 Hz");
        frequency_dial->setRange(400, 16000);
        qfactor_label->setText("0.1");
        qfactor_dial->setRange(1, 100);

        QGridLayout* layout = new QGridLayout();
        layout->addWidget(frequency_dial,0,0);
        layout->addWidget(frequency_label,1,0);
        layout->addWidget(qfactor_dial,0,1);
        layout->addWidget(qfactor_label,1,1);
        setLayout(layout);
}
```

Minden `QDial`-nek van egy egészekkel körülhatárolt tartománya amelyben felvehet egy egész értéket. Ezt a tartomány a `QDial->setRange` függvényével lehet beállítani. Frekvencia esetében ezt a tartományt a 400 és 16000 Hz között határoztam meg, mivel a jósági tényezőt 0.1 és 10 közötti tartományban szerettem volna változtatni így az ehhez tartozó tartományt 1 és 100 között állítottam be, és ennek a tizedét jelenítem meg, illetve továbbítom a plugin felé, így a `slot`ok tartalma az alábbiképpen alakult:

```
void FilterGui::frequencyChanged(int value) {
        float freq = (float)frequency_dial->value()*1.0f;
        frequency_label->setText(QString("%1 Hz").arg(freq));
        write_function(controller, FILTER_FREQUENCY, sizeof(freq), 0, &freq);
}

void FilterGui::qfactorChanged(int value) {
        float qfact = (float)qfactor_dial->value()*0.1f;
        qfactor_label->setText(QString("%1").arg(qfact));
        write_function(controller, FILTER_QFACTOR, sizeof(qfact), 0, &qfact);
}
```

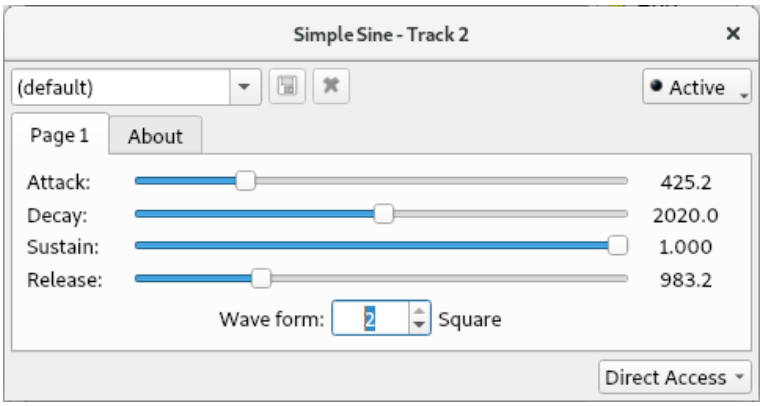
Mindkét esetben először beállítom a használni kívánt paraméter értékét, ezt követően beállítom a megfelelő feliratot, végül továbbítom az értéket a plugin felé `write_function` függvény segítségével.

A grafikus felületnél az `instantiate` függvény példányosítja a `FilterGui`-t, továbbá beállítja a `controller` illetve a `write_function`-t a kapott paraméterek alapján. Valamint ebben a függvényben kapcsolom össze a megfelelő `signalokat` a megfelelő `slotokkal`:

```
        QObject::connect(pluginGui->frequency_dial, SIGNAL(valueChanged(int)),
pluginGui, SLOT(frequencyChanged(int)));



QObject::connect(pluginGui->qfactor_dial, SIGNAL(valueChanged(int)),
pluginGui, SLOT(qfactorChanged(int)));
```



3.2. ábra. *A szintetizátor felülete amit a Qtractor generált*

A `QObject:connect` első paramétere a `signal`, a második maga a `signal`. A harmadik paraméter az értesítendő objektum, végül a negyedik a megfelelő `slot`.

a `port_event` függvényben megvizsgálom hogy a kapott paraméterek megfelelőek-e. Ha megfelelők akkor, megvizsgálom, melyik portból jövő adatot kell beállítanom, amit kerekítéssel a megfelelő `Qdial` értékeként állítok be.

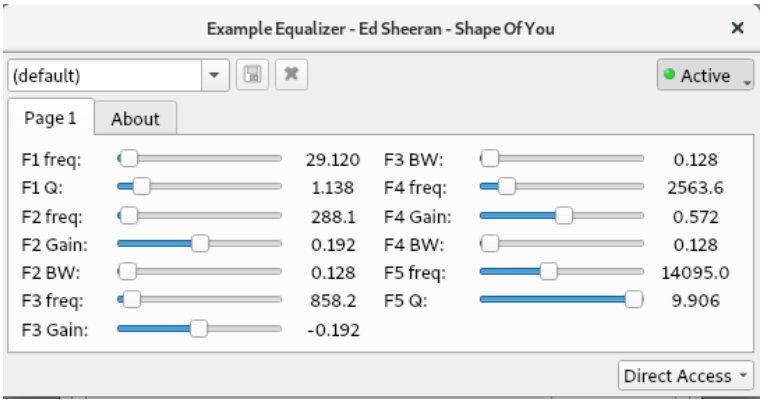
3.5. Modulok fordítása, tesztelése

A modulok a mellékletben a `build.sh`-val fordíthatóak. Ez először törli az előző build eredményeit, ezt követően elkészíti DSPFilterhez a statikus könyvtárat, amelyet az hangszínszabályozó, illetve filter pluginomban használok. Ha ez megtörtént akkor a pluginok könyvtárában található `waf` segítéségével lefordítja a pluginokat, melyek a plugin build könyvtárába kerülnek. A scriptem ezt követően létrehoz egy lvplugins könyvtárat amelybe az elkészült pluginok könyvtárát másolja át. A sikeres fordításhoz a rendszerre telepítve kell lennie az lv2, illetve a qt5 csomagoknak. Miután sikeresen létrejött a könyvtár a pluginokat Qtractorban teszteltem.

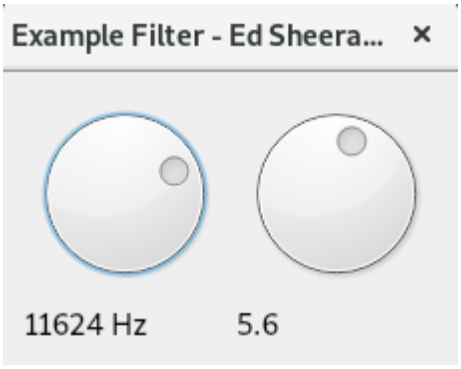
A szintetizátor tesztelése során a program a 3.2. ábrán látható felülettel jelenítette meg az elkészült modult. Mint látható az összes paraméter jelen van a felületen, továbbá a *Wave Form* is a szám mellet kiírja a megfelelő hullámformát. A tesztelés során a felületen beállított hullámformának megfelelő hang szólt. Az ADSR paramétereit változtatva is megfelelő amplitúdó változást tapasztaltam az idő elteltével.

A hangszínszabályzóhoz generált felület a 3.3. ábrán látható. Ezen is jelen van az összes kontroll paraméter amelyet definiáltam. Több különböző zenével is tesztelve a rajta keresztül haladó hang a paraméterek változtatásával a várható hangszínváltozást tapasztaltam.

A filterhez írt grafikus felületet a 3.4. ábra mutatja. A felület az általam definiált módon jelenik meg. A paramétereket változtatva itt is a várható hangszín változást tapasztaltam. Automatizálás során a felület a megfelelő módon jelenítette meg a változásokat.



3.3. ábra. *A hangszínszabályozó felülete amit a Qtractor generált*



3.4. ábra. *A filter általam készített felülete*

4. fejezet

Összefoglalás

Szakdolgazatom elején a linux audio környezet bemutatásával foglalkoztam, melyben bemutattam annak rétegeit, rétegeinek kialakulásának történetét. Továbbá bemutattam, hogyan lehet különböző szoftverek segítégével elindítani a JACK hangrendszert, és milyen eszközök állnak a kapcsolatai szervezésére.

Ezt követően az LV2 pluginok alapvető felépítését szemléltettem, a modul implementációjához szükséges feladatokat részleteztem. Kitértem arra, hogy ezeken a pluginok hogyan lehet betölteni a Qtractor programban.

Végül az általam elkészített szintetizátor-, illetve hangszínszabályozó-modulon keresztül bemutattam hogyan lehet egy hangszer plugint (amely MIDI jelből egy hangot állít elő), illetve effektmodult (amely a bemeneti audiojelet a kimenetén valamilyen formában módosítja) készíteni. Továbbá egy egyszerű szűrő programon bemutattam, hogyan lehet egy modulhoz Qt keretrendszer használatával egyéni grafikus felületet készíteni.

A kitűzött célokat sikeresen megvalósítottam illetve szintetizátor esetén, több funkciót is sikerült implementálni. Ilyen például billentésérzékenység, az ADSR burkológörbe megvalósítása. A munkám során azt tapasztaltam hogy az LV2 kiválóan alkalmas különböző hangszer-, illetve effekt modulok megvalósítására, logikus felépítésű, emiatt könnyen megtanulható.

Irodalomjegyzék

[1] Open Sound System. http://www.opensound.com/oss.html. [Utoljára megtekintve: 2018. november 28.].

[2] Linux Journal - Introduction to sound programming with alsa. https://www.linuxjournal.com/article/6735. [Utoljára megtekintve: 2018. november 28.].

[3] Xruns from the ALSA wiki. https://alsa.opensrc.org/Xruns. [Utoljára megtekintve: 2018. november 28.].

[4] How use PulseAudio and JACK? http://jackaudio.org/faq/pulseaudio_and_jack.html. [Utoljára megtekintve: 2018. november 28.].

[5] Demystifying JACK – A Beginners Guide to Getting Started with JACK. https://libremusicproduction.com/articles/demystifying-jack-%E2%80%93-beginners-guide-getting-started-jack. [Utoljára megtekintve: 2018. november 29.].

[6] [linux-audio-dev] A Plugin API . https://ccrma.stanford.edu/mirrors/lalists/lad/2000/19991108-0307/0676.html. [Utoljára megtekintve: 2018. december 5.].

[7] [linux-audio-dev] LADSPA Version 1 Released . https://ccrma.stanford.edu/mirrors/lalists/lad/2000/Apr/0023.html. [Utoljára megtekintve: 2018. december 5.].

[8] [linux-audio-dev] RFC: Disposable Soft Synth Interface. https://ccrma.stanford.edu/mirrors/lalists/lad/2004/04/0367.html. [Utoljára megtekintve: 2018. december 5.].

[9] [linux-audio-dev] LADSPA2. https://lists.linuxaudio.org/archives/linux-audio-dev/2006-April/015455.html. [Utoljára megtekintve: 2018. december 5.].

[10] Turtle - Terse RDF Triple Language. https://www.w3.org/TeamSubmission/turtle/. [Utoljára megtekintve: 2018. december 5.].

[11] [linux-audio-dev] LV2 Realesed. https://lists.linuxaudio.org/archives/linux-audio-dev/2008-January/019309.html. [Utoljára megtekintve: 2018. december 5.].

[12] Programming LV2 Plugins. http://lv2plug.in/book/. [Utoljára megtekintve: 2018. december 3.].

[13] Qtractor An Audio/MIDI multi-track sequencer. https://qtractor.sourceforge.io/. [Utoljára megtekintve: 2018. december 5.].

[14] ADSR envelope | Libre Music Production. https://libremusicproduction.com/answer/adsr-envelope. [Utoljára megtekintve: 2018. december 6.].

[15] DSPFilters. https://github.com/vinniefalco/DSPFilters. [Utoljára megtekintve: 2018. december 3.].

[16] Legal | Licensing - Qt. https://www.qt.io/licensing/. [Utoljára megtekintve: 2018. december 3.].

[17] Qt Documentation - Signals and Slots. http://doc.qt.io/qt-5/signalsandslots.html. [Utoljára megtekintve: 2018. december 6.].