Desenvolupament d'algoritmes per a síntesi i processament d'audio sota l'estàndard VST

Francisco Javier Ripoll Esteve

Agraïments

Aquest projecte final de carrera ha resultat un sacrifici personal per a mi, ja que aquest any he hagut de fer sonar moltes notes del meu saxòfon. Però també ha sigut un sacrifici per totes aquelles persones que estan al meu voltant, i és per això, que voldria dedicar unes línies d'aquesta memòria a totes les persones que han hagut de suportar-me, que han patit per mi i que m'han ajudat.

Primer de tot, el meu agraïment al tutor del meu projecte, D. Alvar Domenech, perquè sense ell encara estaria pensant què fer en el projecte. Alvar ha sigut el meu llibre durant tot el temps de realització del projecte.

A la meua mare, el meu pare, la meua tia i a Rosabel, que han suportat els mals humors quan les coses no eixien bé, i m'han pegat la llanda per a acabar el projecte.

A Diego, que tantes idees em donava quan tornàvem de Sagunt, i em va fer el favor de gravar-me uns segons de la seua guitarra per posar els exemples del plugins.

També al meu tio Josep Manuel, per haver-me corregit les faltes i expressions ortogràfiques i adequar el text del meu projecte.

Finalment, cal agrair a tota la meua família el suport incondicional que he rebut durant aquests últims sis anys de carrera, i també el fet que m'haja ajudat a aprendre a aprendre.

Doncs a tots, moltes gràcies.

Contingut

1 Introducció

En l'actualitat és moda parlar de multimèdia, però què és realment?

Multimèdia és l'ús de diversos mitjans per a transmetre, administrar o presentar informació. Aquests mitjans poden ser text, gràfiques, àudio i vídeo entre altres. Però amb aquest terme relacionat amb les computadores, ens referim a l'ús de *software* i *hardware* per a emmagatzemar i presentar continguts usant una combinació de text, fotografies, vídeos i àudio.

Aquestes aplicacions tecnològiques són la vertadera novetat, i han fet que el terme multimèdia estiga popularitzat, ja que el fenomen multimèdia està present en quasi totes les formes de comunicació humana.

Com hem esmentat, l'àudio forma part d'aquest popular terme, i és per això, que hi ha una gran quantitat de programes per a la creació, edició i postproducció d'àudio. Una de les més prestigioses empreses de *software* en l'edició i creació musical és Steinberg que, entre els seus productes més populars, té Cubase i Nuendo.

Cubase és un sistema complet de creació musical, ja que podria definir-se com un híbrid entre un seqüenciador Midi, editor d'àudio i mesclador.

Steinberg Cubase no ha aconseguit el nivell de prestigi només per permetre gravar, compondre i mesclar, sinó per disposar d'un ampli conjunt de funcionalitats per a cada tipus de pista. Però la funcionalitat més famosa són els plugins. Aquests plugins, són xicotets programes que s'incrusten en el propi Cubase i tenen dues funcionalitats bàsiques, ja que hi ha dos tipus: els plugins d'efectes i els Instruments Virtuals.

1.2 Motivació

La motivació d'aquest projecte és poder portar a la pràctica la creació de plugins VST per a poder-los utilitzar amb Steinberg Cubase. Els plugins consisteixen en efectes de panorama, talls de freqüència, retards i plugins d'instruments virtuals.

Una vegada parlat amb el professor, el desenvolupament dels plugins VST seran utilitzats amb una finalitat didàctica a manera d'exemple per a assignatures d'àudio, més concretament l'assignatura que s'imparteix en la Facultat d'Informàtica de la Universitat Politècnica de València “Introducció a la Síntesi, Edició i Postproducció d'Àudio”.

1.3 Contingut del cd

El cd amb el que s'adjunta aquest projecte conté una arxiu .pdf amb la memòria del projecte i tres carpetes etiquetades amb els noms: codi, exemples i plugins.

La carpeta etiquetada amb el nom codi conté al seu interior els arxius necessaris per a compilar cada un dels plugins creats durant aquest projecte. En els annexos es pot observar també el codi escrit de cada un d'ells.

A l'interior de la carpeta etiqueta amb el nom exemples, hi ha una sèrie d'arxius d'àudio en format .mp3 per a escoltar de forma sonora l'efecte que produeix cada plugin en una mostra d'àudio.

Finalment a la carpeta plugins, estan els arxius .dll corresponents a cada plugin elaborat per a poder-los utilitzar en una estació de treball que suporte l'estàndard VST.

2 Conceptes previs

2.1 El so digital

2.1.1 Representació del so mitjançant PCM

El maneig del so des d'un computador necessita una representació en format digital. Hi ha moltes tècniques que s'adapten a les necessitats de cada usuari. A la figura 1 es pot observar l'esquema bàsic del flux de l'àudio digital.

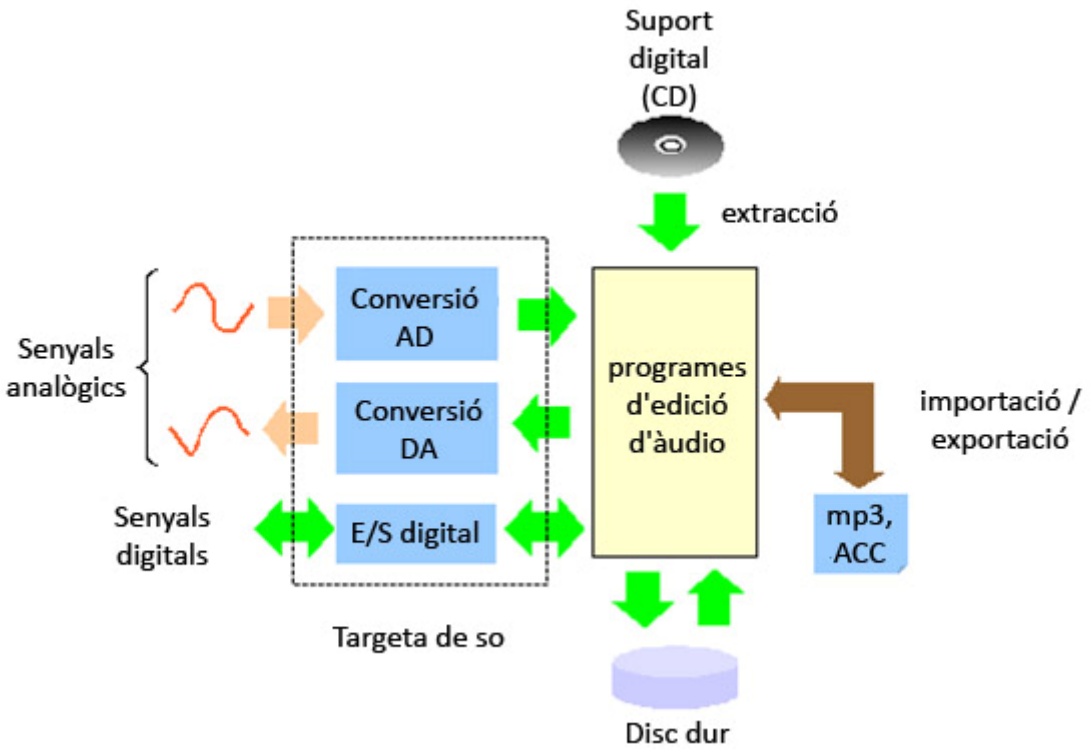

figura 1. Flux de l'àudio digital.

Per a la realització d'aquest projecte, s'utilitza el mètode de la modulació per impulsos codificats (PCM) per poder fer l'intercanvi amb els perifèrics d'entrada/eixida d'àudio.

El mètode de la modulació per impulsos codificats (PCM), és el mètode per a representar digitalment un senyal analògic quan la magnitud del senyal és mostrejada periòdicament a intervals uniformes, i llavors quantificades a un valor (normalment binari). Aquest mètode ha sigut utilitzat en sistemes de telefonia digital i en teclats musicals electrònics. També és la forma estàndard d'àudio digital i vídeo digital als ordinadors.

2.1.2 Mostratge

El mostratge consisteix a prendre mostres del valor del senyal *n* vegades per segon, amb la qual cosa que tindrem *n* nivells de tensió en un segon.

A la figura 2 es pot observar la quantificació per a l'ona sinusoïdal (corba roja) mitjançant PCM. L'ona sinusoïdal es mostra a intervals regulars (eix de les X). Cada mostra és

quantificada mitjançant l'algoritme triat en l'eix de les Y (en aquest cas la funció *floor*). D'aquesta manera, obtenim així una representació totalment discreta del senyal d'entrada com es pot observar en l'àrea ombrejada, i així és molt fàcil codificar aquestes dades digitals per al seu emmagatzemament i manipulació.

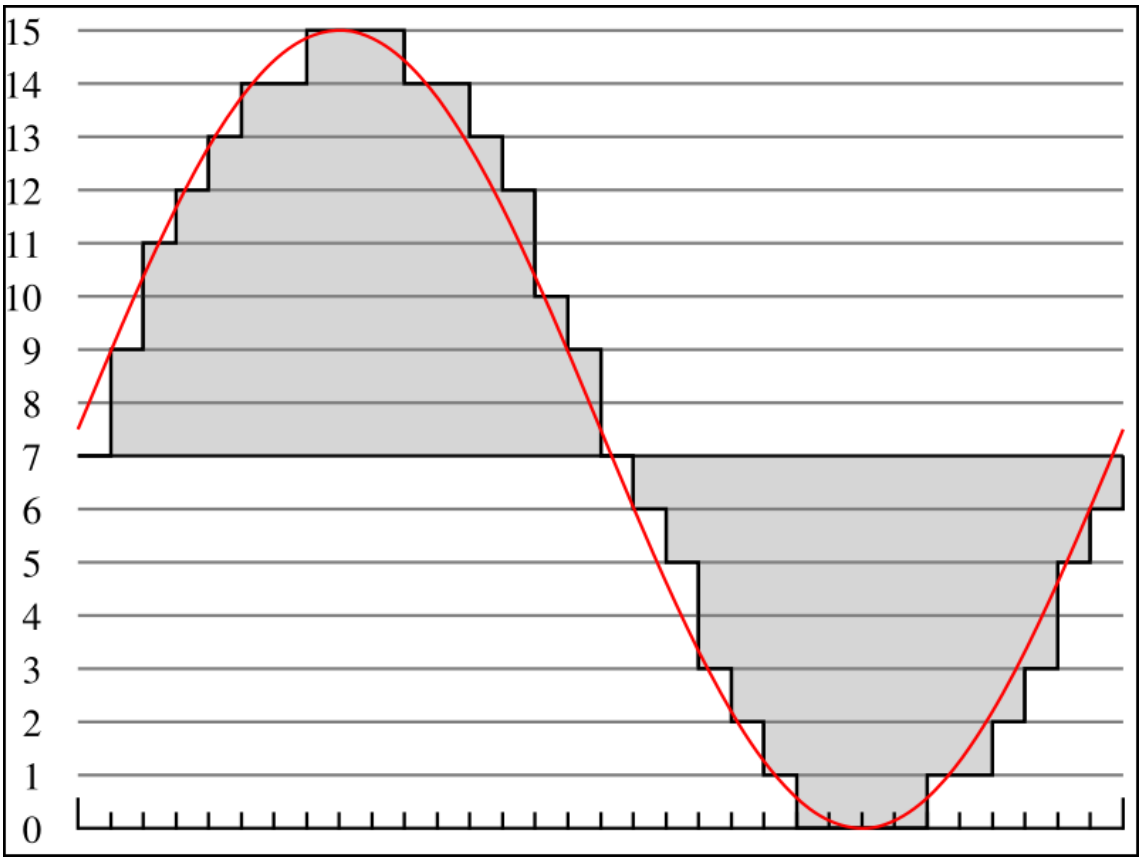

figura 2. Mostreig i qüantificació d'un senyal (roig) de 4 bit PCM

Amb l'exemple de la figura, comprovem que els valors de quantització de cada moment serien: 7, 9, 11, 12, 13, 14, 14, 15, 15, 15, 14, etc. Codificant aquests valors al computador amb valors binaris quedaria de la manera següent: 0111, 1001, 1011, 1100, 1101, 1110, 1110, 1111, 1111, 1111, 1110, etc, i per tant ja podrien ser processats al computador.

Si ens n'adonem, el nombre de mostres que es poden prendre en un minut pot variar segons les necessitats. Per a determinar el nombre de mostres a prendre per segon s'utilitza el teorema de mostratge de Nyquist-Shannon.

El teorema de Nyquist-Shannon a grans trets diu que si es prenen mostres d'un senyal elèctric continu a intervals regulars i amb una freqüència doble a la freqüència màxima que es vullga mostrejar, aquestes mostres contindran tota la informació necessària per a reconstruir el senyal original. A la taula 1, es poden observar les freqüències de mostratge típiques:

taula 1. Freqüències de mostreig típiques	
PER ÀUDIO	
8000 mostres/s	Telèfons. Adequat per a la veu humana però no per a la reproducció musical.
22050 mostres/s	Ràdio. En la pràctica permet reproduir senyals amb components de fins a 10 kHz.
32000 mostres/s	Vídeo digital en format miniDV.
44100 mostres/s	CD. En la pràctica permet reproduir senyals amb components de fins a 20 kHz. També comú en àudio en formats MPEG-1 (VCD, SVCD, MP3).
47250 mostres/s	Format PCM de Nippon Columbia (Denon). En la pràctica permet reproduir senyals amb components de fins a 22 kHz.
48000 mostres/s	So digital utilitzat en la televisió digital, DVD, format de pel·lícules d'àudio professional i sistemes DAT.
50000 mostres/s	Primers sistemes de gravació d'àudio digital de finals dels 70 de les empreses 3M i *Soundstream*.
96000 o 192400 mostres/s	HD DVD. Àudio d'alta definició per a DVD i BD-ROM (Blu-ray Disc).

2.1.3 Quantificació

La quantificació consisteix a assignar un determinat valor discret a cada un dels nivells de tensió obtinguts en el mostratge. Però hi ha el problema que les mostres poden tenir un infinit nombre de valors en la gamma d'intensitat de la veu, i per tant, el que es fa és aproximar al valor més pròxim d'una sèrie de valors predeterminats.

Per tant, s'aplica una codificació a cada nivell de quantificació, i s'assigna un codi binari distint, d'aquesta manera el senyal codificat està llest per a ser transmès. Per exemple, en la telefonia, el senyal analògic vocal amb un amplada de banda de 4kHz es converteix en un senyal digital de 64Kbps.

2.1.4 Estructura d'una interfície d'àudio

Per a poder utilitzar la conversió analògica-digital necessitem un convertidor que dispose d'una entrada analògica i una eixida digital. L'entrada analògica ha d'acceptar un senyal elèctric d'àudio dins d'un rang nominal determinat, com per exemple de -0.5 a +0.5 volts. Aquest convertidor deu mostrejar la tensió regularment sincronitzat amb un rellotge, que normalment oscil·la a una freqüència típica de 44.1 kHz, 48 kHz o 96 kHz i posteriorment és codificada en binari amb una resolució de 16 o 24 *bits*.

Per a la conversió digital-analògica es realitza l'operació inversa al convertidor analògic-digital, ja que en aquest cas l'entrada és un senyal digital d'àudio codificat en PCM amb una freqüència i resolució binària donades i amb l'eixida analògica.

Amb l'entrada/eixida de l'àudio digital, hi ha estàndards de transmissió d'àudio digital codificat en PCM, que són útils per a comunicar en temps real fonts, processadors i sistemes d'emmagatzemament de l'àudio digital. Aquests estàndards especifiquen la comunicació digital sèrie de punt a punt de forma unidireccional. Per tant, els equips d'àudio poden disposar d'entrades i eixides digitals que s'adhereixen a aquests estàndards ja que els dispositius que es connecten disposen dels seus propis rellotges interns.

Les interfícies s'adapten al computador de dues maneres:

- Algunes s'adapten al bus d'expansió del computador, com el PCI o els seus derivats. L'amplada de banda d'aquests busos és alt com per exemple el bus PCI-eXpress que actualment supera els 4 GB/s, i permeten un alt nombre de canals d'entrada i eixida.
- Altres s'adapten a un bus d'entrada/eixida estàndard. Actualment, conforme el rendiment dels busos sèrie creix, aquestes interfícies s'estan difonent. Permeten treballar amb ordinadors de sobretaula i portàtils. Les interfícies menys exigents, amb dues o quatre canals a 48 kHz/24 *bits*, utilitzen USB 2, però quan el nombre de canals és major, el bus apropiat és Firewire 400 Mbps o 800 Mbps.

L'estructura general d'una interfície es pot veure a la figura 3, de la qual podem observar les diferents parts:

- Condicionadors de senyal d'entrada: hi ha uns preamplificadors i atenuadors que ens permeten ajustar el senyal.
- Convertidor d'analògic-digital (*AD*) i digital-analògic (DA).
- Un adaptador d'entrada/eixida (E/S) per a connectar els convertidors al bus. L'adaptació es fa directament al bus d'expansió del sistema com per exemple PCI o a un bus de perifèric com USB o Firewire.
- Controlador dedicat a l'accés directe a la memòria

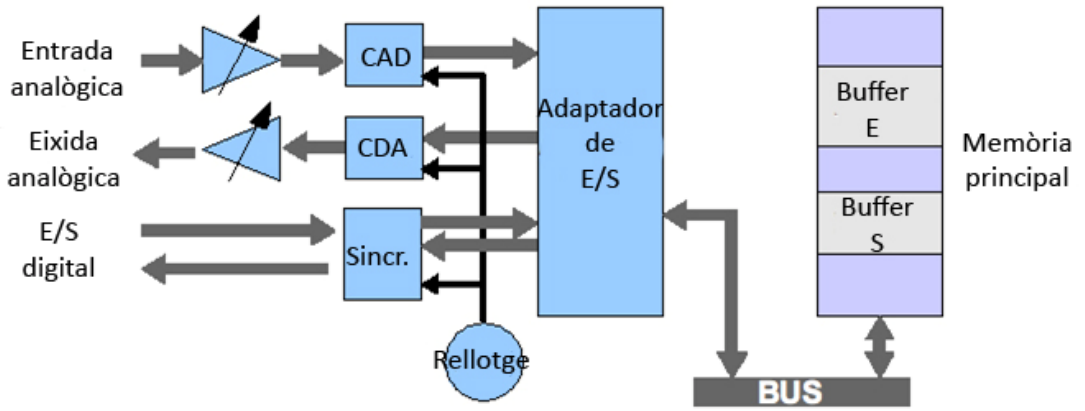

figura 3. Estructura d'interficie d'audio

A dia d'avui és estrany trobar un computador personal sense un sistema de so. Aquests sistemes de so majoritàriament són de baixa qualitat ja que les targetes de so per a un treball més seriós amb àudio han de complir uns requisits determinats:

- Les conversions analògic-digitals i digital-analògiques han de fer-se fora del xassís del computador per a evitar els intensos camps electromagnètics creats dins d'ell.
- Els connectors per als senyals analògics han de ser estàndard i de bona qualitat (*Jack* 1/4", RCA o XLR)
- Els connectors per a senyals digitals han de ser estàndard.
- No és imprescindible, però sí recomanable per a l'ús corrent el fet de disposar d'un amplificador per a auriculars.

figura 4. Connectors estàndars. Jack 1/4", RCA i XLR

Els connectors per als sistemes de so del computador personal tenen una sèrie de codi en forma de color per a saber quina és la seua funció, els quals es poden veure a la taula 2.

taula 2. Tipus de color per als connectors

Verd TRS 3.5 mm	Eixida estèreo, canal davanter.
Negre TRS 3.5 mm	Eixida estèreo, canal posterior
Gris TRS 3.5 mm	Eixida estèreo, canal lateral
Daurat TRS 3.5 mm	Eixida doble, centre i *subwoofer*
Blau TRS 3.5 mm	Entrada estèreo, *line* level
Rosa TS 3.5 mm	Entrada micròfon mono

A continuació vorem diversos exemples de sistemes de so:

- **Sound Blaster X-Fi elit pro**: presa per a guitarra, micròfon i auriculars amb controls de volum independents. Entre altres característiques s'inclou una preamplificació phono

per a la gravació directa de vinils, amb filtre d'àudio, E/S MIDI i E/S òptica i digital coaxial.

figura 5. Sound Blaster X-Fi elit pro

- **MOTU 896HD:** Connexió *Firewire*. Vuit preamplificadors de micròfon amb "*phantom power*" (48v) i pera amb control de voltatge i vuit canals d'entrades/eixides òptiques ADAT a 24 *bits*, 44.1/48kHz - Quatre canals a 88.2/96kHz.

figura 6. MOTU 896HD

2.2 MIDI

MIDI són les sigles de *Musical Instrument Digital Interfície* (Interfície Digital d'Instruments Musicals). MIDI es tracta d'un protocol estàndard a nivell industrial creat perquè computadores, sintetitzadors, seqüenciadors, controladors i altres dispositius musicals electrònics puguen comunicar-se i compartir informació per a poder generar sons.

Aquest protocol defineix diversos tipus de dades per a poder controlar quina nota s'està polsant en un teclat MIDI, fins i tot la intensitat amb què es prem una tecla del teclat.

2.2.1 Història

El protocol MIDI naix de les exigències d'interconnectar sintetitzadors digitals, ja que a l'inici va haver-hi un problema d'incompatibilitat amb els sistemes que usava cada companyia que els fabricava. És per això que era necessari crear un llenguatge comú per a totes les companyies.

La primera persona que proposà aquest estàndard a l'*Àudio Engineering Society* va ser Dave Smith, president de la companyia *Sequential Circuits* en 1981. Més tard, el 1983 es publicà la primera especificació MIDI.

Molts dels formats d'àudio es basen en MIDI, ja que aquests arxius són molt compactes perquè amb tan sols 10 KB d'informació podem produir un minut complet de música, a causa que l'arxiu només emmagatzema instruccions.

MIDI només transmet esdeveniments i missatges controladors, on es poden interpretar de manera arbitrària, segons la programació de cada dispositiu. El protocol MIDI es pot entendre com una partitura, la qual conté instruccions en valors numèrics sobre quan s'ha de generar cada nota de so i les característiques que ha de tenir, com la durada, la intensitat, etc. A l'aparell on s'envie aquesta informació, serà l'encarregat de transformar-la en música audible.

Açò és un gran avantatge per a les aplicacions amb telèfons mòbils, ja que els tons de telefonada poden ocupar molt poc. No obstant això, pot ser un desavantatge per a altres aplicacions, ja que la informació no és capaç de garantir una forma precisa de l'ona que escoltarà l'oient pel fet que cada sintetitzador MIDI té els seus propis mètodes per a produir el so MIDI.

En l'actualitat, quasi totes les gravacions musicals són compatibles amb dispositius MIDI. A més, MIDI també s'utilitza per a controlar el *hardware*, incloent dispositius de registre i mòduls d'efectes de so, així com les actuacions en directe amb la fase de llums i alguns tipus de pedals d'efectes digitals. MIDI permet que els ordinadors, sintetitzadors, controladors MIDI, targetes de so, *samplers* i *drums machines* puguen controlar-se els uns als altres, i a més intercanviar dades amb el sistema.

2.2.2 *Hardware*

Els dispositius MIDI segons el seu ús, la configuració del programa o programes que estiguen fent ús, poden estar rebent i enviant informació. El dispositiu que envia missatges d'activació es denomina Mestre (*Master*), i el que respon a la informació es denomina Esclau (*Slave*).

Els aparells MIDI es poden classificar en tres grans blocs:

- **Controladors**: són els encarregats de generar els missatges MIDI com l'activació o desactivació d'una nota, les variacions de to, etc. El controlador MIDI que més es pot observar en el mercat és el que té forma de teclat de piano, ja que és el més utilitzat pels compositors per a compondre obres musicals. No obstant això, hi ha gran quantitat d'instruments construïts amb capacitat de transmissió via interfície MIDI com poden ser guitarres, sets de percussió, saxos, i molts més. En la figura 7 es mostren els controladors MIDI amb diferents formes d'instruments.

figura 7. Controladors mid: Set de percussió, trompeta, guitarra i saxo.

- **Unitats generadores de so**: o més conegudes com a mòduls de so, que són els encarregats de rebre els missatges MIDI i transformar-los en senyals sonors.
- **Seqüenciadors:** són aparells destinats a gravar, reproduir o editar missatges MIDI. Poden veure's en format *hardware*, *software* o incorporar-se en un sintetitzador.

Encara que aquests siguen els tres tipus d'aparells, en el mercat n'hi ha d'altres que reuneixen dues o tres de les funcions descrites, com per exemple els orgues electrònics, els quals disposen d'un controlador en forma de teclat de piano i una unitat generadora de so, fins i tot alguns models inclouen un seqüenciador.

Els cables MIDI utilitzen un connector de tipus DIN de 5 *pins* o contactes com es pot observar a la figura 8. El *pin* número 5 és l'únic que s'utilitza per a la transmissió de les dades. Els pins número 2 i 4 s'utilitzen com a blindatge i per a transmetre una tensió de +5 volts, per a així poder assegurar-se que l'electricitat flueix en la direcció desitjada. Els pins restants, que són els número 1 i 3 es van reservar per a afegir funcions en el futur.

figura 8. Connector Midi

El cable MIDI serveix per a poder transmetre dades entre dos dispositius o instruments electrònics.

Actualment, fabricants com Casio, Korg i Roland, uns dels més populars, estan substituint aquests cables per cables de connectors de tipus USB, ja que són més fàcils de trobar en els comerços i permeten una connexió fàcil amb els computadors personals.

MIDI només pot transmetre senyals en un sentit, la qual cosa es denomina un funcionament *simplex*. El sentit d'aquestes és sempre des del dispositiu mestre al dispositiu esclau, on el primer genera la informació i el segon la rep.

Un aparell MIDI pot tenir fins a tres tipus de connectors:

- MIDI OUT: és el connector del qual ixen els missatges generats pel dispositiu mestre a un dispositiu esclau.
- MIDI IN: serveix per a introduir els missatges rebuts des del dispositiu mestre al dispositiu esclau
- MIDI THRU: és un connector d'eixida, encara que la seua funció és enviar una còpia exacta dels missatges rebuts pel connector MIDI *IN*.

La forma més simple de connectar dos dispositius MIDI seria unir-los amb un cable MIDI. En un extrem en el connector MIDI OUT del dispositiu mestre (per exemple un teclat MIDI) i a l'altre extrem en el connector MIDI *IN* del dispositiu esclau (per exemple un sintetitzador). D'esta manera els missatges ixen del dispositiu mestre pel connector MIDI OUT i aturaran al connector MIDI *IN* del dispositiu esclau, com es pot veure a la figura 9.

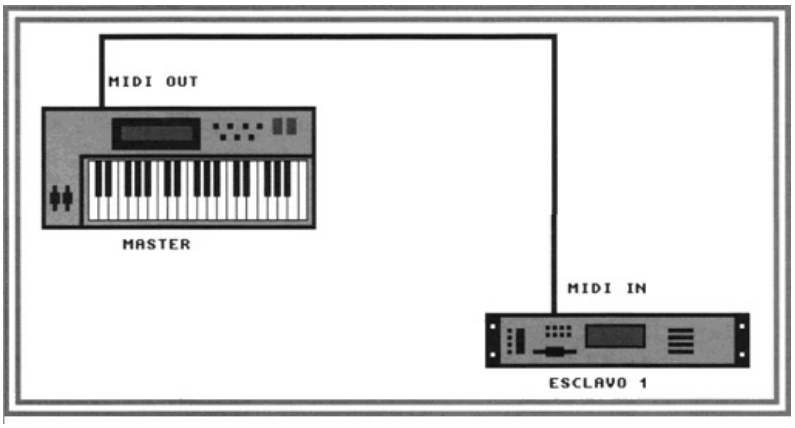

figura 9. Connexió MIDI Master-Slave

2.2.3 *Software*

MIDI, en la seua especificació inclou una organització dels *bytes*.

El *byte* MIDI està compost per deu *bits*, a fi que els dispositius puguen portar un compte de tots els *bytes* enviats i rebuts. El primer *bit* és el d'inici i sempre és 0. L'últim *bit* és el *bit* de terminació i sempre és 1. Els 8 *bits* restants que es troben entre el *bit* d'inici i el *bit* de terminació són els que contenen el missatge MIDI.

Es pot fer una diferència dels tipus de missatge, ja que aquests poden ser d'estat (Estatus *byte*) o d'informació (data *byte*). Per a poder diferenciar-los, només hem de mirar el primer *bit* del missatge MIDI, i si és 1, el *byte* del missatge és d'estat; si és 0, tenim un *byte* de dades.

Per norma general, quan es crea un missatge MIDI, sempre s'envia un *byte* d'estat, el qual pot estar seguit d'una certa quantitat de *bytes* de dades. Per exemple, si s'envia el missatge d'estat "activar nota", aquest ha d'anar seguit d'un *byte* de dades informant quina nota és activada.

Els missatges d'estat poden ser de dos tipus: missatges de canal o missatges de sistema. Els missatges de canal només s'envien a un dispositiu específic, mentre que els missatges de sistema s'envien a tots els dispositius. A la taula 3 tenim una llista amb tots els missatges disponibles.

Taula 3. Missatges MIDI

***Byte* estat**	**Descripció**	***Byte* estat**	**Descripció**
1000cccc	Desactivació de nota	11110101	*Indefinit*
1001cccc	Activació de nota	11110110	Requeriment d'entonació
1010cccc	Postpulsació polifònica	11110111	Fi de missatge exclusiu
1011cccc	Canvi de control	11111000	Rellotge de temporització
1100cccc	Canvi de programa	11111001	*Indefinit*

1101cccc	Postpulsació monofònica de canal	11111010	Inici
1110cccc	*Pitch*	11111011	Continuació
11110000	Missatge exclusiu del fabricant	11111100	Aturada
11110001	Missatge de trama temporal	11111101	*Indefinit*
11110010	Punter posició de cançó	11111110	Espera activa
11110011	Selecció de cançó	11111111	Reset del sistema
11110100	*Indefinit*		

Els *bytes* que els últims quatre bits estan marcats com “cccc”, es refereixen a missatge de canal, la resta de *bytes* són missatges de sistema.

MIDI està pensat per a comunicar mitjançant un mateix mitjà de transmissió un únic controlador amb diverses unitats generadores de so, on cada una pot tenir un o més instruments. És per això que fa falta un mètode per a diferenciar cada un dels instruments, que els denominarem canal.

Els canals MIDI poden també denominar-se veus o instruments. MIDI pot direccionar fins a 16 canals, i per tant, en instal·lar el sistema MIDI serà necessari assignar un nombre de canal per a cada dispositiu.

Existeixen 128 instruments en l'especificació estàndard de MIDI, també coneguts com GM o “*General MIDI*”. Es poden veure a la taula 4.

Taula 4. Instruments MIDI

00 - Piano de cua acústica	32 - Baix acústic	64 - Saxo soprà	96 - Efecte 1 (pluja)
01 - Piano acústic brillant	33 - Baix elèctric polsat	65 - Saxo alt	97 - Efecte 2 (banda sonora)
02 - Piano de cua elèctrica	34 - Baix elèctric punteig	66 - Saxo tenor	98 - Efecte 3 (cristalls)
03 - Piano de cantina	35 - Baix sense trasts	67 - Saxo barítón	99 - Efecte 4 (atmosfera)
04 - Piano Rhodes	36 - Baix colpejat 1	68 - Oboè	100 Efecte 5 (brillantor)
05 - Piano amb "chorus"	37 - Baix colpejat 2	69 - Sanguinyol anglès	101 Efecte 6 (donyets)
06 - Clavicordi	38 - Baix sintetitzat 1	70 - Fagot	102 Efecte 7 (ressons)
07 - Clavinet	39 - Baix sintetitzat 2	71 - Clarinet	103 Efecte 8 (ciència-ficció)
08 - Celesta	40 - Violí	72 - Flautí	104 Sitar
09 - Carilló	41 - Viola	73 - Flauta	105 Banjo
10 - Caixa de música	42 - Violoncello	74 - Flauta dolça	106 Shamisen
11 - Vibràfon	43 - Contrabaix	75 - Flauta de pa	107 Koto
12 - Marimba	44 - Cordes amb trèmolo	76 - Coll de botella	108 Kalimba
13 - Xilòfon	45 - Cordes amb *pizzicato*	77 - Shakuhachi (flauta japonesa)	109 Gaita
14 - Campanes tubulars	46 - Arpa	78 - Xiulet	110 Violí celta
15 - Saltiri	47 - Timbals	79 - Ocarina	111 Shanai
16 - Orgue Hammond	48 - Conjunt de corda 1	80 - Melodia 1 (ona quadrada)	112 Campanetes
17 - Orgue percusiu	49 - Conjunt de corda 2	81 - Melodia 2 (dent de serra)	113 Agogó
18 - Orgue de *rock*	50 - Cordes sintetitzades 1	82 - Melodia 3 (orgue de vapor)	114 Caixes metàl·liques
19 - Orgue d'església	51 - Cordes sintetitzades 2	83 - Melodia 4 (xiuxiueig orgue)	115 Caixa de fusta
20 - Harmònium	52 - Cor Aahs	84 - Melodia 5 (xaranga)	116 Caixa Taiko
21 - Acordió	53 - Veu Oohs	85 - Melodia 6 (veu)	117 Timbal melòdic

22 - Harmònica	54 - Veu sintetitzada	86 - Melodia 7 (cinquenes)	118 Caixa sintetitzada
23 - Bandoneón	55 - Èxit d'orquestra	87 - Melodia 8 (baix i melodies)	119 Platet invertit
24 - Guitarra espanyola	56 - Trompeta	88 - Fons 1 (nova era)	120 Trasteig de guitarra
25 - Guitarra acústica	57 - Trombó	89 - Fons 2 (càlid)	121 So de respiració
26 - Guitarra elèctrica (*jazz*)	58 - Tuba	90 - Fons 3 (polisintetitzador)	122 Platja
27 - Guitarra elèctrica (neta)	59 - Trompeta amb sordina	91 - Fons 4 (cor)	123 Piulada d'ocell
28 - Guitarra elèctrica (apagada)	60 - Sanguinyol francès (trompa)	92 - Fons 5 (d'arc)	124 Timbre de telèfon
29 - Guitarra saturada (overdrive)	61 - Secció de bronzes	93 - Fons 6 (metàl·lic)	125 Helicòpter
30 - Guitarra distorsionada	62 - Bronzes sintetitzats 1	94 - Fons 7 (celestial)	126 Aplaudiment
31 - Harmònics de guitarra	63 - Bronzes sintetitzats 2	95 - Fons 8 (granerets)	127 Tir de fusell

Els sintetitzadors contenen elements de generació de so denominats veus. Per a poder assignar les veus se segueix un procés algorítmic per a poder encaminar les dades d'activació/desactivació de nota des del teclat a cada una de les veus. D'esta manera, es pretén que les notes s'executen correctament i amb la temporització exacta.

Per a poder implementar-ho, ha de definir-se una relació entre els setze canals MIDI i l'assignació de les veus del sintetitzador, per a això existeixen 4 tipus de missatges de mode per a aquest propòsit que es poden veure a la taula 5. Els dits missatges poden ser *omni on* / *omni off*, *poly* i *mono*. *Poly* i *mono* són mútuament excloents. *Omni*, quan està actiu, l'instrument respondrà a missatges en qualsevol canal MIDI, al contrari, quan aquest desactivat, l'instrument respondrà només a un canal en particular (modes 2 i 3) o cada veu serà assignada a un canal en particular (mode 4).

Per a un receptor assignat al canal bàsic N, els quatre modes possibles que sorgeixen dels dos missatges de mode són:

Taula 5. Modes de funcionament MIDI

Número	**Nom**	**Descripció**
1	*Omni on / poly*	Els missatges de veu es reben en tots els canals de veu i s'assignen a les veus polifònicament.
2	*Omni on / mono*	Els missatges de veu es reben en tots els canals de veu i controlen només una veu monofònicament.
3	*Omni off / poly*	Els missatges de veu es reben en el canal de veu N només, i s'assignen a les veus polifònicament.
4	*Omni on / poly*	Els missatges de veu es reben en els canals de veu des de N fins a N+M-1, i s'assignen monofònicament a les veus 1 a N, respectivament. El nombre de veus M està especificat en el tercer *byte* del missatges de manera *mono*.

2.3 Seqüenciador

Un **seqüenciador** és un dispositiu electrònic físic o una aplicació informàtica que permet programar i reproduir esdeveniments musicals de forma seqüencial mitjançant una interfície de control físic o lògic connectat a un o més instruments musicals electrònics. L'interfície de control més estès és l'estàndard MIDI.

El seqüenciador és una de les ferramentes bàsiques de la Informàtica Musical ja que permet la creació d'unes quantes pistes melòdiques, harmòniques o rítmiques, que poden ser tractades, editades i reproduïdes de forma individual o simultània.

A banda de l'afinació, durada i posició de les notes, hi ha molts altres paràmetres que els seqüenciadors ens permeten accedir, com ara volum, efectes, so, etc. tot això sempre amb la possibilitat de tractar no sols la pista en conjunt sinó cada una de les seues notes de forma individual.

figura 10. Finestra principal Cubase

És important recalcar que els seqüenciadors no tenen sons propis, sinó que utilitzen els de la Targeta de So. El que el seqüenciador fa és informar la Targeta de quan ha de reproduir una nota, que volum sonarà, quant durarà, etc. Per tant la qualitat del so no depèn del Software (seqüenciador) sinó del Hardware (Targeta)

Hi ha infinitat de seqüenciadors, fins i tot alguns "especials" per a determinats tipus de música com ara pot ser techno o dance. Tots ells tenen en compte la possible falta de coneixements musicals de l'usuari, i per això, mostren la música amb diferents llenguatges gràfics que no són l'estrictament musical, com són les tecles d'un piano, o diferents tipus de gràfics.

2.3.1 Orígens

Encara que l'origen del seqüenciador és eminentment electrònic es pot dir que un dels seqüenciadors més bàsics i antics que es coneixen és la caixa de música. A partir dels xicotets pivots que sobreïxen d'un corró giratori es fan sonar una sèrie de llengüetes metàl·liques afinades segons l'escala musical. Per tant la seqüència es troba "gravada" en el corró giratori, que realment és una transcripció de la partitura en forma de dades (cada pivot correspondria a un esdeveniment). Anàlogament al funcionament de la caixa de música trobem la pianola, encara que en aquest cas la seqüència es troba en el cartó perforat.

Els primers seqüenciadors electrònics van aparèixer en els anys 1970 i eren analògics, igual que els primers sintetitzadors. L'interfície que utilitzaven era l'anomenat CV/Gate o control per voltatge i consistia a enviar impulsos de corrent continu amb un nivell de tensió en

funció de l'altura de la nota. Eren molt limitats i ardus de programar; només permetien controlar un aparell al mateix temps i tenien pocs compassos disponibles. L'aparició del sistema MIDI en 1983 i l'avanç de la tecnologia digital en matèria musical van suposar una vertadera revolució. Va ser en aquest punt quan van començar a comercialitzar-se els primers seqüenciadors digitals (físics) i programes seqüenciadors per a ordinadors, que han anat guanyant complexitat i prestacions amb el pas dels anys.

Els seqüenciadors actuals mes populars són Cubasis, Music Studio, MicroLogic, Cubase, Cakewalk i Logic entre alters.

2.3.2 Funcionament

El funcionament bàsic d'un seqüenciador en forma d'aplicació informàtica passa per una secció principal on es visualitzen totes les pistes, on cada pista correspon a un so o a una font sonora externa i els paràmetres que les afecten de forma global com el volum, l'entonació, el panorama, o el canal MIDI, així com els controls de reproducció (*play*, *stop*, *loop*, *tempo*, etc) i les funcions bàsiques de copiar i apegar, mute (silenci), moure parts, fusionar parts, etc. Pel que fa a l'edició trobem diverses seccions, sent les més rellevants el corró de piano, la partitures i la llista d'esdeveniments.

- **Corró de piano**: És la forma d'edició més estesa. Consisteix en un teclat virtual adjunt a una quadrícula on es representen els compassos, al mateix temps estan subdividits en una quantitat preestablida (blanques, negres, corxeres, semicorxees, etc). La forma de compondre és dibuixant literalment les notes, variant l'altura i la durada sobre la quadrícula. Com s'aprecia el funcionament és molt semblat al d'una pianola.
- **Partitures**: Més estesa en usuaris de formació clàssica i acadèmica. Com el nom indica, la seua forma d'edició és sobre un pentagrama amb la simbologia pròpia del solfeig. És precís dir que mentre es va component, independentment de la forma d'edició, el programa crea automàticament la partitura corresponent.
- **Llista d'esdeveniments**: És el propi codi font de programació. És útil a l'hora de modificar puntualment algun tipus d'esdeveniment en concret, sobretot, els que no afecten a l'altura o a la durada de la nota (aftertouch, modulació, portamento, nivells d'efecte, etc.)

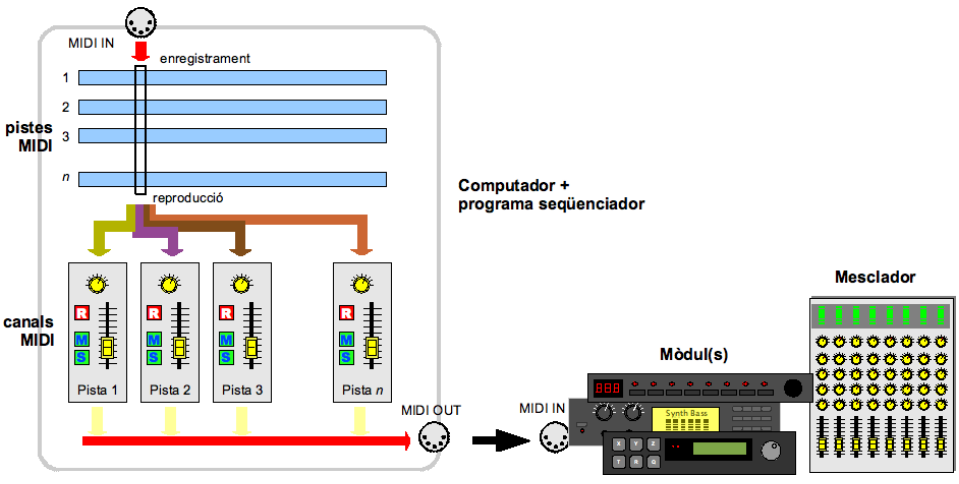

figura 11. Esquema del ús d'un seqüenciador

2.4 DAW

La traducció literal de "*Digital Àudio Workstation*" és: **Estació de treball per a àudio digital**, el qual és un sistema electrònic dissenyat per a gravar, editar i reproduir àudio digital. Una de les característiques clau de les DAWs és l'habilitat de manipular lliurement sons gravats, semblant a un processador de textos mecanografiats que es poden manipular les paraules.

Fabricants i desenvolupadors usen també el terme Estudi (Studio) per a dir-nos que la seua aplicació es tracta d'un entorn on realitzar treballs basats en l'àudio digital.

El terme "DAW" es refereix simplement a una combinació de *software* d'àudio multipista i un *hardware* d'àudio d'alta qualitat. Aquest últim és una unitat de conversió de senyals d'àudio d'analògic-digital i/o digital-analògic. Per exemple, una estació de treball pot tenir vuit entrades d'àudio, i dues o més eixides d'àudio per als monitors o acarrerament del senyal a altres dispositius. Els sistemes poden tenir només un parell d'entrades i eixida mono. Un convertidor professional d'analògic-digital té la mateixa funció que una targeta de so, però generalment de major qualitat.

2.4.1 Història

La primera estació de treball per a àudio digital va ser desenvolupada per Soundstream a finals dels setanta, utilitzant un equipament digital PDP-11 en una minicomputadora i executant un *software* personalitzat anomenat "DAP" (Processador d'àudio digital) per a l'edició digital d'àudio i efectes de so. Un emmagatzemament d'oscil·loscopi que estava connectat a la minicomputadora actuava com a pantalla per a mostrar la forma de l'ona d'àudio.

figura 12. PDP-11

En 1981, l'enginyer de gravació Roger Nichols construí una estació de treball d'àudio digital amb el seu propi disseny, utilitzant un bus S-100. Aquest computador estava basat amb Micropolis 8" amb un disc dur de 32 MB per a l'emmagatzemament de les dades de l'àudio

digital. Roger Nichols va usar aquest sistema durant la gravació i producció de l'àlbum de Donald Fagen anomenat “The Nightfly” en 1982.

A finals dels vuitanta, els ordinadors Apple Macintosh i Commodore Amigal van començar a tenir un gran consum perquè tenien suficient poder per a manejar la tasca d'edició d'àudio digital. Macromedia *Soundedit* va ser el primer *software* d'edició d'àudio que va aparèixer per al Macintosh en 1986, però el concepte es va fer popular per una empresa anomenada Digidesign, que en 1987 va presentar un dels primers equips i paquets de *software* per a la computadora personal d'Apple que també servia per a l'edició d'àudio. El *software* va ser originàriament anomenat Sound Tools i més tard va ser anomenat Pro Tools. Aquest va ser el predecessor de la nova indústria estàndard del sistema Pro Tools.

A partir d'aqueix moment, la majoria d'estacions de treball per a àudio digital es basen en Apple Macintosh (Pro Tools, Studer Dyaxis, Sonic Solutions). Per a versions de Microsoft Windows va començar a partir de l'any 1992, per empreses com Soundscape que més tard va adquirir Mackie, després SSL, i mes tard Sadie i Spectral Syntesis. Tots els sistemes en aquest moment utilitzen un *hardware* dedicat al processament d'àudio.

En 1994, una empresa anomenada OSC, junt amb els tècnics de Digidesign, va llançar una edició d'un gravador i editor de quatre pistes, amb un *software* anomenat DECK II. El primer *software* basat en Windows va ser producte d'Àudio Workshop (SAE) i era capaç de gravar, editar i mesclar quatre pistes d'àudio. Es va convertir en una eina molt usada en les emissores de ràdio dels EE.UU.

2.4.2 Actualitat

Músics i compositors han tingut sempre un desig d'integrar equips estèreo, giradiscos, aparells de gravació, teclats MIDI i fins i tot les guitarres elèctriques amb els ordinadors. Ordinadors potents basats en eines de composició van començar a aparèixer amb l'Atari ST i Amiga. Entusiasmats, van continuar cercant eines per a la creació de tasques d'àudio una miqueta més integrades, més fàcils d'usar i d'alt rendiment.

Quasi qualsevol ordinador personal amb un *software* amb multipista i edició pot funcionar un poc com un DAW, ja que el terme es refereix als sistemes informàtics que tenen un hardware amb una alta qualitat externa de convertidor d'analògic-digital i digital-analògic, i algun tipus de software d'àudio com poden ser: Cubase, Nuendo, Logic Pro, Pro Tools, Adobe Audition, Sadie 5, Sony Sound Forge, Samplitude, soundscape, SONAR, *ACID* Pro, FL Studio (anteriorment Fruityloops), Ableton Live, Tracktion o Digital Performer, alguns dels quals són de lliure *softwar*e com Audacity i Ardour. A més de necessitar targetes de so d'alta gamma, es requereix també una gran quantitat de memòria RAM, CPU ràpida i lliure i de suficient espai en el disc dur.

Són diverses eines enquadrades en una aplicació central en què solen estar presents: el mesclador o taula de mescles, seqüenciador, caixes de ritmes, etc.. Per norma general, solen admetre l'ampliació dels seus components per mitjà de plug-ins. Aquests té un format tal com VST, DX (Microsoft) o RTAS (Pro Tools).

2.4.3 Varietats

Les DAWs generalment es poden trobar en dues varietats:

- **Computadors basats en DAW**: aquests consten de tres components: un computador, un convertidor analògic-digital i digital-analògic i un *software* d'edició d'àudio digital. La computadora actua com un amfitrió de la targeta de so i el *software*, i proporciona una potència de processament per a l'edició de l'àudio. La targeta de so actua com una interfície d'àudio, normalment amb la conversió de senyals analògics d'àudio a format digital. El *software* controla els dos components de *hardware* i proporciona una interfície d'usuari per a permetre la gravació i edició.
- **Sistemes DAW integrats**: Consistirà en una taula de mescles, una superfície de control, un convertidor d'àudio i un emmagatzemament de dades tot en un sol dispositiu. Els sistemes Integrat DAWs eren més populars abans que les computadores personals foren prou poderoses per a executar *software* DAW. Com la potència dels ordinadors està en augment i el preu està disminuint, la popularitat dels costosos sistemes integrats es va reduint.

2.4.4 Edició lineal i no lineal

En l'àrea de la multimèdia, s'entén com *a mitjà lineal* aquell que és equivalent a una cinta (d'àudio o de vídeo) en el qual no es pot reordenar el material en el temps ni inserir nous continguts sense esborrar altres continguts previs. Tot això afecta la manera de treballar amb el mitjà. Per exemple, l'edició en els mitjans lineals no ofereix la possibilitat de desfer l'última operació, ni de replicar continguts en posicions diferents de la cinta.

En el cas particular de l'àudio, els mitjans lineals són normalment cintes. Aquest sistema d'emmagatzemament proporciona una o més pistes, cada una d'elles equivalent a una cinta monoaural. Les pistes estan disposades de forma paral·lela en la cinta i passen per damunt d'un conjunt de tants capçals magnètics com a pistes. Es pot observar a la figura 13.

figura 13. Tascam de 16 pistes

L'accés és simultani a totes elles, encara que cada pista admet gravació o reproducció independent. La gravació d'una pista no afecta les altres, però esborra el contingut previ de la pista en qüestió. En el vocabulari usual de l'àudio, es diu que una pista està *armada* quan el

capçal corresponent està preparat per a gravar. A la figura 14, es pot vore un esquema de les pistes d'una cinta i el seu pas per damunt dels capçals.

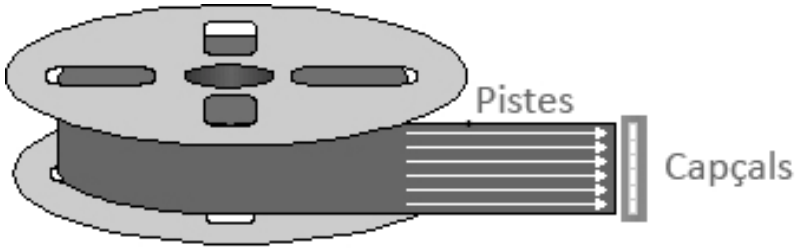

figura 14. Pistes d'una cinta

A la figura 15 a la es pot vore una foto dels capçals d'una unitat *Studer* de 24 pistes.

figura 15. Unitat Studer de 24 pistes

Els *mitjans no lineals*, generalment basats en un disc dur amb un sistema d'arxius, ofereixen una interfície semblant a la cinta, perquè donen la mateixa sensació de seqüència temporal; però en realitat és la imatge construïda per l'entorn de treball a partir d'una estructura de dades que permet accés directe i flexible als continguts. En un mitjà no lineal, l'edició és *no destructiva*: la reordenació i la replicació del material són operacions senzilles i l'esborrament és reversible. En els sistemes moderns de producció d'àudio i de vídeo, el medi és no lineal.

Centrant-nos en el treball amb àudio, la pista continua sent una unitat per a les operacions de reproducció i gravació del sistema. La implementació d'una pista s'aconsegueix mitjançant un o més arxius d'àudio, dins dels quals es defineixen regions o segments. La pista és una *cue list* o llista de referències a les regions. La figura 16 il·lustra la idea: en una pista apareixen sis fragments distints numerats, un d'ells repetit. La regió 1 correspon a un arxiu complet *A*, la resta correspon a diversos fragments distribuïts en tres arxius *B*, *C* i *D*. Observa com la mateixa regió 3 apareix dues vegades a la pista, i com les regions 5 o 6 se solapen parcialment a l'arxiu *D*. La resta de la pista és silenci.

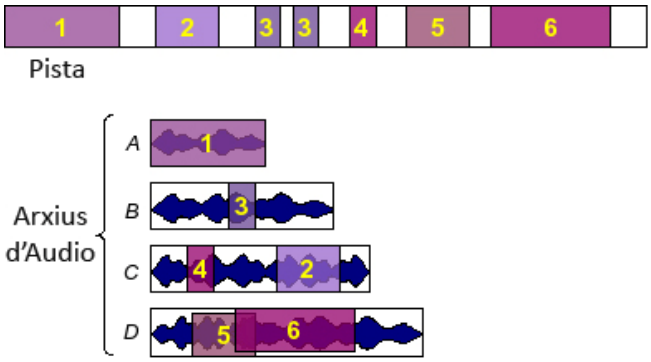

figura 16. Representació de una pista d'audio

Normalment, els arxius contenen material monoaural o estereofònic, però els formats d'arxiu més utilitzats poden contenir un nombre arbitrari de canals de so. Les pistes també poden ser mono o estèreo; l'associació de pistes a arxius pot tenir més o menys restriccions segons l'aplicació.

Quan el sistema té diverses pistes, la reproducció i gravació del so és simultani en totes elles. El sistema manté un cursor de temps i els comandaments de tipus magnetòfon que el controlen s'anomena *transport*.

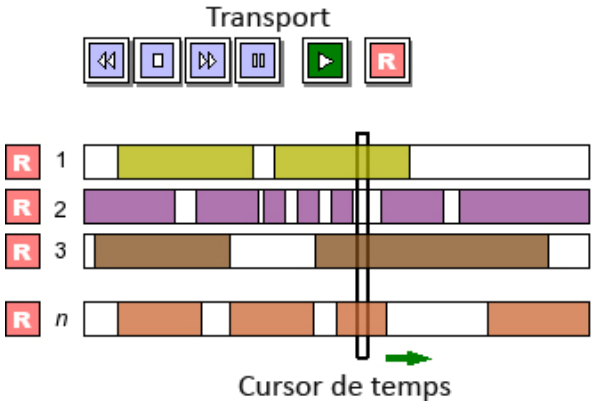

figura 17. Representació del tranport i el cursor de temps

Associada a cada pista, està el comandament per a armar-la (*Record enabler*). Perquè es puga inserir nou contingut a una pista, aquesta ha d'estar armada i el botó de gravació general activat. La gravació d'una pista crea un nou arxiu amb una única regió ubicada en el punt de temps corresponent.

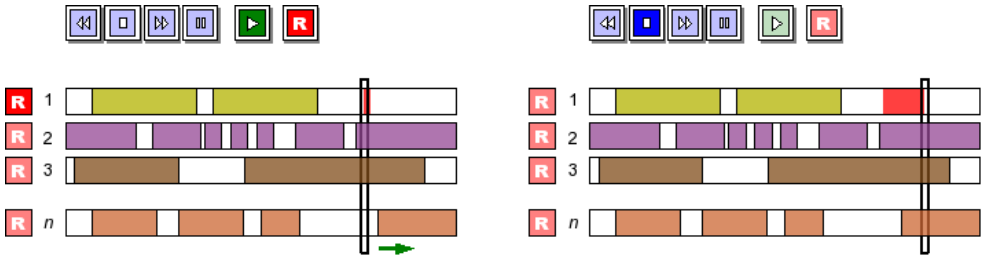

figura 18. Representació de la grabació

2.4.5 Gestió de les regions

La DAW ha d'oferir una sèrie d'operacions bàsiques sobre pista:

- Definir el marc de començament i la longitud de cada regió dins de l'arxiu d'àudio que la conté
- Ubicar regions a les pistes, replicar-les i esborrar-les
- Crear transicions: *fade-in*, *fade-out* i *cross-fade* sobre les regions seleccionades

Per a la creació de transicions, una DAW genera nous fragments d'àudio aplicant *fade-in*, *fade-out* i *crossfade*s a les regions disposades en una pista. Suposa que en una pista s'han disposat dues regions 1 i 2 (cada una definida en un arxiu) que se solapen (figura 19).

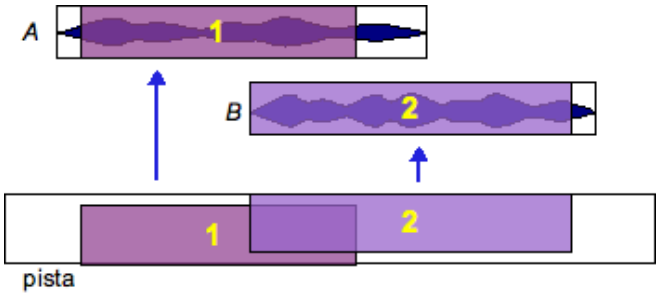

figura 19. Exemple de dues regions solapades

Les transicions aplicables seran:

- Un *fade-in* al principi de la regió 1
- Un *cross-fade* entre les regions 1 i 2, a la zona d'intersecció
- Un *fade-out* al final de la regió 2

La DAW ajusta les regions existents i crea automàticament unes altres de noves on s'emmagatzemen les transicions creades. En molts casos, les transicions s'emmagatzemen juntes en un únic arxiu d'àudio. Aquest arxiu no sol ser estàndard, sinó que conté els fragments d'àudio codificats en el format PCM intern de la DAW, amb mostres codificades en coma flotant. El resultat final de les transicions es pot observar a la figura 20.

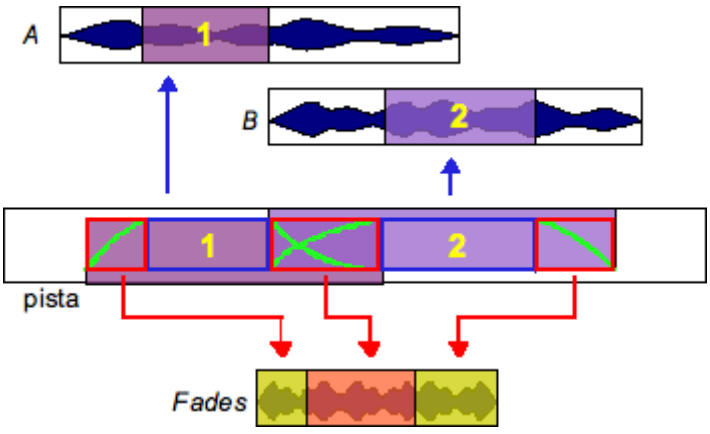

figura 20. Resultat final

2.4.6 Acarreament del senyal

2.4.6.1 Canals de pista

Un canal de pista és un element de comandament sobre una pista que conté:

- Entrada i eixida. Una entrada pot ser una pista o un terminal d'entrada de la interfície.
- Elements fixos de control: equalització, panorama estereofònic i comandament de volum d'eixida (*fader*).
- Elements variables de control o inserits. Un inserit es pot instanciar dins d'un conjunt d'efectes disponibles dins de l'aplicació per al control de dinàmica, retards, reverberació i altres.
- Interruptors de *solo* i *mute*, per a l'escolta selectiva de les pistes.
- Un selector de mode que permet armar la pista per a gravació.

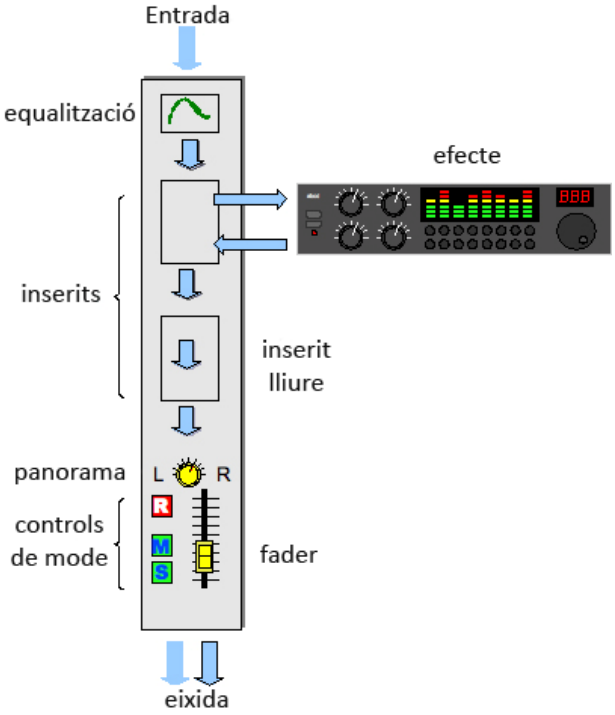

figura 21. Canal de pista

2.4.6.2 Una DAW simple

Combinant una interfície de dos canals d'entrades i dos d'eixida amb canals de pista s'obté una DAW simple amb un nombre arbitrari de pistes apta per a produir material estereofònic. El nombre de canals d'entrada de la interfície limita el nombre de pistes que poden gravar-se simultàniament.

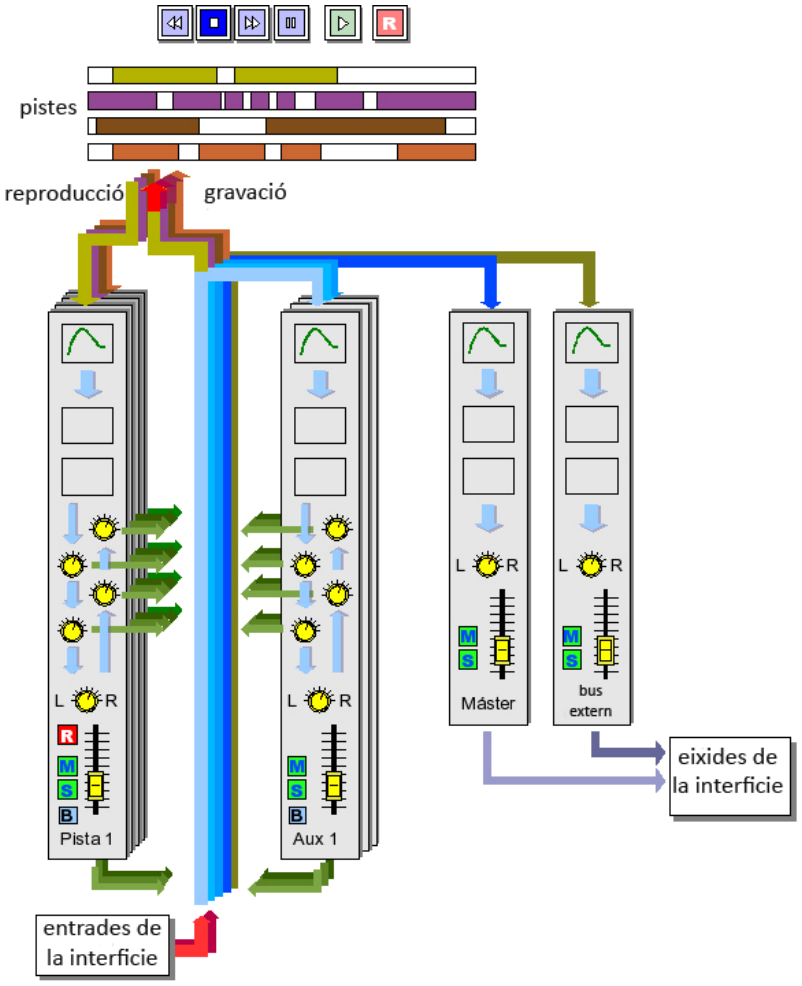

figura 22. DAW simple

L'eixida dels canals de pista es connecten a un *bus* comú d'eixida. En el context de l'àudio, un bus és un dispositiu de mescla que crea el senyal resultant de sumar les eixides dels canals que se li connecten. El bus alimenta les entrades d'un canal d'eixida denominat *Màster*. Si ens adonem el canal màster no inclou control de mode de gravació, perquè no està associada a una pista concreta.

El treball usual amb aquesta DAW inclouria la gravació dels fragments sonors dins de les pistes, ajustament de transicions, inserció d'efectes, elecció de volums i panorames per a cada pista i l'obtenció de la mescla final. Aquesta última operació (*Mix-down* o *Bounce*) produeix un arxiu d'àudio de format estàndard que conté una còpia del so enviat a la interfície.

2.4.6.3 Busos i enviaments

Al sistema simple anterior li falta flexibilitat. La seua limitació més important és que només és capaç de produir una mescla. Des dels orígens de l'electrònica d'àudio, els enginyers de so han valorat que els seus sistemes de producció disposaren d'uns quants busos (com més, millor) alternatius al màster. Els busos poden ser interns o disposar d'eixides a l'exterior, però tots tenen en comú que permeten una mescla de senyals.

La presència dels busos afecta el disseny del sistema de canals:

- Els comandaments han de permetre la selecció del bus on se suma l'eixida del canal.

- Fan falta canals auxiliars, alimentats pel bus, per a controlar la mescla resultant.
- Cal incloure enviaments. Un enviament (o *send*) és una bifurcació que permet enviar una còpia del senyal a un bus seleccionat amb un guany regulable.

Un posible aspecte general d'un canal en una DAW amb busos es pot observar a la figura 23.

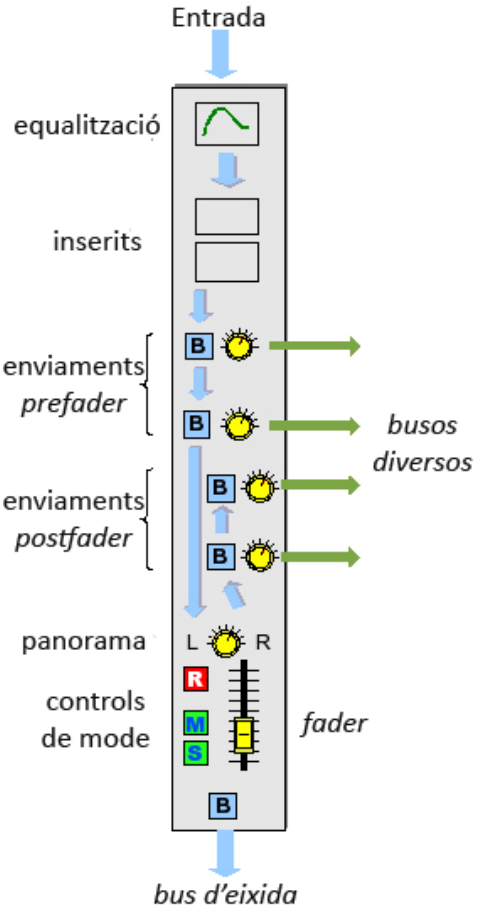

figura 23. Canal de una DAW amb busos

L'eixida del canal i els enviaments inclouen un selector B que permet especificar el bus destinatari del so. La quantitat de senyal que s'envia al bus d'eixida es regula mitjançant el *fader*. En els enviaments *prefader*, la quantitat de senyal depèn només del comandament de volum associat a l'enviament. En els enviaments *postfader*, la quantitat de senyal depèn de dos comandaments: del *fader* i el volum d'enviament.

2.4.6.4 Una DAW amb busos

Cada fabricant defineix les possibilitats d'acarrerament del seu DAW. La figura següent pretén mostrar una fórmula general de partida. Totes les DAW tenen, almenys, tres tipus de canals:

- Canals associats a una pista. Disposen de comandament de gravació R a partir d'una entrada de la interfície o d'un bus intern. Disposen d'inserits i d'eixides i enviaments a busos seleccionables.

- Canals auxiliars, no associats a cap pista, també amb inserits i amb eixides i enviaments a bus. No disposen de comandament de gravació R.
- Canals d'eixida, amb inserits. Com que van associats a una eixida física, no tenen selector B de bus a l'eixida ni tampoc comandament de gravació R. Normalment, no tenen enviaments. El canal Màster és un cas particular de canal d'eixida.

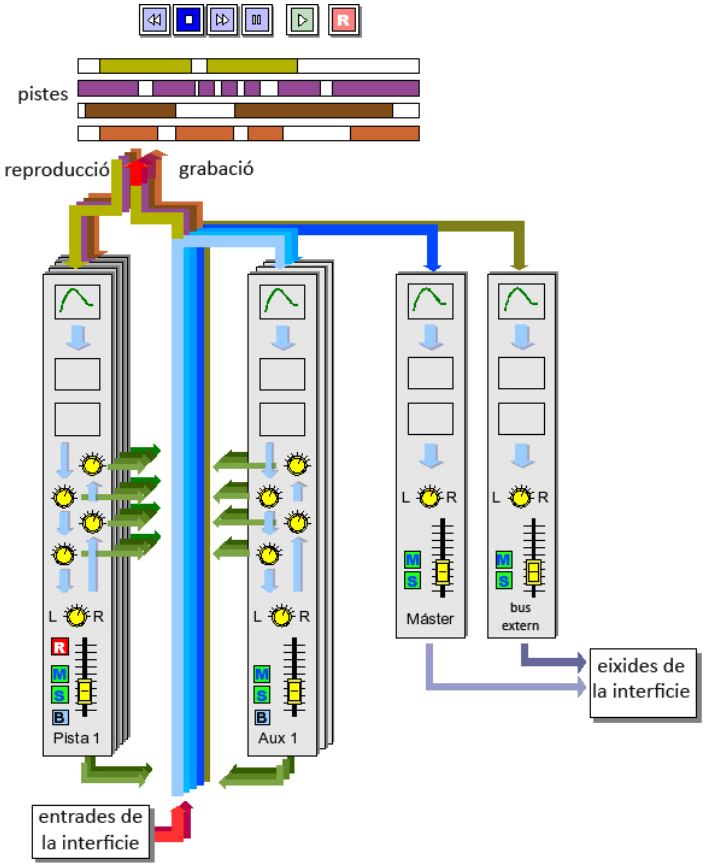

figura 24.Una DAW amb busos

2.4.6.5 Esquemes bàsics d'acarrerament amb busos

Hi ha efectes, com la reverberació, que afigen so al senyal d'entrada. La forma corrent de situar la reverberació és com a inserit en un canal auxiliar alimentat per un bus intern. Per a afegir reverberació a una pista, basta d'enviar senyal pel bus. El canal d'efectes aplicarà a la mescla rebuda els efectes que tinga inserits i afegirà el resultat a la mescla.

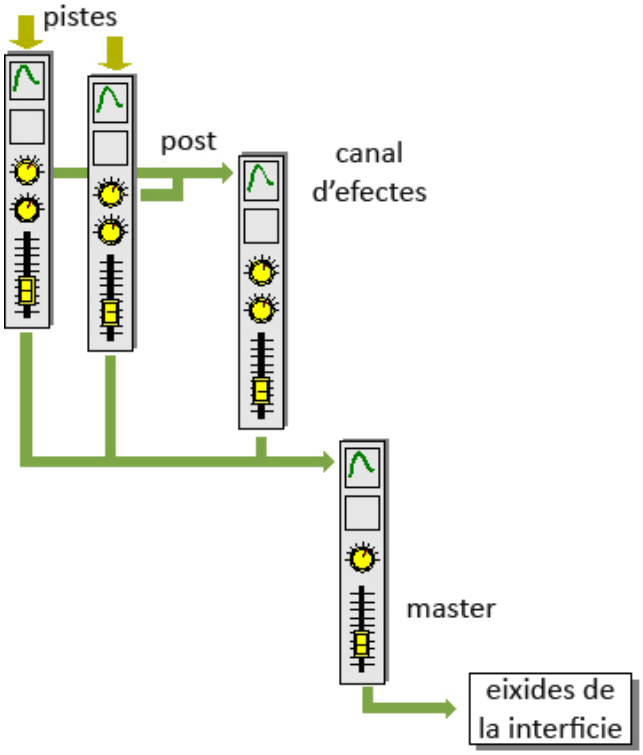

figura 25. Aplicació del canal d'efectes

Els enviaments als canals d'efectes solen ser de tipus *postfader,* perquè s'entén que la quantitat de senyal enviada des del canal de pista ha de ser proporcional al *fader*.

A vegades és necessari escoltar una mescla distinta per raons de producció o per tocar en directe. Per exemple, pot haver-hi una pista de referència que du el temps que no apareix en la mescla principal, que els músics que estan gravant han de sentir durant la gravació d'altres pistes. El procediment més còmode consisteix a triar un bus amb eixida a exterior i enviar-li so des de les pistes d'interès. Per exemple, a un baixista que estiga gravant un cançó, només amb sentir el ritme de la bateria li serviria, ja que sols li interessa el anar-hi acompanyat. Els enviaments, ara hauran de ser *prefader*.

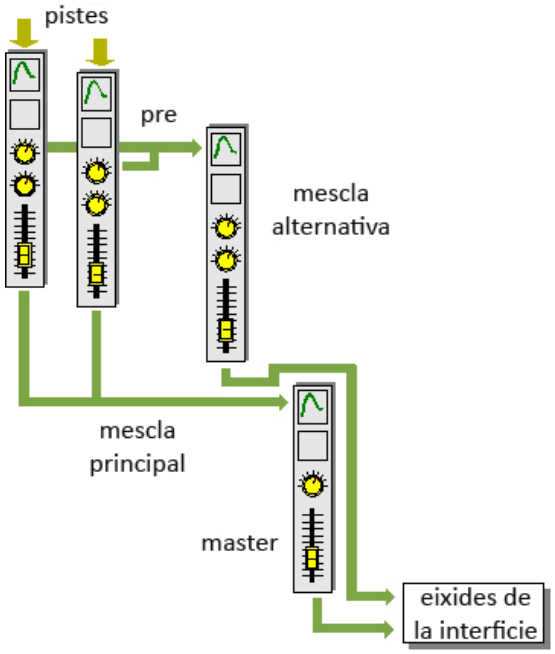

figura 26. Mescla alternativa

Quan hi ha moltes pistes que han de ser controlades en comú, pot ser convenient agrupar-les mitjançant busos. Les pistes amb les veus dels figurants en una escena de pel·lícula, les pistes de la bateria d'un grup de *rock,* la secció de vent d'un grup de música, tots ells es troben en aquest cas. Per a formar un grup, es dirigeix l'eixida dels canals de pista afectats cap a un bus intern amb el seu canal auxiliar.

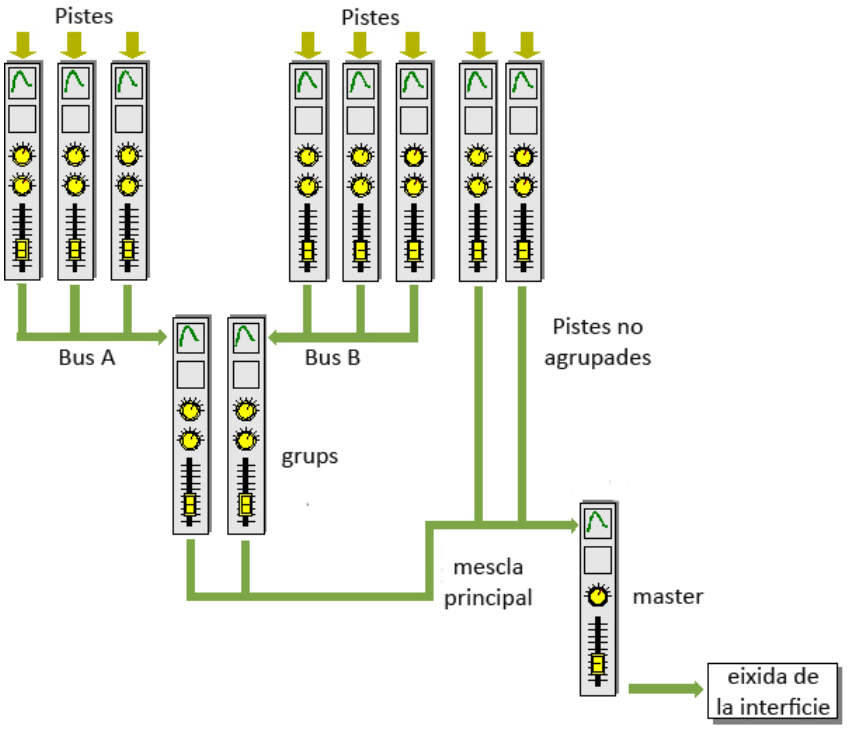

figura 27. Grups de pistes

D'aquesta manera, hi ha dos nivells de control: a cada pista es pot aplicar equalització i efectes individuals, a més del propi volum dins del grup, i al grup se li poden aplicar efectes comuns i volum general.

3. Entorn de treball

3.1 Cubase

Cubase és un programa per a l'ordinador de producció i gravació musical. El programa ofereix la gravació, producció i mescla de sons a fi de fer la música per a la seua distribució en CD o Internet. Cubase, com hem explicat abans és una estació de treball d'àudio digital.

Cubase va iniciar la seua vida a finals dels 80 com un seqüenciador i editor MIDI. El programa, originàriament, va ser desenvolupat per a l'Atari_ST. Més tard es va fer la versió per a Apple Macintosh i a mitjan dels anys 90 per a Microsoft Windows.

El Cubase original emprava un sistema operatiu anomenat MROS (MIDI Real-estafe Operating System) que corria davall el propi sistema operatiu nadiu. Permetia executar diverses aplicacions MIDI a l'ordinador i passar les dades entre aquestes a temps real. El MROS no treballava bé en Windows 3.0, ja que aquest no va ser previst per a aplicacions en temps real. No obstant això, els sistemes operatius moderns estan dissenyats per a suportar aplicacions multimèdia, així que les versions actuals de Cubase no usen MROS.

El llançament de Cubase Àudio en 1991 per a l'Atari Falcon va obrir una bretxa en la tecnologia de programes DSP posant a l'abast domèstic, la manipulació d'àudio en temps real sense la necessitat de costoses targetes addicionals, com era el cas del costós Pro Tools i altres sistemes semblants.

El funcionament bàsic és el següent:

Tenim una pantalla amb unes quantes pistes (àudio, MIDI, Bateries, Pistes mestres...). A les pistes d'àudio podem gravar qualsevol tipus d'àudio (guitarres, veu, pianos...) controlar el seu volum, panorama, equalització i efectes.

Uns dels avantatges d'aquest programa és que qualsevol modificació que fem sobre la gravació és no destructiva, és a dir les gravacions que estan en les diferents pistes mai no es modifiquen, sinó que en qualsevol cosa que hi fem Cubase calcula en temps real el resultat per a reproduir-la. Així, qualsevol efecte o equalització que fem sempre serà totalment reversible, i ens permet anar provant els milers d'efectes dels quals disposem, un a un, o molts al mateix temps sense haver de preocupar-nos per no destruir la nostra preada gravació original.

A més, comporta més avantatges en calcular el resultat en temps real, ja que si volem gravar un tros d'àudio i després repetir-lo en un altre moment de la cançó, simplement tallem la gravació en trossos tot utilitzant copiar/apegar, i no crea un nou arxiu d'àudio, sinó que en temps real llig de l'original.

A les pistes de MIDI, el que es grava és so de format MIDI, és a dir que es grava com si fora una partitura. Com ja hem explicat anteriorment, ofereix el gran avantatge que l'arxiu ocuparà molt poc espai, però té el desavantatge que és un sistema altament dependent del *hardware* que s'usarà per a reproduir les partitures.

En tractar-se de partitures, amb Cubase no cal disposar d'un instrument MIDI per a gravar-les, sinó que les podem compondre nosaltres mateixos a mà ja que per a això, Cubase disposa d'un Editor de Patrons, semblant al que seria compondre amb pentagrames, però amb una filosofia radicalment distinta, més senzilla i més intuïtiva.

Si Steinberg ha aconseguit el nivell de prestigi que té és per una cosa: Cubase no es limita només a gravar, compondre i mesclar, sinó que disposa d'un ampli conjunt de funcionalitats per a cada tipus de pista, i una d'elles és la que ha fet famós a aquest producte: Els *plugins*.

Els *plugins* són xicotets programes que s'incrusten en el propi Cubase. Hi ha dos tipus: Els *plugins* d'efectes i els instruments virtuals.

Els *plugins* d'efectes són exclusius per a les pistes d'àudio, i amplien els set d'efectes que podem inserir a una pista d'àudio. Cada *plugin* és totalment configurable. Per exemple, un *plugin* de distorsió per a guitarra podem configurar el to, el guany, entrada, eixida, etc... Per a un de reverberació, la distància, la grandària de la sala "virtual"... Cal dir que hi ha milions de *plugins* i molts en són realment espectaculars. Aquests es troben per Internet i molts són comercials, encara que hi ha altres gratuïts.

Els *plugins* d'instruments virtuals són, com el seu propi nom indica, instruments virtuals. Suposem que hem creat una pista MIDI, i hi fem una partitura per a un violí i no tenim cap sintetitzador MIDI capaç de reproduir un violí. Llavors, podem instal·lar un *plugin* que siga un violí virtual i escoltaríem el violí. Igual que per al violí, hi ha instruments musicals per a la bateria, baix, piano…

Per a nosaltres Cubase és un producte realment meravellós i excepcional, amb la gran virtut de ser apte tant per a professionals com per a principiants, al contrari d'altres programes com Protools, considerat el millor, però molt difícil de manejar i de requeriments tècnics de *hardware* molt exigents i cars. Cubase funciona amb qualsevol targeta de so i amb qualsevol versió de Windows, encara que és altament recomanable usar Windows 2000 o XP.

figura 28. Interfície del Cubase

3.2 VST

Steinberg's Virtual Studio Technology (Tecnologia d'estudi virtual) o també VST és un interfície per a la integració de software d'àudio i *plugins* d'efectes amb els editors d'àudio i sistemes de gravació. VST utilitza tecnologies semblants a les del Processament Digital de Senyals per a poder simular un estudi tradicional de gravació.

Un VST és un programa de *software* que s'executa en una aplicació que suporta aquesta tecnologia. A aquesta aplicació se l'anomena VST Host. Alguns exemples d'aquests són Cubase, Logic Pro o Sonar. Hi ha milers de Plugins VST que són recolzats per aquestes aplicacions.

Els VSTs tenen la capacitat de processar i generar àudio, com també interactuar amb interfícies MIDI. Els VST's que processen àudio s'anomenen efectes VST, i els encarregats de generar àudio són els anomenats VSTi mitjançant VST instrument.

VST està disponible per als sistemes operatius Windows i Mac OS. Com que els arxius que es creen són arxius binaris i són dependents de la plataforma on s'executen, un VST compilat per a Mac OS no funcionarà amb Windows i viceversa. Amb els sistemes Linux, poden utilitzar-se els arxius compilats amb Windows emprant el programa Wine.

Steinberg's VST SDK és un conjunt de classes C++ basades en un entorn API de C++. El SDK es pot descarregar des de la seua pàgina web. A més, s'ha desenvolupat Steinberg VST GUI, que és un altre conjunt de classes C++, que pot emprar-se per a construir la interfície gràfica. Hi ha classes per a botons, lliscadors, etc. Cal tenir en compte que en tractar-se de programació de baix nivell, l'aspecte i la imatge és creada pel *plugin* fabricant.

3.2.1 VST plugins

Amb Hardware i drivers adequats, com per exemple una targeta de so que suporta ASIO, VST *plugins* poden ser utilitzats en temps real. ASIO ofereix a Windows una baixa latència. Hi ha tres tipus de plugins VST:

- **Instruments VST**: és un *plugin* VST que genera àudio. Generalment, ja siga virtual o *samplers* sintetitzadors.
- **Efectes VST**: és un *plugin* VST que s'utilitza per a processar una entrada d'àudio, com per exemple l'efecte del *delay* o el *chorus*. La majoria de *hosts* permeten múltiples efectes encadenats. Molts d'aquests efectes estan disponibles com a descàrregues gratuïtes o de compra per tota la xarxa.
- **Efectes MIDI VST**: és un *plugin* VST que s'utilitza per a processar missatges MIDI per a l'acarrerament de les dades MIDI entre altres instruments o maquinari VST. Per exemple per a transportar o per a crear arpegis.

3.2.2 Classe AudioEffect

Un efecte d'àudio és una instància de la classe AudioEffectX, que és una extensió de la classe base AudioEffect. El perfil és el següent:

```
AudioEffectX::AudioEffectX(audioMasterCallback audioMaster, VstInt32
numPrograms, VstInt32 numParams)
```

Al perfil es pot observar que se li ha de preveure el nombre de paràmetres (numParams) i programes (numPorgrams) que tindrà el nostre efecte. Per exemple, l'efecte de distorsió sol tenir dos paràmetres, on es controla el valor de llindar per a produir la distorsió i el guany.

El programa és un *preset*, és a dir, un cas particular de valors dels paràmetres que té una utilitat concreta. Per exemple, es pot definir el programa "net" per a tocar la guitarra sense distorsió, i per tant el valor de llindar estaria en el màxim perquè no haguera distorsió; després es pot definir el programa "distorsió" on el valor de llindar estiga en un valor mínim perquè existisca la distorsió.

Al perfil es pot observar diversos elements com:

- **audioMaster**: és una funció que la passa l'hoste que s'encarregarà del diàleg amb l'efecte.
- **VstInt32:** és el tipus numèric sencer de 32 bits declarat per conveniència per a fer el codi independent de la plataforma.

Per a començar a programar un efecte VST basta d'implementar els mètodes de la classe AudioEffectX que es mostren a continuació:

```
Class Efecte : public AudioEffectX {
	public:
	Efecte (audioMasterCallback audioMaster);
	~Efecte ();
	//---from AudioEffect-----------------------
	virtual void processReplacing (float** inputs, float** outputs, VstInt32
	sampleFrames);
	virtual void setProgram (VstInt32 program);
	virtual void setProgramName (char* name);
	virtual void getProgramName (char* name);
	virtual bool getProgramNameIndexed (VstInt32 category, VstInt32 index,
	char* text);
	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char* label);
	virtual void getParameterDisplay (VstInt32 index, char* text);
	virtual void getParameterName (VstInt32 index, char* text);
	virtual void resume ();
	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }
	virtual VstPlugCategory getPlugCategory () { return kPlugCategEffect;
}
```

Per a poder crear un efecte, cal derivar-lo de la classe AudioEffectX. En el seu constructor caldrà definir el nombre d'entrades i eixides i inicialitzar una sèrie de paràmetres.

```
Efecte::Efecte(audioMasterCallback audioMaster)
:AudioEffectX(audioMaster, num_programas, num_parametros) {
	setNumInputs(n);
	setNumOutputs(m);
	setUniqueID(nombre);
	canProcessReplacing();
	canDoubleReplacing();
}
```

Taula 3 . Funcions de configuració en la inicialització d'un VST	
virtual void setNumInputs(VstInt32 inputs);	Estableix el nombre d'entrades que el *plugin* manejarà. Aquest valor es fixa en el termini de construcció i no es pot canviar fins que el *plugin* es destruïsca.
virtual void setNumOutputs (VstInt32 outputs);	Estableix el nombre d'eixides que el *plugin* manejarà. Aquest valor es fixa en el termini de construcció i no es pot canviar fins que el *plugin* es destruïsca.
void AudioEffect::setUniqueID (VstInt32 aneu);	Aquesta funció estableix un número d'identificació exclusiu. L'hoste utilitza açò per a identificar cada plugin.
virtual void canProcessReplacing (bool state = true);	Indica que *processReplacing* pot utilitzar-se. En VST 2.4 és obligatori.
virtual void canDoubleReplacing (bool state = true);	Indica que *processDoubleReplacing* està implementat.
virtual float getSampleRate ();	Torna l'actual *Sample Rate*.

En el cas del destructor, aquest és trivial:

```
Efecte:~Efecte() {
}
```

3.2.3 El procés numèric

Les funcions encarregades del procés numèric són *processReplacing* i *processDoubleReplacing*. Aquestes funcions han de realitzar el mateix algoritme, però manejant tipus numèrics distints, amb *float* i *double*. El programa hoste serà l'encarregat de triar una de les dues funcions segons el tipus de dades que estiga manejant.

El tipus de dades *float* està compost per 32 bits, i té un rang de representació [1.17549e-38,3.40288e+38], suportant 6 dígits de precisió.

L'altre tipus de dades que es pot emprar és el tipus de dades double, el qual està compost per 67 bits, i té un rang de representació [2.22507e-308,1.79769e+308], suportant 15 dígits de precisió.

Durant el procés numèric, s'han de tractar un nombre *sampleFrames* de mostres per cada una de les entrades, ja que el mètode té una matriu de punters a les dades d'entrada (*inputs*) als quals se'ls aplica l'algoritme de processament del senyal, quan, una vegada finalitzat s'escriu el resultat en un buffer d'eixida (*outpups*) i reprèn l'operació. Principalment, aquest procés serà dut a terme per un bucle *while (--sampleFrames >= 0)* iterant en cada *frame* de mostra.

El rang de representació de les entrades i eixides és [-1.0…+1.0].

3.2.4 Els valors en l'interfície gráfica

Als paràmetres d'entrada cal associar-los un índex, per a així poder utilitzar-los en moltes de les funcions que explicarem a continuació.

En la interfície gràfica, cada paràmetre haurà de tenir el seu nom, i per a això s'utilitza la funció *getParameterName*. Com els paràmetres d'entrada estan numerats, una possible utilització seria la següent:

```
void Efecte::getParameterName (VstInt32 index, char* label) {
	switch (index) {
		case KParametre1 : vst_strncpy (label, “Nom parametre 1”,
kVstMaxParamStrLen);	break;
		case KParametre2 : vst_strncpy (label, “Nom parametre 2”,
kVstMaxParamStrLen);	break;
	}
}
```

Perquè puga aparèixer el valor de cada paràmetre en la pantalla i així poder-li fer les conversions necessàries, s'utilitza la funció *getParameterDisplay* d'aquesta manera:

```
void Efecte::getParameterDisplay (VstInt32 index, char* text) {
	switch (index) {
		case KParametre1 : dB2string(valor1,text, kVstMaxParamStrLen);
	break;
		case KParametre2 : int2string(valor2,text, kVstMaxParamStrLen);
	break;
	}
}
```

Si ens n'adonem s'utilitzen unes funcions de conversió entre unitats. Aquestes funcions s'anomenen *helpers*, i serveixen per a la gestió d'entrada/eixida dels paràmetres. Els *helpers* són els següents:

```
virtual void dB2string (float value, char* text, VstInt32
maxLen);
virtual void Hz2string (float samples, char* text, VstInt32
maxLen)
virtual void ms2string (float samples, char* text, VstInt32
maxLen);
virtual void float2string (float value, char* text, VstInt32
maxLen);.
virtual void int2string (VstInt32 value, char* text, VstInt32
maxLen);
inline char* vst_strncat (char* dst, const char* src, size_t
maxLen);
```

Una altra funció emprada és la utilitzada per a indicar les unitats de cada paràmetre, per a això s'utilitza la funció *getParameterLabel*, i es pot utilitzar de la manera següent:

```
void Efecte::getParameterLabel (VstInt32 index, char* text) {
```

```
	switch (index) {
		case KParametre1 : vst_strncpy (label, “Unitat1”,
kVstMaxParamStrLen);	break;
		case KParametre2 : vst_strncpy (label, “Unitat2”,
kVstMaxParamStrLen);	break;
	}
}
```

3.2.5 Obtenció dels paràmetres

Des del nostre punt de vista de programador, els valors de tots els paràmetres són un nombre real definit en l'interval [0.0…1.0]. En el programa hoste ens ix una barra per a modificar el valor d'un paràmetre. Internament, quan el selector de la barra es troba a l'esquerra del tot, val 0.0, i, quan es troba a la dreta del tot, val 1.0. Per a poder treballar, hem de subministrar un codi que transforme el valor de la posició en la barra en valors que nosaltres puguem treballar. Per a això, tenim la funció *setParameter*, que és on apareix el codi per a fer l'esmentada conversió. Aquesta funció és invocada per l'hoste cada vegada que l'usuari moga la barra de control. Un possible ús d'aquesta funció seria:

```
void Efecte::setParameter (VstInt32 index, float value) {
	switch (index) {
		case kParametre1 : ValorPrivat1 = value * 100;
			break;
		case kParametre2 : ValorPrivat2 = VMin *
			exp(log(VMax/VMin)*value);
			break;
	}
}
```

Igualment, és necessari codificar el mètode invers, el qual ha de ser capaç de poder convertir un valor d'una variable en una posició en la barra de control. Per a això fem ús de la funció *getParameter*. Un possible ús seria el següent:

```
float Efecte::getParameter (VstInt32 index) {
	float value;
	switch (index) {
		case kParametre1 : value = ValorPrivat1 / 100;
			break;
		case kParametre2 : value = (ValorPrivat2 - VMin) / (VMax -
			VMin);
			break;
	}
}
```

3.3 Entorn de programació

Nosaltres hem usat el sdk amb l'entorn de programació Microsoft Visual Studio 6.0. El sistema de desenvolupament Microsoft Visual Studio és un conjunt d'eines de desenvolupament dissenyades per a ajudar als desenvolupadors de *software* (tant si són principiants com si són professionals amb experiència) a enfrontar-se als desafiaments complexos i crear solucions innovadores. La funció de Visual Studio és millorar el procés de desenvolupament i facilitar el treball necessari per a aconseguir grans avanços i fer-ho amb major satisfacció. Suporta diversos llenguatges de programació com ara Visual C++, Visual C#, Visual J#, ASP.NET i Visual Basic .NET, encara que actualment s'han desenvolupat les extensions necessàries per a molts altres.

3.3.1 Historia

Microsoft va presentar la primera versió de Visual Studio en 1997, incloent per primera vegada en el mateix paquet moltes de les seues eines de programació. Visual Studio 5.0 va ser llançat al mercat en dues edicions: Professional i Enterprise. Incloïa Visual Basic 5.0 i Visual C++ 5.0, per a programació en Windows principalment; Visual J++ 1.1 per a programació a Java i Windows; i Visual FoxPro 5.0 per a programació en xBase. Va introduir Visual Interdev per a la creació dinàmica de llocs web mitjançant ASP (Active Server Pages). S'incloïa una rèplica de la llibreria Microsoft Developer Network a manera de documentació.

La següent versió, la 6.0, es va llançar en 1998 i va ser l'última versió a executar-se a la plataforma Win9x. Els números de versió de totes les parts constituents van passar a 6.0, incloent Visual J++ i Visual InterDev que es trobaven en les versions 1.1 i 1.0 respectivament. Aquesta versió va ser la base per al sistema de desenvolupament de Microsoft per als següents 4 anys, en els que Microsoft va migrar la seua estratègia de desenvolupament al Framework .NET.

Visual Studio 6.0 va ser l'última versió en què Visual Basic s'incloïa de la forma en què es coneixia fins llavors; versions posteriors incorporarien una versió molt diferent del llenguatge amb moltes millores, fruit de la plataforma .NET. També va suposar l'última versió a incloure Visual J++, que proporcionava extensions de la plataforma Java, la qual cosa ho feia incompatible amb la versió de Sun Microsystems. Açò va ocasionar problemes legals a Microsoft, i es va arribar a un acord en què Microsoft deixava de comercialitzar eines de programació que utilitzaren la màquina virtual de Java.

Encara que l'objectiu a llarg termini de Microsoft era unificar totes les ferramentes en un únic entorn, aquesta versió en realitat afegia un entorn més a Visual Studio 5.0: Visual J++ i Visual Interdev se separaven de l'entorn de Visual C++, alhora que Visual FoxPro i Visual Basic continuaven mantenint el seu entorn específic.

Visual Studio 5.0 va suposar el primer intent de Microsoft perquè diversos llenguatges utilitzaren el mateix entorn de desenvolupament. Visual C++, Visual J++, Interdev i MSDN Library feien ús d'un únic entorn, denominat Developer Studio. D'altra banda, Visual Basic i Visual FoxPro usaven diferents entorns.

3.3.2 Característiques

Visual Studio permet als desenvolupadors crear aplicacions, llocs i aplicacions web, així com serveis web en qualsevol entorn que suporte la plataforma .NET. Així es poden crear aplicacions que s'intercomuniquen entre estacions de treball, pàgines web i dispositius mòbils.

Les eines Visual Studio ofereixen als desenvolupadors de *software* millors maneres d'aconseguir més malgastant menys esforç en repeticions i treballs pesats gràcies als editors de codi eficaç, *IntelliSense*, assistents i diversos llenguatges de codificació en un mateix entorn de desenvolupament integrat (IDE) fins a productes avançats d'administració del cicle de vida de les aplicacions. Les noves versions de Visual Studio continuen aportant eines innovadores perquè els desenvolupadors puguen centrar-se en la solució de problemes i no perden el temps en nimietats.

Es tracta d'un producte integrat que comprèn eines, servidors i serveis. Els productes de Visual Studio funcionen bé conjuntament, no sols entre si, sinó també amb un altre software de Microsoft, com els productes de servidor de Microsoft i el sistema Microsoft Office.

Visual Studio ofereix una varietat d'eines per a totes les fases del desenvolupament de *software* (desenvolupament, proves, implementació, integració i administració) tant per al principiant com per al professional amb experiència. Visual Studio també s'ha dissenyat per a admetre el desenvolupament en qualsevol classe de dispositius: equips, servidors, la web i dispositius mòbils.

3.3.3 Visual C++

Visual C++ (també conegut com MSVC, Microsoft Visual C++) és un entorn de desenvolupament integrat (IDE) per a llenguatges de programació Microsoft C, C++ i C++/CLI. Aquest està dissenyat especialment per al desenvolupament i depuració de codi escrit per a les API's de Microsoft Windows, DirectX i la tecnologia Microsoft .NET Framework.

Visual C++ fa ús extensiu del framework Microsoft Foundation Classes (o simplement MFC), el qual és un conjunt de classes C++ per al desenvolupament d'aplicacions gràfiques en Windows. Disposa d'una versió Express, anomenada Microsoft Visual C++ Express Edition, que és gratuïta i es pot descarregar des de la web de Microsoft.

El llenguatge de programació utilitzat per aquesta eina, del mateix nom, està basat en C++, i és compatible en la major part del seu codi amb aquest llenguatge, al mateix temps que la seua sintaxi és exactament igual. En algunes ocasions aquesta incompatibilitat impedeix que altres compiladors, sobretot en altres sistemes operatius, funcionen bé amb codi desenvolupat en aquest llenguatge.

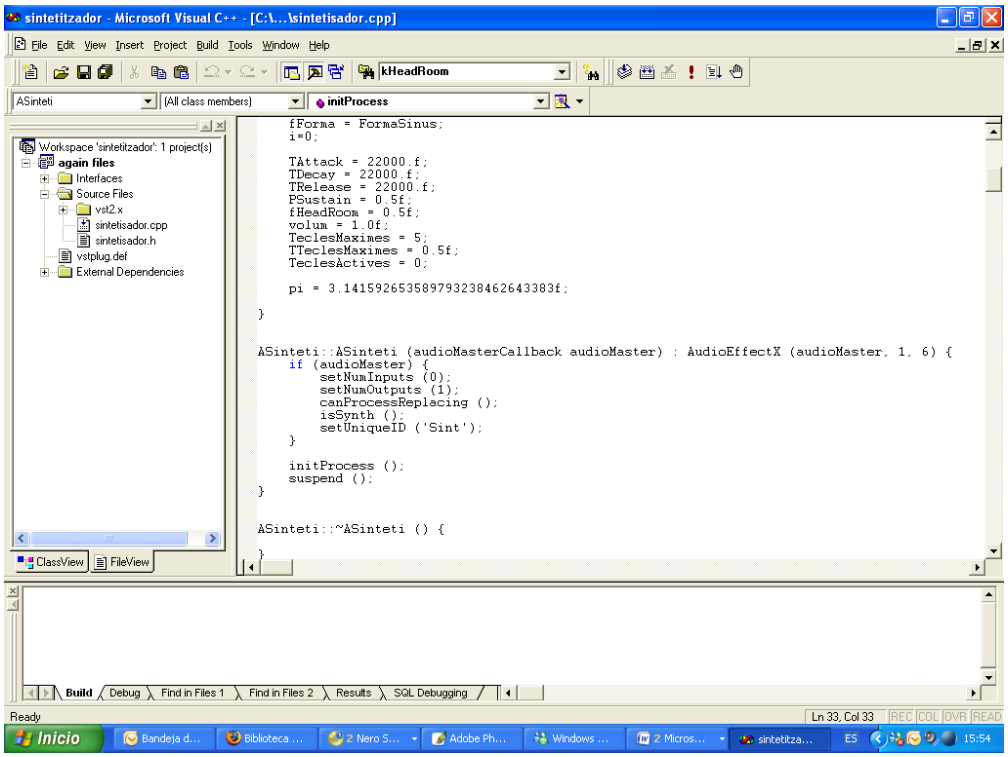

figura 29. Entorn de treball de Visual C++

4 Col·lecció de plugins

4.1 Balanç

4.1.1 Descripció

El balanç és el procés de posicionament d'un senyal d'àudio en el panorama de so. Per exemple, si es tracta d'una mostra d'àudio estèreo, l'usuari pot dir aïllar un canal, o canviar la mescla perquè se senta més un canal que un altre. Aquest efecte s'aconseguix movent una barra en la posició desitjada.

El funcionament del balanç depén de si la mostra a aplicar l'efecte de Balanç és mono o estèreo. L'eixida del balanç és sempre estèreo. Si la font de senyal és mono, la posició del balanç determina quina quantitat del senyal mono es dirigix a l'esquerra o a la dreta. Si la barra de balanç es troba al màxim, ja siga esquerra o dreta, el 100% del senyal bonic és redirigit al canal d'eixida apropiat, i en el canal oposat el nivell es reduïx a zero. Si la barra de balanç es col·loca en el centre, la meitat del senyal mono eixirà pel canal dret i l'altra meitat pel canal esquerre.

Per al nostre cas, si la font de senyal per al balanceig és estèreo, el balanç funciona d'una forma un poc diferent. La separació per canals existix ja que es manté que el canal d'entrada esquerre sempre es dirigix al canal d'eixida esquerre i el canal d'entrada dret sempre es dirigix al canal d'eixida dret. Per això, l'usuari, en la barra selectora del balanç, realment selecciona l'atenuació del canal contrari, ja que si la barra selectora es troba en el 50% del canal dret, estem atenuant el canal esquerre en un 50%, i, si la posició és del 100% de la dreta, l'atenuació del canal esquerre és del 100%, i per tant, només s'escoltaria el canal dret.

4.1.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- Balanç: Ens permet determinar el balanceig del senyal entre el canal esquerre i el canal dret. En la pràctica, escoltar el so per l'altaveu esquerre o per l' altaveu dret.

4.1.3 Operació

El procés per a la execució es molt senzill, ja que sols s'ha de calcular l'atenuació del canal corresponent tenint en compte la posició en la barra de selecció del balanç.

```
void ABalance::setParameter (VstInt32 index, float value) {
	fBalance = (value*200) -100;
}
```

El resultat d'aquest càlcul es troba en [-100,100]. Aleshores, si el valor resultant està entre [-100,0[l'ateuació s'aplicarà al canal dret, mentre que si el valor està entre]0,100] l'atenuació s'aplicarà al canal esquerre. Aquest procés s'aplica al *processReplacing* com es pot apreciar a continuació:

```
if(fBalance==0.0) {
                    (*out1++) = (*in1++);
                    (*out2++) = (*in2++);
             }
```

```
if(fBalance>0.0) {  //Dreta
	float aux=fBalance/100;
	(*out1++) = (*in1++)*(1-aux);
	(*out2++) = (*in2++);
}

if(fBalance<0.0) {  //Esquerra
	float aux=fBalance/100;
	aux= -aux;
	(*out1++) = (*in1++);
	(*out2++) = (*in2++)*(1-aux);
}
```

El resultat final de la interfície gràfica depén de l'hoste amb què s'execute el *plugin*. Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la Figura 30.

figura 30. Balance executat amb Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/balance.mp3.

4.2 Distorsió

4.2.1 Descripció

La distorsió es pot definir com la “deformació” que patix un senyal després del seu pas per un sistema. La deformació que nosaltres aplicarem és la saturació del senyal d'àudio i aconseguirem com a resultat la producció d’una distorsió com la de les primeres generacions de guitarres amb pedals. Per a fer això, hem de tallar el senyal de la guitarra.

El nostre resultat obtingut s'aconseguix amb un enfocament un poc dur, ja que les noves generacions de distorsions tenen un so molt més ric.

4.2.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- Distorsió: Amb este paràmetre controlem el valor llindar a partir del qual produïx la saturació. En el "més", és a dir, en el final de l'escala de la distorsió, el llindar es fixa en 0.0725 (ja que el valor de la mostra està comprés entre [0,1]), i per tant, les mostres que presenten un valor igual o superior s'establiran amb aquest valor. En el "menys", és a dir, al principi de l'escala de distorsió, el llindar es fixa en 1.0.

- Guany: Atés que les mostres una vegada saturades perden amplitud, podem utilitzar el guany per a compensar aquesta pèrdua.

4.2.3 Operació

Retallar les mostres digitalment és molt fàcil. Només hem de comparar el valor de la mostra amb el valor límit seleccionat i substituir la mostra amb el valor llindar, si aquest és superat. El codi per a poder realitzar açò es mostra a continuació.

```
if (inleft > distorsio) {
	inleft=distorsio;
}
else if (inleft < -distorsio) {
	inleft = -distorsio;
}

if (inright > distorsio) {
inright=distorsio;
}
else if (inright < -distorsio ) {
	inright = -distorsio;
}
```

A l'escoltar el resultat, notem que hi ha una mostra d'amplitud que es perd a conseqüència de la saturació que podem compensar amb l'augment del guany. Aquest ajust es realitza amb una simple multiplicació com es pot observar a cotinuació.

```
*out1++ = inleft*fGain;
*out2++=  inright*fGain;
```

El resultat final de la interfície gràfica depén de l'hoste amb què s'execute el *plugin*. Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la Figura 31.

figura 31. Distorsió executat a Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/distorsio.mp3.

4.3 Delay

4.3.1 Descripció

El *delay* (retard en valencià) és l'efecte que consistix en la multiplicació i retard modulat d'un senyal sonor pel qual una vegada processat el senyal és mesclat amb l'original. L'ús del *delay* és comú en la indústria discogràfica i també molt utilitzat per guitarristes elèctrics.

Encara que aquest efecte s'anomene *delay*, podem fer amb ell dos efectes de so molt semblants: per un costat el *delay* pròpiament dit, i per un altre costat l'*Echo* (eco).

Tots dos es diferencien únicament en el temps en què s'escolten els retards dels sons, ja que el *delay* té la funcionalitat de donar cos al so, mentre que el eco té la funcionalitat d'enriquir el ritme. Per això els retards de curta duració (< 200 mil·lisegons) podem denominar-los *delay*, mentre els retards per damunt dels 200 mil·lisegons els considerem eco.

4.3.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- *Delay*: Ens permet determinar el temps de retard des d'1 mil·lisegons fins a 200 mil·lisegons en escala logarítmica.

- *Dry/Wet*: Es tracta d'un control per a mesclar el senyal original i el senyal de l'efecte de so. En la pràctica, té la seua funció per a ubicar el so dins de la mescla musical.

4.3.3 Operació

El diagrama de blocs es pot observar en la Figura 32. Com es pot apreciar, el senyal pot recórrer dos camins. L'efecte del retard l'anem a implementar amb un simple *buffer* circular amb una longitud fixada per un retard màxim de 200 mil·lisegons i controlat pel retard sol·licitat per l'usuari. El codi utilitzat per a calcular la posició on hem d'agafar el senyal retardat és el següent:

```
//***** Calcul de la posició on es trobarà el frame del retard *****//
      int posicionretardo = posicion-delay;

      if (posicionretardo<0) {
            posicionretardo= L+posicionretardo;
      }
```

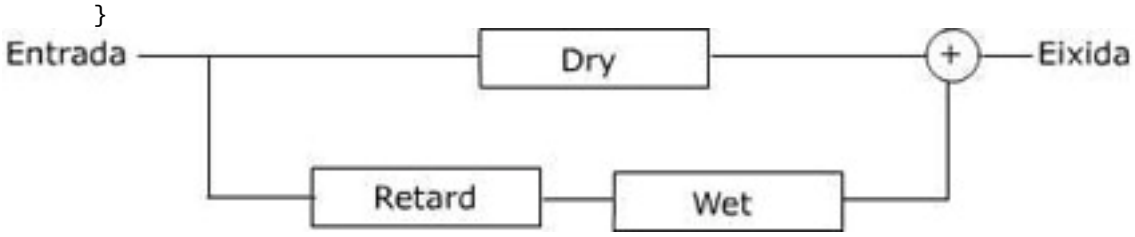

figura 32. Diagrama de blocs del efecte Delay

El *buffer* circular es pot observar en la Figura 33.

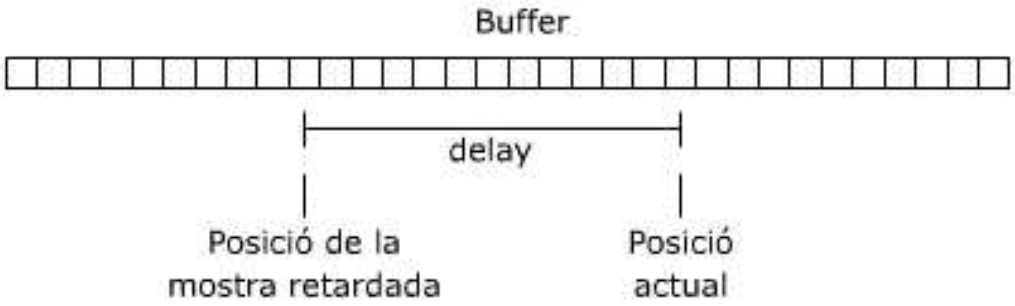

figura 33. Buffer circular

Llavors, l'operació d'eixida, una vegada calculada quina és la posició del senyal d'eixida, és molt fàcil, només basta de sumar aquest dos senyals amb el percentatge del paràmetre *wet/dry*, ací es pot observar amb el codi.

```
*out1++ = float(inleft*(1-(fDryWet/100))+
      buffer1[posicionretardo] * (fDryWet/100.0));
*out2++=  float(inright*(1-(fDryWet/100))+ buffer2[posicionretardo] *
     (fDryWet/100.0));
```

El resultat final de la interfície gràfica depén de l'hoste amb què s'execute el *plugin*. Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la Figura 34.

figura 34. Efecte delay executat a Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/delay.mp3.

4.4 Eco

4.4.1 Descripció

L'eco l'anem a tractar com un efecte idèntic al *delay* explicat anteriorment. L'única diferència es troba que el rang de retard es troba entre 200 mil·lisegons i 4 segons. Com ja ho havíem explicat anteriorment, el *delay* té la funcionalitat de donar cos al so, mentres que l'eco té la funcionalitat d'enriquir el ritme.

Perquè no siga tan idèntic al *delay*, inserirem el paràmetre de retroalimentació (*Feedback*), ja que amb aquest paràmetre podrem controlar el nombre de repeticions que li donarem a l'eco. Com més alt siga aquest valor, més llarg serà l'efecte d'eco. Si el posem al màxim, les repeticions seran infinites.

4.4.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- *Delay*: Ens permet determinar el temps de retard des de 200 mil·lisegons fins a 4 segons en escala logarítmica.

- *Dry/Wet*: Es tracta d'un control per a mesclar el senyal original i el senyal de l'efecte de so. En la pràctica té la seua funció per a ubicar el so dins de la mescla musical.

-*Feedback*: Ens permet determinar el número de repeticions que se li aplica al eco en un percentatge de repeticions.

4.4.3 Operació

El diagrama de blocs es pot observar en la figura 35. Com es pot apreciar, es pràcticament igual que el diagrama de blocs de l'efecte delay, només que aquest inclou el *Feedback*. Igualment, el senyal pot recórrer dos camins. L'efecte del retard l'anem a implementar amb un simple *buffer* circular amb una longitud fixada per un retard màxim de 4 segons i controlat pel retard sol·licitat per l'usuari.

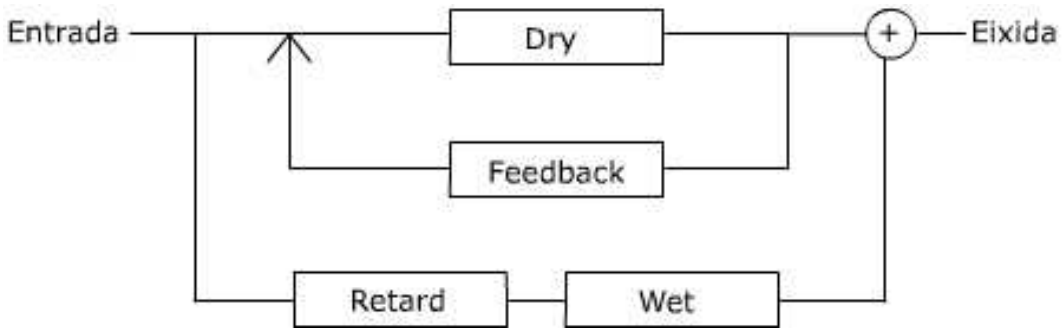

figura 35. Diagrama de blocs del efecte Eco

L'operació d'eixida, una vegada calculat quina és la posició del senyal d'eixida es idèntic al de l'efecte *delay*, en el qual només basta de sumar aquests dos senyals amb el percentatge del paràmetre *wet/dry*. Ací es pot observar amb el codi.

```
*out1++ = float(inleft*(1-(fDryWet/100))+ buffer1[posicionretardo] *
(fDryWet/100.0));
```

```
*out2++=  float(inright*(1-(fDryWet/100))+ buffer2[posicionretardo] *
(fDryWet/100.0));
```

L'única diferència que existix el teniu a continuació, com podeu observar aquest codi:

```
//***** Guardem el in actual en el buffer actual *****//
buffer1[posicion]=(1-(fFeed/100))*inleft+(fFeed/100)*outleft;
buffer2[posicion]=(1-(fFeed/100))*inright+(fFeed/100)*outright;
```

A l'hora de guardar l'entrada en el *buffer*, aquesta no és directa com es feia abans, sinó que es guarda la suma utilitzant el percentatge del paràmetre *feedback* de l'entrada sense modificar amb l'eixida en què ha sigut aplicat l'efecte de l'eco.

Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la figura 36.

figura 36. Efecte Eco executat a Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/eco.mp3.

4.5 Chorus

4.5.1 Descripció

La definició de l'efecte *Chorus* és el resultat de mesclar un senyal sense processar amb una senyal amb *vibrato*. En el nostre cas, el resultat serà de mesclar un senyal retardat i modulat per una baixa freqüència amb el senyal sense processar. Aquest efecte té per nom *Chorus* ja que s'utilitza per a fer una sola veu igual que molts de so.

Els temps de retards utilitzats per al *Chorus* són alts, i per tant l'efecte del eco és evident. El diagrama de blocs es pot veure en la figura 1.

El resultat sonor d'aquest efecte és el *syrupy*, que apareix en moltes de les gravacions dels anys 60 i 70.

4.5.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- *Dry/Wet*: Es tracta d'un control per a mesclar el senyal original i la senyal de l'efecte de so. En la pràctica té la seua funció per a ubicar el so dins de la mescla musical.

- Guany: El guany, el podem utilitzar per a compensar alguna pèrdua en la modulació de la senyal.

- *Delay* Freqüencia: Indica la freqüència d'escombrat de l'efecte.

- *Delay Min*: Establix la profunditat mínima de l'efecte en forma de retard.

- *Delay Max*: Establix la profunditat mínima de l'efecte en forma de retard.

4.5.3 Operació

Aquest efecte és semblant als dos anteriors, ja que cal modular un senyal alentit. Per a això anem a utilitzar la interpolació de Lagrange, ja que la finalitat és que a partir d'una sèrie de punts, puguem obtenir una equació de manera que la seua corba passe per tots ells o tan prop com puga ser. El primer que cal fer és calcular els coeficients.

```
            nT = 1/44100 * (n++);
            deltai = (delayMax-delayMin)/2 + delayMin + (delayMax-delayMin)/2
* sin(2 * pi * fFreq * nT);
            // Delay line emprant Lagrange interpolator
            di = floorf(deltai) - 1;
            t0 = deltai - di;
            t1 = t0 - 1;
            t2 = t0 - 2;
            t3 = t0 - 3;
            b0 = -t1*t2*t3/6;
            b1 =  t0*t2*t3/2;
            b2 = -t0*t1*t3/2;
            b3 =  t0*t1*t2/6;
```

Seguidament haurem d'emmagatzemar les 4 posicions alentides que anem a utilitzar:

```
            for (int i=0;i<4;i++) {
                  outPoint[i] = (posicion - 1 - di + i);
                  if (outPoint[i] < 0)
                        outPoint[i] += L;
                  else if (outPoint[i] >= L)
```

```
                outPoint[i] -= L;
            listate[i] = buffer1[outPoint[i]];
        }
```

I una vegada tinguem les 4 posicions alentides i els coeficients, només cal multiplicar-los entre ells.

```
feedbackL = (b0*listate[0] + b1*listate[1] + b2*listate[2] + b3*listate[3]);
```

El resultat final de la interfície gràfica depén de l'hoste amb què s'execute el *plugin*. Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la figura 37.

figura 37. Efecte Chorus executat a Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/chorus.mp3.

4.6 Generador

4.6.1 Descripció

La funció d'un generador de senyal és produir un senyal dependent del temps amb unes característiques determinades de freqüència, amplitud i forma. Algunes vegades aquestes característiques són externament controlades a través de senyals de control; l'oscil·lador controlat per tensió (*voltage-controlled oscillator* o VCO) és un clar exemple. Per a executar la funció dels generadors de senyal s'empra algun tipus de realimentació conjuntament amb dispositius que tinguen característiques dependents del temps (normalment condensadors).

Nosaltres construirem un generador amb tres tipus de formes: sinusoïdal, quadrada i triangular.

4.6.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- Freqüència: On es pot triar una freqüència per al generador entre 20 i 20.000 Hz.

- Amplitud: Es tria en percentatge l'amplitud de l'ona del senyal.

- Forma del senyal: Se selecciona la forma que tindrà el senyal. Es pot triar entre sinusoïdal, quadrada o triangular.

4.6.3 Operació

La primera cosa que cal realitzar és el càlcul de la fase.

```
fase = fmod(fFrequencia*i,1);
i++;
```

Una vegada calculada la fase, i segons la forma del senyal que s'haja seleccionat, s'haurà de realitzar una operació diferent. Per a la forma sinusoïdal és tan simple com:

```
*out++ = fAmplitud*sin(2*pi*fase);
```

Ja que només fa falta l'amplitud que la selecciona l'usuari i la fase que ja està calculada amb anterioritat.

Per a la forma quadrada, cal dividir-ho en dues parts, si la fase és menor que la meitat o si és major. Les operacions per a la forma quadrada són les següents:

```
if(fase<0.5) {
	*out++ = fAmplitud;
}
else {
	*out++ = -fAmplitud;
}
```

Per a la forma triangular és un poc mes difícil, ja que cal basar-nos en les formules de les rectes. Com es pot apreciar a la figura 38, hi ha dues rectes clarament diferenciables, i per tant cal dividir la fase en dos i aplicar a cada una la formula de la recta y=m*x+b.

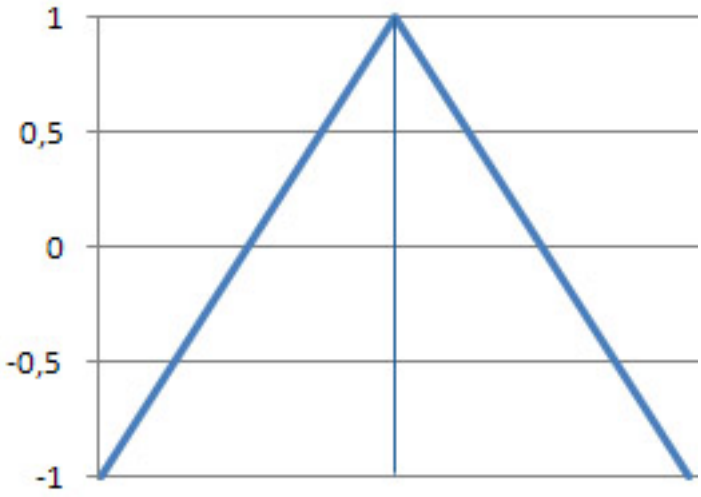

figura 38. Forma triangular

Per a la primera meitat, cal calcular quin seria el pendent, que es calcula m=ΔY / ΔX.

ΔY = 2 * amplitud ΔX = 1/2

Una vegada calculat el pendent, i sabent que x = fase i b = -amplitud, només hauríem d'aplicar tots aquests valors a la fórmula y = m*x+b.

```
float m = 2 * fAmplitud / 0.5f;
*out++ = m * fase - fAmplitud;
```

Per a l'altra meitat, es faria el mateix, es calcularia el pendent i els valors per a la fórmula y = m*x+b, quedaria el codi de la manera següent:

```
float m = -2 * fAmplitud / 0.5f;
*out++ = m * (fase - 0.5f) + fAmplitud;
```

El resultat final de la interfície gràfica depén de l'hoste amb què s'execute el *plugin*. Es pot observar el resultat utilitzant l'hoste Steinberg Cubase en la figura 39.

figura 39. Generador a Cubase

Pots escoltar un exemple d'aquest efecte en /exemples/generador.mp3.

4.7 Sintetitzador

4.7.1 Descripció

Un sintetitzador és un aparell que genera i manipula sons artificials per mitjans electrònics, usant tècniques com la síntesi additiva, substractiva, de modulació de freqüència, de modelatge físic o modulació de fase. Amb aquest artefacte es poden crear sons nous i també reproduir instruments musicals coneguts. La forma de l'ona generada és alterada en la seua durada, altura i timbre mitjançant l'ús de dispositius.

Nosaltres utilitzarem un teclat MIDI de la marca Ego-systems, model 25 KeyControl (figura 40) que té aquestes característiques:

Característiques físiques:

- 25 tecles sensibles semi-contrapesades

-Rodes de Pitch, Mod i Dades

- 8 botons rotatius assignables i 8 faders assignables

MIDI:

- 1) 1 port MIDI USB (16 canals)

- 2) 1 *In* / 1 Out MIDI (16 canals)

figura 40. Controlador Ego-systems 25KeyControl

Utilitzem el teclat com a controlador perquè és el més popular per a tot tipus de sintetitzadors. És una font que subministra tensions proporcionals a la tecla que s'oprimeix. També proporciona tensions de control per a indicar-nos la velocitat de pulsació de la tecla o la pressió sobre la mateixa.

Nosaltres construirem un sintetitzador polifònic, amb la component d'una envoltant ADSR (figura 41). Aquesta envoltant és capaç de poder adaptar el timbre al sintetitzador, per a així que sone més com un instrument mecànic. Un ràpid atac ajuda perquè sone més com un orgue, un llarg decaïment fa que s'assemble més a una guitarra.

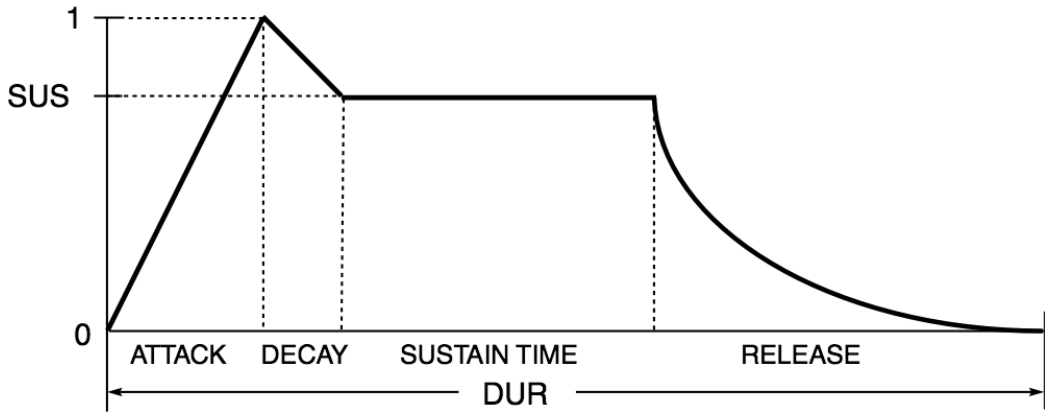

figura 41. Envoltant ADSR

El controlador disposa de botons rotatius assignables, els quals anem a fer-los valdre per a canviar els paràmetres com el volum o com els relacionats amb l'envoltant.

4.7.2 Paràmetres

Els paràmetres per a utilitzar aquest efecte són:

- Forma del senyal: Se selecciona la forma que tindrà el senyal. Es pot triar entre sinusoïdal, quadrada o triangular.

-Temps d'atac (*Attack*): Selecciona el temps d'atac de l'envoltant.

-Temps de decadència (*Decay*): Selecciona el temps de decadència de l'envoltant.

-Temps de llibertat (*Release*): Selecciona el temps llibertat de l'envoltant.

-Percentatge en la nota mantinguda (Sustain): Selecciona el percentatge sobre la velocitat pressionada de la tecla.

- Tecles màximes: Se selecciona el nombre màxim de polifonia.

4.7.3 Operació

Per a fer el sintetitzador, hem d'utilitzar un generador de senyals, per a la qual cosa utilitzarem el codi del generador realitzat anteriorment.

Primer de tot, haurem de crear una estructura de nota en la qual emmagatzemarem els paràmetres següents:

```
struct nota_struct {
	bool noteIsOn;
	VstInt32 currentNote;
	VstInt32 Velocity;
	VstInt32 Delta;
	float fPhase1, fPhase2;
```

```
	int estat;
	int CAttack, CDecay, CRelease;
	float NoteFreq;
	float salida;

};
```

Una nota pot estar en diversos estats: *Off*, Attack, Decay, *Sustain* o Release. Aquests estats són els que conté l'envoltant, i per tant hem de tenir present en quin temps de cada estat està utilitzant les variables CAttack, CDecay i CRelease.

Una vegada creada aquesta estructura, caldrà fer un vector amb aquesta, de 85 posicions que són les que es poden utilitzar amb el teclat MIDI (sis octaves).

```
nota_struct nota[85];
```

Tot seguit, cal crear la funció encarregada de capturar les interrupcions del teclat MIDI. Les interrupcions que s'encarregarà de filtrar seran la d'activació i desactivació d'una tecla i la de totes les tecles desactivades.

```
VstInt32 ASinteti::processEvents (VstEvents* ev) {
	for (VstInt32 i = 0; i < ev->numEvents; i++) {
		if ((ev->events[i])->type != kVstMidiType)
			continue;

		VstMidiEvent* event = (VstMidiEvent*)ev->events[i];
		char* midiData = event->midiData;
		VstInt32 status = midiData[0] & 0xf0;  // Seleccionem els 4
primers bits
		if (status == 0x90 || status == 0x80) {// Comparem que els 4
primers bits siguen activar o desactivar nota
			VstInt32 note = midiData[1] & 0x7f;
			VstInt32 velocity = midiData[2] & 0x7f;
			if (status == 0x80)
				noteOff (note);
			else
				noteOn (note, velocity, event->deltaFrames);
		}
		else if (status == 0xb0) {
			if (midiData[1] == 0x7e || midiData[1] == 0x7b)     //
totes les notes off
				noteOffAll();
		}
		event++;
	}
	return 1;
}
```

Si ens n'adonem en la funció anterior, quan es desactiva una tecla s'anomena la funció *noteOff* passant-li com a paràmetre que nota és la que aquesta deixa de ser polsada, quan s'activa una nota s'anomena la funció *noteOn* passant-li els paràmetres de quina nota és la que s'activa, la velocitat amb què és polsada la tecla i el *deltaFrames*. Aquestes funcions s'encarreguen d'inicialitzar en l'estat d'*Attack (noteOn*), o en l'estat *Release (noteOff*).

```
void ASinteti::noteOn (VstInt32 note, VstInt32 velocity, VstInt32 delta) {
	int pos;
	pos = conversorNotaenPosicion(note);
	nota[pos].Velocity = velocity;
	nota[pos].Delta = delta;
	nota[pos].noteIsOn = true;
```

```
	nota[pos].fPhase1 = nota[pos].fPhase2 = 0;
	nota[pos].CAttack = nota[pos].CDecay = nota[pos].CRelease = 0;
	nota[pos].salida = 0.0f;
	nota[pos].estat = EAttack;
}

void ASinteti::noteOff (VstInt32 note) {
	int pos;
	pos = pos = conversorNotaenPosicion(note);
	nota[pos].estat = ERelease;
}
```

Com la denominació de la nota al teclat ve en hexadecimal des de la nota 24 a la 96, hem de crear una funció que ens convertisca la nota en hexadecimal en la posició corresponent en el vector.

```
int ASinteti::conversorNotaenPosicion(VstInt32 Nota) {
	if (Nota == 0x18) return 0; // C octava1
	else if (Nota == 0x19) return 1;
	else if (Nota == 0x1A) return 2;
	else if (Nota == 0x1B) return 3;
	else if (Nota == 0x1C) return 4;
	else if (Nota == 0x1D) return 5;
	else if (Nota == 0x1E) return 6;
	else if (Nota == 0x1F) return 7;
	else if (Nota == 0x20) return 8;
…
```

Per a poder utilitzar els botons rotatius del controlador, hem de saber quin serà el missatge MIDI que rebrem. Com s'ha explicat anteriorment, els missatges de canvi de control són els que el *byte* d'estat és 1011cccc. Una vegada rebut el missatge de canvi de control, hem de saber el canal associat a aquest missatge, ja que cada canal indica un canvi de control diferent.

Hi ha canals predeterminats per a un tipus de control, dels quals utilitzarem:

- Canal 7: Volum.
- Canal 72: *Release time*.
- Canal 73: *Attack time*.
- Canal 75: *Decay time*.

Dels canals lliures utilitzarem el canal 79 per al *Sustain time*.

Per tant, al codi del *processEvent*s hem d'afegir el següent:

```
	if (midiData[1] == 0x49) {
		int midiData2 = midiData[2];
		setParameter(kAttack,midiData2/127.0f);
		getParameter(kAttack);
	}

	if (midiData[1] == 0x4B) {
		int midiData2 = midiData[2];
		setParameter(kDecay,midiData2/127.0f);
		getParameter(kDecay);
	}
	if (midiData[1] == 0x4F) {
		int midiData2 = midiData[2];
		setParameter(kSustain,midiData2/127.0f);
		getParameter(kSustain);
	}
	if (midiData[1] == 0x07) {
```

```
		int midiData2 = midiData[2];
		volum = midiData2/127.0f;
	}
```

Per a la seua utilització, cada persona ha d'associar els canals als botons rotatius. En el nostre cas, per a configurar el controlador *Ego System Keycontrol* 25 hem de realitzar els passos següents:

- Pas 1: Polseu el botó *"edit"* per a entrar en mode edició. El LED del botó "edit" s'il·luminarà.
- Pas 2: Polseu el botó *"Ctrl. Assign"* (etiquetatge amb el número 2).
- Pas 3: Moveu el botó rotatiu que es desitja canviar el canal, i llavors en la pantalla apareixerà el canal que té assignat.
- Pas 4: Introduiu el número de canal que es desitja assignar al botó utilitzant el Pad numèric o el botó + / - i polsar el botó "Intro". El mode torna a l'estat inicial del mode d'edició.

Una vegada creat tot açò, cal realitzar les operacions necessàries al *processReplacing*. En cada frame, hem de calcular les notes que estiguen actives, en què instant del generador es troben, i segons l'estat també aplicar-los la funció de l'envoltant.

```
switch (nota[pos].estat) {
	case EAttack:
		PAttack = float(nota[pos].CAttack) / TAttack;
		nota[pos].salida = nota[pos].salida * PAttack;
		nota[pos].CAttack++;
		if (nota[pos].CAttack >= TAttack) {
			nota[pos].estat = EDecay;
		}
		break;
	case EDecay:

		PDecay = ((PSustain-1.0f) / TDecay) * float(nota[pos].CDecay) +
1.0f;
		nota[pos].salida = nota[pos].salida * PDecay;
		nota[pos].CDecay++;
		if (nota[pos].CDecay >= TDecay) {
			nota[pos].estat = ESustain;
		}
		break;
	case ESustain:
		nota[pos].salida = nota[pos].salida*PSustain;
		break;
	case ERelease:
		PRelease = (-PSustain / TRelease) * float(nota[pos].CRelease) +
PSustain;
		nota[pos].salida = nota[pos].salida*PRelease;
		nota[pos].CRelease++;
		if (nota[pos].CRelease >= TRelease) {
			nota[pos].estat= EOff;
			nota[pos].noteIsOn = false;
		}
		break;
}
```

Una vegada calculada l'eixida per a cada nota activa, cal sumar l'eixida de totes les notes actives:

```
for(pos=0;pos<85;pos++) {
                    if (nota[pos].noteIsOn) {
                           exit = nota[pos].salida + exit;

                    }
             }
             *out++ = exit;
```

5 Annexos

Annex I

balance.h

```
//-------------------------------------
// VST Plug-Ins SDK
// BALANCE
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "balance.h"


#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"


enum {
	NumeroProgramBalance,
	kNumParams
};

class ABalance : public AudioEffectX {
public:
	ABalance (audioMasterCallback audioMaster);
	~ABalance ();

	// Proces
	virtual void processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames);
	virtual void processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames);

	// Parametres
	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char* label);
	virtual void getParameterDisplay (VstInt32 index, char* text);
	virtual void getParameterName (VstInt32 index, char* text);

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion ();

protected:
	float fBalance;
	char programName[kVstMaxProgNameLen+1];
};

#endif
```

Annex II

balance.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// BALANCE
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "balance.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster)
{
	return new ABalance (audioMaster);
}


ABalance::ABalance (audioMasterCallback audioMaster) : AudioEffectX
(audioMaster, 1, 1) {
	setNumInputs (2);		// stereo in
	setNumOutputs (2);		// stereo out
	setUniqueID ('Bal');		// Identificador
	canProcessReplacing ();
	canDoubleReplacing ();

	fBalance = 0;
}


ABalance::~ABalance () {

}


void ABalance::setParameter (VstInt32 index, float value) {
	fBalance = (value*200) -100;

}


float ABalance::getParameter (VstInt32 index) {
	float v;
	v=(fBalance+100.f)/200.f;
	return v;
}


void ABalance::getParameterName (VstInt32 index, char* label) {
	switch (index) {
		case NumeroProgramBalance : vst_strncpy (label, "Balance",
kVstMaxParamStrLen);
			break;
	}
}


void ABalance::getParameterDisplay (VstInt32 index, char* text) {
	float2string (fBalance, text, kVstMaxParamStrLen);
}
```

```
void ABalance::getParameterLabel (VstInt32 index, char* label) {
	vst_strncpy (label, "%", kVstMaxParamStrLen);
}


bool ABalance::getEffectName (char* name) {
	vst_strncpy (name, "Balance", kVstMaxEffectNameLen);
	return true;
}


bool ABalance::getProductString (char* text) {
	vst_strncpy (text, "Balance", kVstMaxProductStrLen);
	return true;
}


bool ABalance::getVendorString (char* text) {
	vst_strncpy (text, "Francisco Javier Ripoll", kVstMaxVendorStrLen);
	return true;
}


VstInt32 ABalance::getVendorVersion () {
	return 1000;
}


void ABalance::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {
    float* in1  =  inputs[0];
    float* in2  =  inputs[1];
    float* out1 = outputs[0];
    float* out2 = outputs[1];

    while (--sampleFrames >= 0)
    {
             if(fBalance==0.0) {
                    (*out1++) = (*in1++);
                    (*out2++) = (*in2++);
             }

             if(fBalance>0.0) {  //Dreta
                    float aux=fBalance/100;
                    (*out1++) = (*in1++)*(1-aux);
                    (*out2++) = (*in2++);
             }

             if(fBalance<0.0) {  //Esquerra
                    float aux=fBalance/100;
                    aux= -aux;
                    (*out1++) = (*in1++);
                    (*out2++) = (*in2++)*(1-aux);
             }


    }
}


void ABalance::processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames)
{
    double* in1  =  inputs[0];
    double* in2  =  inputs[1];
    double* out1 = outputs[0];
    double* out2 = outputs[1];
```

```
    while (--sampleFrames >= 0)
    {
              if(fBalance==0.0) {
                     (*out1++) = (*in1++);
                     (*out2++) = (*in2++);
              }

              if(fBalance>0.0) {  //Dreta
                     double aux= fBalance/100.0;
                     (*out1++) = (*in1++)*(1-aux);
                     (*out2++) = (*in2++);
              }

              if(fBalance>0.0) {  //Esquerra
                     double aux=fBalance/100.0;
                     aux= -aux;
                     (*out1++) = (*in1++);
                     (*out2++) = (*in2++)*(1-aux);
              }


    }
}
```

Annex III

distorsio.h

```
//-------------------------------------
// VST Plug-Ins SDK
// DISTORSIÓ
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#ifndef __ADistorsio__
#define __ADistorsio__

#include "public.sdk/source/vst2.x/audioeffectx.h"


enum {
	kDistorsio = 0,
	kGain,
	kNumParams
};


class ADistorsio : public AudioEffectX {
public:
	ADistorsio (audioMasterCallback audioMaster);
	~ADistorsio ();

	virtual void processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames);
	virtual void processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames);

	virtual void setProgramName (char* name);
	virtual void getProgramName (char* name);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char* label);
	virtual void getParameterDisplay (VstInt32 index, char* text);
	virtual void getParameterName (VstInt32 index, char* text);

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion ();

protected:
	float fGain;
	float iDistorsio;
	float fDistorsio;
	char programName[kVstMaxProgNameLen+1];
	float MaxBits;
	float MinBits;
};

#endif
```

Annex IV

distorsio.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// DISTORSIÓ
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "distorsio.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new ADistorsio (audioMaster);
}


ADistorsio::ADistorsio (audioMasterCallback audioMaster): AudioEffectX
(audioMaster, 1, 2) {

	setNumInputs (2);		// stereo in
	setNumOutputs (2);		// stereo out
	setUniqueID ('Dist');		// identify
	canProcessReplacing ();
	canDoubleReplacing ();

	MaxBits=1.f;  //32767
	MinBits=0.0725f;   //2376

	fGain = 1.f;			// default to 0 dB

	fDistorsio=0.f;
	iDistorsio=MaxBits;
	vst_strncpy (programName, "Default", kVstMaxProgNameLen);  // default
program name
}


ADistorsio::~ADistorsio () {

}


void ADistorsio::setProgramName (char* name) {
	vst_strncpy (programName, name, kVstMaxProgNameLen);
}

void ADistorsio::getProgramName (char* name) {
	vst_strncpy (name, programName, kVstMaxProgNameLen);
}


void ADistorsio::setParameter (VstInt32 index, float value) {
	switch (index)	{
		case kDistorsio :   iDistorsio= MaxBits-value*(MaxBits-MinBits);
                                        fDistorsio=value*100;
			break;
		case kGain : fGain = value*2;
			break;
	}
}

float ADistorsio::getParameter (VstInt32 index) {
```

```
	float v = 0;

	switch (index)	{
		case kDistorsio : v = fDistorsio/100;  break;
		case kGain	: v = fGain/2;		break;
	}
	return v;
}

void ADistorsio::getParameterName (VstInt32 index, char* label) {
		switch (index) {
			case kDistorsio : strcpy (label, "Distorsio");	break;
			case kGain	: strcpy (label, "Ganancia"); break;
	}
}

void ADistorsio::getParameterDisplay (VstInt32 index, char* text) {
	switch (index) {
		case kDistorsio : float2string (fDistorsio, text,
kVstMaxParamStrLen);	break;
		case kGain	: dB2string (fGain, text, kVstMaxParamStrLen);
			break;
	}
}

void ADistorsio::getParameterLabel (VstInt32 index, char* label) {
	switch (index)	{
		case kDistorsio : strcpy (label, "poca -- molta");  break;
		case kGain	: strcpy (label, "dB");			break;
	}
}

bool ADistorsio::getEffectName (char* name) {
	vst_strncpy (name, "Distorsio", kVstMaxEffectNameLen);
	return true;
}

bool ADistorsio::getProductString (char* text) {
	vst_strncpy (text, "Distorsio", kVstMaxProductStrLen);
	return true;
}

bool ADistorsio::getVendorString (char* text) {
	vst_strncpy (text, "Francisco Javier Ripoll Esteve",
kVstMaxVendorStrLen);
	return true;
}

VstInt32 ADistorsio::getVendorVersion () {
	return 1000;
}

void ADistorsio::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {
    float* in1  =  inputs[0];
    float* in2  =  inputs[1];
    float* out1 = outputs[0];
    float* out2 = outputs[1];
	float inleft;
	float inright;
	float distorsio;

    while (--sampleFrames >= 0)
    {
		inleft  = (*in1++);
		inright = (*in2++);
		distorsio = iDistorsio;
```

```
            if (inleft > distorsio) {
                  inleft=distorsio;
            }
            else if (inleft < -distorsio) {
                  inleft = -distorsio;
            }

            if (inright > distorsio) {
                  inright=distorsio;
            }
            else if (inright < -distorsio ) {
                  inright = -distorsio;
            }


            *out1++ = inleft*fGain;
            *out2++=  inright*fGain;

    }
}


void ADistorsio::processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames)
{
    double* in1  =  inputs[0];
    double* in2  =  inputs[1];
    double* out1 = outputs[0];
    double* out2 = outputs[1];

    while (--sampleFrames >= 0)
    {
            double inleft = (*in1++);
            double inright = (*in2++);

            double distorsio= double(iDistorsio);

            if (inleft > distorsio) {
                  inleft=distorsio;
            }
            else if (inleft < -distorsio) {
                  inleft = -distorsio;
            }

            if (inright > distorsio) {
                  inright=distorsio;
            }
            else if (inright < -distorsio ) {
                  inright = -distorsio;
            }


            *out1++ = inleft*fGain;
            *out2++=  inright*fGain;

    }
}
```

Annex V

delay.h

```
//-------------------------------------
// VST Plug-Ins SDK
// DELAY
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"


enum {
	kDelay = 0,
	kDryWet,
	kOut,
	kNumParams
};


class ADelay;


class ADelay : public AudioEffectX {

public:
	ADelay (audioMasterCallback audioMaster);
	~ADelay ();

	virtual void processReplacing (float **inputs, float **outputs, VstInt32
sampleFrames);
	virtual void processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char *label);
	virtual void getParameterDisplay (VstInt32 index, char *text);
	virtual void getParameterName (VstInt32 index, char *text);

	virtual void resume ();

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }

	virtual VstPlugCategory getPlugCategory () {
		return kPlugCategEffect;
	}

protected:
	void setDelay (float delay);

	float *buffer1;
	float *buffer2;
	double *buffer1d;
	double *buffer2d;
```

```
        float fDelay, fDryWet,fOut;

        int delay;
        int L;
        long posicion;
};

#endif
```

Annex VI

delay.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// DELAY
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "delay.h"
#include "math.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new ADelay (audioMaster);
}


ADelay::ADelay (audioMasterCallback audioMaster) : AudioEffectX (audioMaster,
1, 2) {
	L = 8820;	//Un segon 44100, 200 mil·lisegons 8820
	buffer1 = new float[L];
	buffer2 = new float[L];
	buffer1d = new double[L];
	buffer2d = new double[L];

	fDelay = 0.1f;
	fDryWet = 0;

	posicion = 0;

	delay = int(L*fDelay);


	setNumInputs (2);	// stereo input
	setNumOutputs (2);	// stereo output

	setUniqueID ('ADly');

	resume ();		// flush buffer
}


ADelay::~ADelay () {
	if (buffer1)
		delete[] buffer1;

	if (buffer2)
		delete[] buffer2;
}



void ADelay::setDelay (float value)  {
	 fDelay = float(0.001*exp(value*log(0.200/0.001)));
	 delay= int(fDelay*L);
}


//void ADelay::setProgramName (char* name) {
	//vst_strncpy (programName, name, kVstMaxProgNameLen);
```

```
//}


void ADelay::resume () {
	memset (buffer1, 0, L * sizeof (float));
	memset (buffer2, 0, L * sizeof (float));
	AudioEffectX::resume ();
}


void ADelay::setParameter (VstInt32 index, float value) {
	switch (index)	{
		case kDelay  : setDelay(value);  break;
		case kDryWet : fDryWet = (value*100); break;
	}
}


float ADelay::getParameter (VstInt32 index) {
	float v = 0;
	switch (index)	{
		case kDelay  : v = float(log(fDelay/0.001)/(log(0.200/0.001)));
	break;
		case kDryWet : v = fDryWet/100; break;
	}
	return v;
}


void ADelay::getParameterName (VstInt32 index, char* label)
{
	switch (index) {
		case kDelay  : strcpy (label, "Delay");	break;
		case kDryWet : strcpy (label, "Dry -- Wet");  break;
	}
}


void ADelay::getParameterDisplay (VstInt32 index, char* text)
{
		switch (index) {
			case kDelay  : float2string (fDelay, text,
kVstMaxParamStrLen);		break;
			case kDryWet : float2string (fDryWet, text,
kVstMaxParamStrLen);	break;
	}
}


void ADelay::getParameterLabel (VstInt32 index, char* label) {
	switch (index) {
		case kDelay  : strcpy (label, "segons");	break;
		case kDryWet : strcpy (label, "%");		break;
	}
}


bool ADelay::getEffectName (char* name) {
	strcpy (name, "Delay");
	return true;
}


bool ADelay::getProductString (char* text) {
	strcpy (text, "Delay");
	return true;
}
```

```
bool ADelay::getVendorString (char* text) {
	strcpy (text, "Francisco Javier Ripoll Esteve");
	return true;
}


void ADelay::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {
	float* in1 = inputs[0];
	float* in2 = inputs[1];
	float* out1 = outputs[0];
	float* out2 = outputs[1];

	while (--sampleFrames >= 0)	{

		//***** Guardem les entrades en variables*****//
		float inleft = (*in1++);
		float inright = (*in2++);


		//***** Calcul de la posició on es trobarà el frame del retard
*****//
		int posicionretardo = posicion-delay;

		if (posicionretardo<0) {
			posicionretardo= L+posicionretardo;
		}

		//**** Reproducció amb el in actual i el buffer(posició-retard)
****//
		*out1++ = float(inleft*(1-
(fDryWet/100))+buffer1[posicionretardo]*(fDryWet/100.0));
		*out2++=  float(inright*(1-
(fDryWet/100))+buffer2[posicionretardo]*(fDryWet/100.0));


		//***** Guardem el in actual en el buffer actual *****//
		buffer1[posicion]=inleft;
		buffer2[posicion]=inright;


		posicion++;

		if (posicion==L) {
			posicion=0;
		}
	}
}

void ADelay::processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames)
{
    double* in1  =  inputs[0];
    double* in2  =  inputs[1];
    double* out1 = outputs[0];
    double* out2 = outputs[1];


	while (--sampleFrames >= 0)	{



		//***** Guardem les entrades en variables*****//
		double inleft  = (*in1++);
		double inright = (*in2++);
```

```
            //***** Calcul de la posició on es trobarà el frame del retard
*****//
            int posicionretardo = posicion-delay;

            if (posicionretardo<0) {
                  posicionretardo= L+posicionretardo;
            }


            //**** Reproducció amb el in actual i el buffer(posició-retard)
****//
            *out1++ = double(inleft*(1-
(fDryWet/100))+buffer1d[posicionretardo]*(fDryWet/100.0));
            *out2++=  double(inright*(1-
(fDryWet/100))+buffer2d[posicionretardo]*(fDryWet/100.0));


            //***** Guardem el in actual en el buffer actual *****//
            buffer1d[posicion]=inleft;
            buffer2d[posicion]=inright;


            posicion++;

            if (posicion==L) {
                  posicion=0;
            }
      }
}
```

Annex VII

eco.h

```
//-------------------------------------
// VST Plug-Ins SDK
// ECO
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"


enum
{
	kDelay = 0,
	kDryWet,
	kFeed,
	kNumParams
};



class AEco;


class AEco : public AudioEffectX {
public:
	AEco (audioMasterCallback audioMaster);
	~AEco ();

	virtual void processReplacing (float **inputs, float **outputs, VstInt32
sampleFrames);
	virtual void processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char *label);
	virtual void getParameterDisplay (VstInt32 index, char *text);
	virtual void getParameterName (VstInt32 index, char *text);

	virtual void resume ();

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }

	virtual VstPlugCategory getPlugCategory () {
		return kPlugCategEffect;
	}

protected:
	void setDelay (float delay);


	float *buffer1;
	float *buffer2;
```

```
        double *buffer1d;
        double *buffer2d;
        float fDelay, fDryWet,fFeed;

        int delay;
        int L;
        long posicion;
};

#endif
```

Annex VIII

eco.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// ECO
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "eco.h"
#include "math.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new AEco (audioMaster);
}


AEco::AEco (audioMasterCallback audioMaster) : AudioEffectX (audioMaster, 1,
3)
{
	L = 44100*4;
	buffer1 = new float[L];
	buffer2 = new float[L];
	buffer1d = new double[L];
	buffer2d = new double[L];

	fDelay = 0.5;
	fDryWet = 0;
	fFeed = 0;

	posicion = 0;

	delay = int(L*fDelay);


	setNumInputs (2);   // stereo input
	setNumOutputs (2);  // stereo output

	setUniqueID ('ADly');

	resume ();          // flush buffer
}


AEco::~AEco () {
	if (buffer1)
		delete[] buffer1;

	if (buffer2)
		delete[] buffer2;
}


void AEco::setDelay (float value) {
	 fDelay = float(0.002*exp(value*log(4.0/0.002)));
	 delay= int(fDelay*L/4);
}


void AEco::resume () {
```

```
	memset (buffer1, 0, L * sizeof (float));
	memset (buffer2, 0, L * sizeof (float));
	AudioEffectX::resume ();
}


void AEco::setParameter (VstInt32 index, float value) {

	switch (index)
	{
		case kDelay  : setDelay(value);		break;

		case kDryWet : fDryWet = (value*100);  break;
		case kFeed   : fFeed = (value*100);	break;
	}
}


float AEco::getParameter (VstInt32 index)
{
		float v = 0;

	switch (index)	{
		case kDelay  : v = float(log(fDelay/0.002)/(log(4.0/0.002)));
	break;
		case kDryWet : v = fDryWet/100;
				break;
		case kFeed   : v = fFeed/100;
				break;
	}
	return v;
}


void AEco::getParameterName (VstInt32 index, char* label) {
	switch (index)	{
	case kDelay	: strcpy (label, "Delay");		break;
		case kDryWet : strcpy (label, "Dry -- Wet");  break;
		case kFeed   : strcpy (label, "FeedBack");	break;
	}
}


void AEco::getParameterDisplay (VstInt32 index, char* text) {
		switch (index)	{
		case kDelay  : float2string (fDelay, text, kVstMaxParamStrLen);
		break;
		case kDryWet : float2string (fDryWet, text, kVstMaxParamStrLen);
	break;
		case kFeed   : float2string (fFeed, text, kVstMaxParamStrLen);
		break;
	}
}


void AEco::getParameterLabel (VstInt32 index, char* label) {
	switch (index)	{
		case kDelay  : strcpy (label, "segons");	break;
		case kDryWet : strcpy (label, "%");		break;
		case kFeed   : strcpy (label, "%");		break;
	}
}


bool AEco::getEffectName (char* name) {
	strcpy (name, "Eco");
	return true;
}
```

```
bool AEco::getProductString (char* text) {
	strcpy (text, "Eco");
	return true;
}


bool AEco::getVendorString (char* text) {
	strcpy (text, "Francisco Javier Ripoll Esteve");
	return true;
}


void AEco::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {
	float* in1 = inputs[0];
	float* in2 = inputs[1];
	float* out1 = outputs[0];
	float* out2 = outputs[1];

	while (--sampleFrames >= 0)	{

		//***** Guardem les entrades en variables*****//
		float inleft = (*in1++);
		float inright = (*in2++);


		//***** Calcul de la posició on es trobarà el frame del retard
*****//
		int posicionretardo = posicion-delay;

		if (posicionretardo<0) {
			posicionretardo= L+posicionretardo;
		}


		//**** Reproducció amb el in actual i el buffer(posició-retard)
****//

		float outleft = float(inleft*(1-
(fDryWet/100))+buffer1[posicionretardo]*(fDryWet/100));
		float outright = float(inright*(1-
(fDryWet/100))+buffer2[posicionretardo]*(fDryWet/100));
		*out1++ = outleft;
		*out2++=  outright;


		//***** Guardem el in actual en el buffer actual *****//
		buffer1[posicion]=(1-(fFeed/100))*inleft+(fFeed/100)*outleft;
		buffer2[posicion]=(1-(fFeed/100))*inright+(fFeed/100)*outright;



		posicion++;

		if (posicion==L) {
			posicion=0;
		}
	}
}


void AEco::processDoubleReplacing (double** inputs, double** outputs, VstInt32
sampleFrames) {
	double* in1 = inputs[0];
	double* in2 = inputs[1];
	double* out1 = outputs[0];
```

```
	double* out2 = outputs[1];

	while (--sampleFrames >= 0)	{


		//***** Guardem les entrades en variables*****//
		double inleft = (*in1++);
		double inright = (*in2++);


		//***** Calcul de la posició on es trobarà el frame del retard
*****//
		int posicionretardo = posicion-delay;

		if (posicionretardo<0) {
			posicionretardo= L+posicionretardo;
		}


		//**** Reproducció amb el in actual i el buffer(posició-retard)
****//

		double outleft = double(inleft*(1.0-
(fDryWet/100.0))+buffer1d[posicionretardo]*(fDryWet/100.0));
		double outright = double(inright*(1.0-
(fDryWet/100.0))+buffer2d[posicionretardo]*(fDryWet/100.0));
		*out1++ = outleft;
		*out2++=  outright;


		//***** Guardem el in actual en el buffer actual *****//

		buffer1d[posicion]=(1.0-
(fFeed/100.0))*inleft+(fFeed/100.0)*outleft;
		buffer2d[posicion]=(1-
(fFeed/100.0))*inright+(fFeed/100.0)*outright;



		posicion++;

		if (posicion==L) {
			posicion=0;
		}
	}
}
```

Annex IX

chorus.h

```
//-------------------------------------
// VST Plug-Ins SDK
// CHORUS
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"


enum {
	kNumPrograms = 1,

	kDryWet = 0,
	kGain,
	kFreq,
	kDelayMin,
	kDelayMax,

};



class AChorus;


class AChorus : public AudioEffectX {
public:
	AChorus (audioMasterCallback audioMaster);
	~AChorus ();

	virtual void processReplacing (float **inputs, float **outputs, VstInt32
sampleFrames);
	virtual void processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char *label);
	virtual void getParameterDisplay (VstInt32 index, char *text);
	virtual void getParameterName (VstInt32 index, char *text);

	virtual void resume ();

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }

	virtual VstPlugCategory getPlugCategory () { return kPlugCategEffect; }

protected:
	void setDelayMax (float delay);
	void setDelayMin (float delay);


	float *buffer1;
	float *buffer2;
```

```
        double *buffer1d;
        double *buffer2d;
        float listate[4];
        int outPoint[4];
        double feedbackL, feedbackR;
        float fDryWet,fGain,fFreq,fDelayMin, fDelayMax;
        float freq;

        int  delayMin, delayMax;
        int L;
        int n;
        long posicion;
        int sampleCount;
};

#endif
```

Annex X

chorus.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// CHORUS
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "chorus.h"
#include "math.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new AChorus (audioMaster);
}


AChorus::AChorus (audioMasterCallback audioMaster) : AudioEffectX
(audioMaster, 1, 5) {
	L = 44100;
	sampleCount = 0;
	buffer1 = new float[L];
	buffer2 = new float[L];
	buffer1d = new double[L];
	buffer2d = new double[L];

	//inicialitzem els buffers
	for (int i=0;i<L;i++) {
		buffer1[i] = 0.f;
		buffer2[i] = 0.f;
		buffer1d[i] = 0;
		buffer2d[i] = 0;
	}

	fGain=  1.f;
	fFreq = 2.5f;
	fDelayMin = 0.2f;
	fDelayMax = 0.3f;
	fDryWet = 0.5;
	n=0;

	posicion = 0;

	delayMin = int (L*fDelayMin);
	delayMax = int (L*fDelayMax);


	setNumInputs (2);   // stereo input
	setNumOutputs (2);  // stereo output

	setUniqueID ('Acho');

	resume ();
}


AChorus::~AChorus () {
	if (buffer1)
		delete[] buffer1;

	if (buffer2)
		delete[] buffer2;
```

```
	if (buffer1d)
		delete[] buffer1d;

	if (buffer2d)
		delete[] buffer2d;
}

void AChorus::setDelayMin (float value)  {
	 fDelayMin = value;
	 delayMin= int(value*L);
}

void AChorus::setDelayMax (float value) {
	 fDelayMax = value;
	 delayMax= int(value*L);
}

void AChorus::resume () {
	memset (buffer1, 0, L * sizeof (float));
	memset (buffer2, 0, L * sizeof (float));
	AudioEffectX::resume ();
}

void AChorus::setParameter (VstInt32 index, float value) {

	switch (index)	{
		case kDryWet   : fDryWet = value;		break;
		case kGain     : fGain = value*2;	break;
		case kFreq     : fFreq = value * 5.f;	break;
		case kDelayMin : setDelayMin(value);	break;
		case kDelayMax : setDelayMax(value);	break;
	}
}


float AChorus::getParameter (VstInt32 index) {
	float v = 0;

	switch (index)	{
		case kDryWet   : v = fDryWet;
				 break;
		case kGain     : v = fGain/2;
				 break;
		case kFreq     : v = fFreq / 5.f;
				 break;
		case kDelayMin : v = fDelayMin;
				 break;
		case kDelayMax : v = fDelayMax;
				 break;
	}
	return v;
}

void AChorus::getParameterName (VstInt32 index, char* label) {
	switch (index)	{
		case kDryWet   : strcpy (label, "Dry -- Wet");	break;
		case kGain     : strcpy(label, "Ganancia");	break;
		case kFreq     : strcpy(label, "Dealy Freq"); break;
		case kDelayMin : strcpy (label, "DelayMin");  break;
		case kDelayMax : strcpy (label, "DelayMax");  break;
	}
}

void AChorus::getParameterDisplay (VstInt32 index, char* text) {
		switch (index)	{
		case kDryWet   : float2string (fDryWet*100, text,
kVstMaxParamStrLen);		 break;
```

```
            case kGain     : dB2string(fGain, text, kVstMaxProgNameLen);
                             break;
            case kFreq     : float2string(fFreq, text, kVstMaxProgNameLen);
                             break;
            case kDelayMin : float2string (fDelayMin*1000, text,
kVstMaxParamStrLen);      break;
            case kDelayMax : float2string (fDelayMax*1000, text,
kVstMaxParamStrLen);      break;
      }
}

void AChorus::getParameterLabel (VstInt32 index, char* label) {
      switch (index)      {
            case kDryWet   : strcpy (label, "%");         break;
            case kGain     : strcpy(label, "dB");         break;
            case kFreq     : strcpy(label, "Hz");         break;
            case kDelayMin : strcpy (label, "ms");        break;
            case kDelayMax : strcpy (label, "ms");        break;
      }
}

bool AChorus::getEffectName (char* name) {
      strcpy (name, "Delay");
      return true;
}

bool AChorus::getProductString (char* text) {
      strcpy (text, "Delay");
      return true;
}


bool AChorus::getVendorString (char* text) {
      strcpy (text, "Francisco Javier Ripoll Esteve");
      return true;
}

void AChorus::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {


    float *in1  =  inputs[0];
    float *in2  =  inputs[1];
    float *out1 =  outputs[0];
    float *out2 =  outputs[1];
      float T = 1/44100;
      int di=0;
      double pi = 3.1415926;
      double nT=0, deltai=0, t0=0, t1=0, t2=0, t3=0, b0=0, b1=0, b2=0, b3=0;

    while (--sampleFrames >= 0) {
            nT = 1/44100 * (n++);
            deltai = (delayMax-delayMin)/2 + delayMin + (delayMax-delayMin)/2
* sin(2 * pi * fFreq * nT);
            // Delay line emprant Lagrange interpolator
            di = floorf(deltai) - 1;
            t0 = deltai - di;
            t1 = t0 - 1;
            t2 = t0 - 2;
            t3 = t0 - 3;
            b0 = -t1*t2*t3/6;
            b1 =  t0*t2*t3/2;
            b2 = -t0*t1*t3/2;
            b3 =  t0*t1*t2/6;


            if (posicion >= L)
                  posicion = 0;
```

```
            //Esquerra
            buffer1[posicion] = fGain*(*in1);
            for (int i=0;i<4;i++) {
                  outPoint[i] = (posicion - 1 - di + i);
                  if (outPoint[i] < 0)
                        outPoint[i] += L;
                  else if (outPoint[i] >= L)
                        outPoint[i] -= L;
                  listate[i] = buffer1[outPoint[i]];
            }

            feedbackL = (b0*listate[0] + b1*listate[1] + b2*listate[2] +
b3*listate[3]);


            //Dreta
            buffer2[posicion] = fGain*(*in2);
            for (int j=0;j<4;j++) {
                  outPoint[j] = (posicion - 1 - di + j);
                  if (outPoint[j] < 0)
                        outPoint[j] += L;
                  if (outPoint[j] >= L)
                        outPoint[j] -= L;
                  listate[j] = buffer2[outPoint[j]];
            }

            feedbackR = (b0*listate[0] + b1*listate[1] + b2*listate[2] +
b3*listate[3]);

            // ixida
            (*out1++) = (*in1++)*fDryWet + (1-fDryWet) * feedbackL;
            (*out2++) = (*in2++)*fDryWet + (1-fDryWet) * feedbackR;

            posicion++;

   }
}

void AChorus::processDoubleReplacing (double** inputs, double** outputs,
VstInt32 sampleFrames) {

      double *in1  =  inputs[0];
    double *in2  =  inputs[1];
    double *out1 =  outputs[0];
    double *out2 =  outputs[1];
      double T = 1/44100;
      int di=0;
      double pi = 3.1415926;
      double nT=0, deltai=0, t0=0, t1=0, t2=0, t3=0, b0=0, b1=0, b2=0, b3=0;

    while (--sampleFrames >= 0) {
            nT = 1/44100 * (n++);
            deltai = (delayMax-delayMin)/2 + delayMin + (delayMax-delayMin)/2
* sin(2 * pi * fFreq * nT);
            // Delay line emprant Lagrange interpolator
            di = floorf(deltai) - 1;
            t0 = deltai - di;
            t1 = t0 - 1;
            t2 = t0 - 2;
            t3 = t0 - 3;
            b0 = -t1*t2*t3/6;
            b1 =  t0*t2*t3/2;
            b2 = -t0*t1*t3/2;
            b3 =  t0*t1*t2/6;


            if (posicion >= L)
```

```
                  posicion = 0;

            //Esquerra
            buffer1[posicion] = fGain*(*in1);
            for (int i=0;i<4;i++) {
                  outPoint[i] = (posicion - 1 - di + i);
                  if (outPoint[i] < 0)
                        outPoint[i] += L;
                  else if (outPoint[i] >= L)
                        outPoint[i] -= L;
                  listate[i] = buffer1[outPoint[i]];
            }

            feedbackL = (b0*listate[0] + b1*listate[1] + b2*listate[2] +
b3*listate[3]);


            //Dreta
            buffer2[posicion] = fGain*(*in2);
            for (int j=0;j<4;j++) {
                  outPoint[j] = (posicion - 1 - di + j);
                  if (outPoint[j] < 0)
                        outPoint[j] += L;
                  if (outPoint[j] >= L)
                        outPoint[j] -= L;
                  listate[j] = buffer2[outPoint[j]];
            }

            feedbackR = (b0*listate[0] + b1*listate[1] + b2*listate[2] +
b3*listate[3]);

            // ixida
            (*out1++) = (*in1++)*fDryWet + (1-fDryWet) * feedbackL;
            (*out2++) = (*in2++)*fDryWet + (1-fDryWet) * feedbackR;

            posicion++;

   }
}
```

Annex XI

generador.h

```
//---------------------------------------
// VST Plug-Ins SDK
// GENERADOR
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 06/06/2008
//---------------------------------------

#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"

const  float  FreqMax = 20000; // 20 kHz
const  float  FreqMin = 20; // 20 Hz


enum {
	kFrequencia = 0,
	kAmplitud,
	kForma
};

enum{
	FormaSinus = 0,
	FormaQuadra,
	FormaTriangle,
	NombreDeFormes
};


class AGenerador;


class AGenerador : public AudioEffectX {

public:
	AGenerador (audioMasterCallback audioMaster);
	~AGenerador ();

	virtual void processReplacing (float **inputs, float **outputs, VstInt32
sampleFrames);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char *label);
	virtual void getParameterDisplay (VstInt32 index, char *text);
	virtual void getParameterName (VstInt32 index, char *text);

	virtual void resume ();

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }

	virtual VstPlugCategory getPlugCategory () {
		return kPlugCategEffect;
	}
```

```
protected:

	float fAmplitud, fFrequencia;
	int fForma;
	int tipus;
	double i;
	float f;
	float pi;

};

#endif
```

Annex XII

generador.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// GENERADOR
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 13/05/2008
//-------------------------------------

#include "generador.h"
#include "math.h"


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new AGenerador (audioMaster);
}


AGenerador::AGenerador (audioMasterCallback audioMaster) : AudioEffectX
(audioMaster, 1, 3) {

	f = AudioEffectX::updateSampleRate ();
	fAmplitud = 0.5;
	fFrequencia = 1000/f;
	fForma = FormaSinus;
	tipus = 0;
	i=0;

	pi = 3.14159265358979323846264338 3f;

	setNumOutputs (1);  // mono

	setUniqueID ('AGen');

	resume ();          // flush buffer
}


AGenerador::~AGenerador () {

}


void AGenerador::resume () {
	AudioEffectX::resume ();
}


void AGenerador::setParameter (VstInt32 index, float value) {
	switch (index)	{
		case kFrequencia : fFrequencia = FreqMin *
exp(log(FreqMax/FreqMin)*value) / f;   break;
		case kAmplitud   : fAmplitud = value;
                                        break;
		case kForma      : fForma = int (float (NombreDeFormes - 1) *
value );          break;
	}
}


float AGenerador::getParameter (VstInt32 index) {
	float v = 0;
	switch (index)	{
```

```
            case kFrequencia : v = log(fFrequencia*f/FreqMin) / log
(FreqMax/FreqMin);  break;
            case kAmplitud   : v = fAmplitud;      break;
            case kForma      : v = float (fForma) / float (NombreDeFormes -
1);
      }
      return v;
}


void AGenerador::getParameterName (VstInt32 index, char* label)
{
      switch (index) {
            case kFrequencia : strcpy (label, "Freqüencia");           break;
            case kAmplitud   : strcpy (label, "Amplitud");
      break;
            case kForma      : strcpy (label, "Forma de senyal");      break;
      }
}


void AGenerador::getParameterDisplay (VstInt32 index, char* text)
{
            switch (index) {
                  case kFrequencia : float2string (fFrequencia*f, text,
kVstMaxParamStrLen);            break;
                  case kAmplitud   : float2string (fAmplitud*100, text,
kVstMaxParamStrLen);            break;
                  case kForma      : switch (fForma) {
                                                case FormaSinus    :
vst_strncpy (text, "Sinus", kVstMaxParamStrLen);    break;
                                                case FormaQuadra   :
vst_strncpy (text, "Quadrat", kVstMaxParamStrLen);  break;
                                                case FormaTriangle :
vst_strncpy (text, "Triangle", kVstMaxParamStrLen); break;
                                          } break;
      }
}


void AGenerador::getParameterLabel (VstInt32 index, char* label) {
      switch (index) {
            case kFrequencia : strcpy (label, "Hz");      break;
            case kAmplitud   : strcpy (label, "%");       break;
            case kForma      : strcpy (label, "Tipus"); break;
      }
}


bool AGenerador::getEffectName (char* name) {
      strcpy (name, "Generador");
      return true;
}


bool AGenerador::getProductString (char* text) {
      strcpy (text, "Generador");
      return true;
}


bool AGenerador::getVendorString (char* text) {
      strcpy (text, "Francisco Javier Ripoll Esteve");
      return true;
}
```

```
void AGenerador::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {

	float* out = outputs[0];
	double fase;
	tipus =1;

	while (--sampleFrames >= 0)	{
		fase = fmod(fFrequencia*i,1);
		i++;


		switch (fForma) {
		case FormaSinus :
			*out++ = fAmplitud*sin(2*pi*fase);
			break;
		case FormaQuadra :
			if(fase<0.5) {
				*out++ = fAmplitud;
			}
			else {
				*out++ = -fAmplitud;
			}
			break;
		case FormaTriangle :
			if(fase <= 0.5) {
				// y = m*x + b
				// m= DY / DX    DY = 2*Amplitud    DX = 1/2
				// x = fase
				// b = -Amplitud
				float m = 2 * fAmplitud / 0.5f;
				*out++ = m * fase - fAmplitud;
			}
			else {
				// y = m*x + b
				// m= DY / DX    DY = -2*Amplitud    DX = 1/2
				// x = (fase-0.5)   per a dur-ho a x=0
				// b = Amplitud

				float m = -2 * fAmplitud / 0.5f;
				*out++ = m * (fase - 0.5f) + fAmplitud;

			}

			break;

		}  //switch
	}

}
```

Annex XIII

sint.h

```
//-------------------------------------
// VST Plug-Ins SDK
// SINTETISADOR
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 18/06/2008
//-------------------------------------

#ifndef __again__
#define __again__

#include "public.sdk/source/vst2.x/audioeffectx.h"

const  float  FreqMax = 20000; // 20 kHz
const  float  FreqMin = 20; // 20 Hz


enum {
	kForma =0,
	kAttack,
	kDecay,
	kRelease,
	kSustain,
	kTecles
};

enum {
	FormaSinus = 0,
	FormaQuadra,
	FormaTriangle,
	NombreDeFormes
};

enum {
	EOff = 0,
	EAttack,
	EDecay,
	ESustain,
	ERelease
};

struct nota_struct {
	bool noteIsOn;
	VstInt32 currentNote;
	VstInt32 Velocity;
	VstInt32 Delta;
	float fPhase1, fPhase2;
	int estat;
	int CAttack, CDecay, CRelease;
	float NoteFreq;
	float salida;

};


class ASinteti;


class ASinteti : public AudioEffectX {
```

```
public:
	ASinteti (audioMasterCallback audioMaster);
	~ASinteti ();

	virtual void processReplacing (float **inputs, float **outputs, VstInt32
sampleFrames);
	virtual VstInt32 processEvents (VstEvents* events);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char *label);
	virtual void getParameterDisplay (VstInt32 index, char *text);
	virtual void getParameterName (VstInt32 index, char *text);

	virtual void resume ();

	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; }

	virtual VstPlugCategory getPlugCategory () {
		return kPlugCategEffect;
	}

protected:


	int fForma;
	double i;
	float f;
	float pi;
	nota_struct nota[85];
	float volum;
	int TeclesMaximes;
	float TTeclesMaximes;
	int TeclesActives;

	/*float fPhase1, fPhase2;
	VstInt32 currentNote;
	VstInt32 currentVelocity;
	VstInt32 currentDelta;
	bool noteIsOn;*/

	//int estat;
	float TAttack, TDecay,TRelease, PSustain;
	float fHeadRoom;
	//int CAttack, CDecay, CRelease;

	float conversorPosicionenFrecuencia(int j);
	int conversorNotaenPosicion(VstInt32 Nota);
	void noteOn (VstInt32 note, VstInt32 velocity, VstInt32 delta);
	void noteOff (VstInt32 note);
	void noteOffAll();
	void initProcess ();

};

#endif
```

Annex XIV

sint.cpp

```
//-------------------------------------
// VST Plug-Ins SDK
// SINTETISADOR
//
// Francisco Javier Ripoll Esteve
// ripollete@hotmail.com
// 18/06/2008
//-------------------------------------

#include "sintetisador.h"
#include "math.h"
#include <stdio.h>


AudioEffect* createEffectInstance (audioMasterCallback audioMaster) {
	return new ASinteti (audioMaster);
}


void ASinteti::initProcess () {

	//Inicialitzem totes les posicions en valors inicials
	for(int pos=0; pos<85; pos++) {
		nota[pos].noteIsOn = false;
		nota[pos].estat = EOff;
		nota[pos].CAttack = 0;
		nota[pos].CDecay = 0;
		nota[pos].CRelease = 0;
		nota[pos].fPhase1 = 0;
		nota[pos].fPhase2 = 0;
		nota[pos].Delta = nota[pos].Velocity = 0;
		nota[pos].NoteFreq = conversorPosicionenFrecuencia(pos);
		nota[pos].salida = 0.0f;
	}

	f = AudioEffectX::updateSampleRate ();
	fForma = FormaSinus;
	i=0;

	TAttack = 22000.f;
	TDecay = 22000.f;
	TRelease = 22000.f;
	PSustain = 0.5f;
	fHeadRoom = 0.5f;
	volum = 1.0f;
	TeclesMaximes = 5;
	TTeclesMaximes = 0.5f;
	TeclesActives = 0;

	pi = 3.14159265358979323846264338f;

}


ASinteti::ASinteti (audioMasterCallback audioMaster) : AudioEffectX
(audioMaster, 1, 6) {
	if (audioMaster) {
		setNumInputs (0);
		setNumOutputs (1);
		canProcessReplacing ();
		isSynth ();
```

```
		setUniqueID ('Sint');
	}

	initProcess ();
	suspend ();
}


ASinteti::~ASinteti () {

}


void ASinteti::resume () {
	AudioEffectX::resume ();
}


void ASinteti::setParameter (VstInt32 index, float value) {
	switch (index)	{
		case kForma	: fForma = int (float (NombreDeFormes - 1) *
value );	break;
		case kAttack  : TAttack = value * f;
				break;
		case kDecay	 : TDecay = value * f;
				break;
		case kRelease : TRelease = value * f;
			break;
		case kSustain	: PSustain = value;
				break;
		case kTecles	: TTeclesMaximes = value;
					TeclesMaximes =
int(TTeclesMaximes*10.0f);		break;
	}
}


float ASinteti::getParameter (VstInt32 index) {
	float v = 0;
	switch (index)	{
		case kForma	: v = float (fForma) / float (NombreDeFormes -
1);		break;
		case kAttack  : v = TAttack / f;
				break;
		case kDecay	 : v = TDecay / f;
				break;
		case kRelease : v = TRelease / f;
				break;
		case kSustain	: v = PSustain;
				break;
		case kTecles   : v = TTeclesMaximes;
				break;
	}
	return v;
}


void ASinteti::getParameterName (VstInt32 index, char* label)
{
	switch (index) {
		case kForma	: strcpy (label, "Forma de senyal");	break;
		case kAttack  : strcpy (label, "Temps Attack");		break;

		case kDecay	 : strcpy (label, "Temps Decay");
	break;
		case kRelease : strcpy (label, "Temps Release");		break;
		case kSustain	: strcpy (label, "Sustain");		break;
```

```
            case kTecles   : strcpy (label, "TeclesMaximes");
      break;
      }
}


void ASinteti::getParameterDisplay (VstInt32 index, char* text)
{
            switch (index) {
                  case kForma      : switch (fForma) {
                                                case FormaSinus    :
vst_strncpy (text, "Sinus", kVstMaxParamStrLen);    break;
                                                case FormaQuadra   :
vst_strncpy (text, "Quadrat", kVstMaxParamStrLen);  break;
                                                case FormaTriangle :
vst_strncpy (text, "Triangle", kVstMaxParamStrLen); break;
                                                //case FormaSinus :
vst_strncpy (text, "Sinus", kVstMaxParamStrLen); break;
                                               } break;
                  case kAttack  : float2string (TAttack/f, text,
kVstMaxParamStrLen);    break;
                  case kDecay          : float2string (TDecay/f, text,
kVstMaxParamStrLen);     break;
                  case kRelease : float2string (TRelease/f, text,
kVstMaxParamStrLen);   break;
                  case kSustain    : float2string (PSustain, text,
kVstMaxParamStrLen);     break;
                  case kTecles   : int2string    (TeclesMaximes, text,
kVstMaxParamStrLen);        break;
      }
}


void ASinteti::getParameterLabel (VstInt32 index, char* label) {
      switch (index) {
            case kForma      : strcpy (label, "Tipus");          break;
            case kAttack  : strcpy (label, "ms");         break;
            case kDecay          : strcpy (label, "ms");         break;
            case kRelease : strcpy (label, "ms");         break;
            case kSustain    : strcpy (label, "%");              break;
            case kTecles   : strcpy (label, "Tecles");          break;
      }
}


bool ASinteti::getEffectName (char* name) {
      strcpy (name, "Generador");
      return true;
}


bool ASinteti::getProductString (char* text) {
      strcpy (text, "Generador");
      return true;
}


bool ASinteti::getVendorString (char* text) {
      strcpy (text, "Francisco Javier Ripoll Esteve");
      return true;
}


float ASinteti::conversorPosicionenFrecuencia(int pos) {
      if (pos == 0) return 32.71f; // C octava1
      else if (pos == 1) return 34.65f;
      else if (pos == 2) return 36.71f;
      else if (pos == 3) return 38.89f;
```

```
else if (pos == 4) return 41.20f;
else if (pos == 5) return 43.65f;
else if (pos == 6) return 46.25f;
else if (pos == 7) return 49.0f;
else if (pos == 8) return 51.91f;
else if (pos == 9) return 55.0f;
else if (pos == 10) return 58.27f;
else if (pos == 11) return 61.74f;
else if (pos == 12) return 65.41f; // C octava2
else if (pos == 13) return 69.30f;
else if (pos == 14) return 73.42f;
else if (pos == 15) return 77.78f;
else if (pos == 16) return 82.41f;
else if (pos == 17) return 87.31f;
else if (pos == 18) return 92.50f;
else if (pos == 19) return 98.0f;
else if (pos == 20) return 103.83f;
else if (pos == 21) return 110.0f;
else if (pos == 22) return 116.54f;
else if (pos == 23) return 123.47f;
else if (pos == 24) return 130.81f; // C octava 3
else if (pos == 25) return 138.59f;
else if (pos == 26) return 146.83f;
else if (pos == 27) return 155.56f;
else if (pos == 28) return 164.81f;
else if (pos == 29) return 174.61f;
else if (pos == 30) return 185.0f;
else if (pos == 31) return 196.0f;
else if (pos == 32) return 207.65f;
else if (pos == 33) return 220.0f;
else if (pos == 34) return 233.08f;
else if (pos == 35) return 246.94f;
else if (pos == 36) return 261.63f;  // C octava 4
else if (pos == 37) return 277.18f;
else if (pos == 38) return 293.66f;
else if (pos == 39) return 311.13f;
else if (pos == 40) return 329.63f;
else if (pos == 41) return 349.23f;
else if (pos == 42) return 369.99f;
else if (pos == 43) return 392.0f;
else if (pos == 44) return 415.30f;
else if (pos == 45) return 444.0f;
else if (pos == 46) return 466.16f;
else if (pos == 47) return 493.88f;
else if (pos == 48) return 523.25f;  // C octava 5
else if (pos == 49) return 554.37f;
else if (pos == 50) return 587.33f;
else if (pos == 51) return 622.25f;
else if (pos == 52) return 659.26f;
else if (pos == 53) return 698.46f;
else if (pos == 54) return 739.99f;
else if (pos == 55) return 783.99f;
else if (pos == 56) return 830.61f;
else if (pos == 57) return 880.0f;
else if (pos == 58) return 932.33f;
else if (pos == 59) return 987.77f;
else if (pos == 60) return 1046.50f; // C octava 6
else if (pos == 61) return 1108.73f;
else if (pos == 62) return 1174.66f;
else if (pos == 63) return 1244.51f;
else if (pos == 64) return 1318.51f;
else if (pos == 65) return 1396.91f;
else if (pos == 66) return 1479.98f;
else if (pos == 67) return 1567.98f;
else if (pos == 68) return 1661.22f;
else if (pos == 69) return 1760.0f;
else if (pos == 70) return 1864.66f;
else if (pos == 71) return 1975.53f;
```

```
	else if (pos == 72) return 2093.0f;
	else if (pos == 73) return 2217.46f;
	else if (pos == 74) return 2349.32f;
	else if (pos == 75) return 2489.02f;
	else if (pos == 76) return 2637.02f;
	else if (pos == 77) return 2793.83f;
	else if (pos == 78) return 2959.96f;
	else if (pos == 79) return 3135.96f;
	else if (pos == 80) return 3322.44f;
	else if (pos == 81) return 3520.0f;
	else if (pos == 82) return 3729.31f;
	else if (pos == 83) return 3951.07f;
	else if (pos == 84) return 4186.01f; // C octava 7


}


int ASinteti::conversorNotaenPosicion(VstInt32 Nota) {
	if (Nota == 0x18) return 0; // C octava1
	else if (Nota == 0x19) return 1;
	else if (Nota == 0x1A) return 2;
	else if (Nota == 0x1B) return 3;
	else if (Nota == 0x1C) return 4;
	else if (Nota == 0x1D) return 5;
	else if (Nota == 0x1E) return 6;
	else if (Nota == 0x1F) return 7;
	else if (Nota == 0x20) return 8;
	else if (Nota == 0x21) return 9;
	else if (Nota == 0x22) return 10;
	else if (Nota == 0x23) return 11;
	else if (Nota == 0x24) return 12; // C octava2
	else if (Nota == 0x25) return 13;
	else if (Nota == 0x26) return 14;
	else if (Nota == 0x27) return 15;
	else if (Nota == 0x28) return 16;
	else if (Nota == 0x29) return 17;
	else if (Nota == 0x2A) return 18;
	else if (Nota == 0x2B) return 19;
	else if (Nota == 0x2C) return 20;
	else if (Nota == 0x2D) return 21;
	else if (Nota == 0x2E) return 22;
	else if (Nota == 0x2F) return 23;
	else if (Nota == 0x30) return 24; // C octava 3
	else if (Nota == 0x31) return 25;
	else if (Nota == 0x32) return 26;
	else if (Nota == 0x33) return 27;
	else if (Nota == 0x34) return 28;
	else if (Nota == 0x35) return 29;
	else if (Nota == 0x36) return 30;
	else if (Nota == 0x37) return 31;
	else if (Nota == 0x38) return 32;
	else if (Nota == 0x39) return 33;
	else if (Nota == 0x3A) return 34;
	else if (Nota == 0x3B) return 35;
	else if (Nota == 0x3C) return 36;  // C octava 4
	else if (Nota == 0x3D) return 37;
	else if (Nota == 0x3E) return 38;
	else if (Nota == 0x3F) return 39;
	else if (Nota == 0x40) return 40;
	else if (Nota == 0x41) return 41;
	else if (Nota == 0x42) return 42;
	else if (Nota == 0x43) return 43;
	else if (Nota == 0x44) return 44;
	else if (Nota == 0x45) return 45;
	else if (Nota == 0x46) return 46;
	else if (Nota == 0x47) return 47;
```

```
	else if (Nota == 0x48) return 48;  // C octava 5
	else if (Nota == 0x49) return 49;
	else if (Nota == 0x4A) return 50;
	else if (Nota == 0x4B) return 51;
	else if (Nota == 0x4C) return 52;
	else if (Nota == 0x4D) return 53;
	else if (Nota == 0x4E) return 54;
	else if (Nota == 0x4F) return 55;
	else if (Nota == 0x50) return 56;
	else if (Nota == 0x51) return 57;
	else if (Nota == 0x52) return 58;
	else if (Nota == 0x53) return 59;
	else if (Nota == 0x54) return 60; // C octava 6
	else if (Nota == 0x55) return 61;
	else if (Nota == 0x56) return 62;
	else if (Nota == 0x57) return 63;
	else if (Nota == 0x58) return 64;
	else if (Nota == 0x59) return 65;
	else if (Nota == 0x5A) return 66;
	else if (Nota == 0x5B) return 67;
	else if (Nota == 0x5C) return 68;
	else if (Nota == 0x5D) return 69;
	else if (Nota == 0x5E) return 70;
	else if (Nota == 0x5F) return 71;
	else if (Nota == 0x60) return 72;
	else if (Nota == 0x61) return 73;
	else if (Nota == 0x62) return 74;
	else if (Nota == 0x63) return 75;
	else if (Nota == 0x64) return 76;
	else if (Nota == 0x65) return 77;
	else if (Nota == 0x66) return 78;
	else if (Nota == 0x67) return 79;
	else if (Nota == 0x68) return 80;
	else if (Nota == 0x69) return 81;
	else if (Nota == 0x6A) return 82;
	else if (Nota == 0x6B) return 83;
	else if (Nota == 0x6C) return 84; // C octava 7
}


void ASinteti::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames) {

	float* out = outputs[0];
	double fase;
	float eixida;
	float PAttack; //Percentatge Attack
	float PDecay;
	float PRelease;
	float fFrequencia;
	float fAmplitud;

	while (--sampleFrames >= 0)	{
		for(int pos=0;pos<85;pos++){
			if(nota[pos].noteIsOn) {

				fFrequencia = nota[pos].NoteFreq / f;
				fase = fmod(fFrequencia*i,1);
				fAmplitud = double(nota[pos].Velocity)/127.0f;

				switch (fForma) {
					case FormaSinus :
						nota[pos].salida =
fAmplitud*sin(2*pi*fase);
						break;
					case FormaQuadra :
						if(fase<0.5) {
							nota[pos].salida = fAmplitud;
```

```
                                        }
                                        else {
                                                nota[pos].salida = -fAmplitud;
                                        }
                                        break;
                                case FormaTriangle :
                                        if(fase <= 0.5) {
                                                // y = m*x + b
                                                // m= DY / DX    DY = 2*Amplitud
DX = 1/2
                                                // x = fase
                                                // b = -Amplitud
                                                float m = 2 * fAmplitud / 0.5f;
                                                nota[pos].salida = m * fase -
fAmplitud;
                                        }
                                        else {
                                                // y = m*x + b
                                                // m= DY / DX    DY = -
2*Amplitud    DX = 1/2
                                                // x = (fase-0.5)   per a dur-ho
a x=0
                                                // b = Amplitud

                                                float m = -2 * fAmplitud / 0.5f;
                                                nota[pos].salida = m * (fase -
0.5f) + fAmplitud;

                                        }
                                        break;
                                }  //fi switch (fForma)

                                switch (nota[pos].estat) {
                                case EAttack:
                                        PAttack = float(nota[pos].CAttack) /
TAttack;
                                        nota[pos].salida = nota[pos].salida *
PAttack;
                                        nota[pos].CAttack++;
                                        if (nota[pos].CAttack >= TAttack) {
                                                nota[pos].estat = EDecay;
                                        }
                                        break;
                                case EDecay:
                                        //                DY         DX
x                       b
                                        PDecay = ((PSustain-1.0f) / TDecay) *
float(nota[pos].CDecay) + 1.0f;
                                        nota[pos].salida = nota[pos].salida *
PDecay;
                                        nota[pos].CDecay++;
                                        if (nota[pos].CDecay >= TDecay) {
                                                nota[pos].estat = ESustain;
                                        }
                                        break;
                                case ESustain:
                                        nota[pos].salida =
nota[pos].salida*PSustain;
                                        break;
                                case ERelease:
                                        PRelease = (-PSustain / TRelease) *
float(nota[pos].CRelease) + PSustain;
                                        nota[pos].salida =
nota[pos].salida*PRelease;
                                        nota[pos].CRelease++;
                                        if (nota[pos].CRelease >= TRelease) {
                                                nota[pos].estat= EOff;
                                                nota[pos].noteIsOn = false;
```

```
                                        }
                                        break;
                                }
                        } // fi if(nota[pos].noteIsOn)
                        else nota[pos].salida = 0.0f;
                } // fi for(int pos=0;pos<85;pos++)

                float exit = 0.0f;

                // Suma de totes les notes que estan actives
                for(pos=0;pos<85;pos++) {
                        if (nota[pos].noteIsOn) {
                                exit = nota[pos].salida*1/TeclesMaximes + exit;

                        }
                }
                *out++ = exit * (volum*2);
                i++;


        } // fi while

}



VstInt32 ASinteti::processEvents (VstEvents* ev) {
        for (VstInt32 i = 0; i < ev->numEvents; i++) {
                if ((ev->events[i])->type != kVstMidiType)
                        continue;

                VstMidiEvent* event = (VstMidiEvent*)ev->events[i];
                char* midiData = event->midiData;
                VstInt32 status = midiData[0] & 0xf0;  // Seleccionem els 4
primers bits
                if (status == 0x90 || status == 0x80) {// Comparem que els 4
primers bits siguen activar o desactivar nota
                        VstInt32 note = midiData[1] & 0x7f;
                        VstInt32 velocity = midiData[2] & 0x7f;
                        if (status == 0x80)
                                noteOff (note);
                        else {
                                if (TeclesActives < TeclesMaximes)
                                        noteOn (note, velocity, event->deltaFrames);
                        }
                }
                else if (status == 0xb0) {
                        if (midiData[1] == 0x7e || midiData[1] == 0x7b)     //
totes les notes off
                                noteOffAll();
                        if (midiData[1] == 0x48) {
                                int midiData2 = midiData[2];
                                setParameter(kRelease,midiData2/127.0f);
                                getParameter(kRelease);
                        }
                        if (midiData[1] == 0x49) {
                                int midiData2 = midiData[2];
                                setParameter(kAttack,midiData2/127.0f);
                                getParameter(kAttack);
                        }

                        if (midiData[1] == 0x4B) {
                                int midiData2 = midiData[2];
                                setParameter(kDecay,midiData2/127.0f);
                                getParameter(kDecay);
                        }
                        if (midiData[1] == 0x4F) {
```

```
                        int midiData2 = midiData[2];
                        setParameter(kSustain,midiData2/127.0f);
                        getParameter(kSustain);
                }
                if (midiData[1] == 0x07) {
                        int midiData2 = midiData[2];
                        volum = midiData2/127.0f;

                }
        }
        event++;
    }
    return 1;
}


void ASinteti::noteOn (VstInt32 note, VstInt32 velocity, VstInt32 delta) {
    int pos;
    pos = conversorNotaenPosicion(note);
    nota[pos].Velocity = velocity;
    nota[pos].Delta = delta;
    nota[pos].noteIsOn = true;
    nota[pos].fPhase1 = nota[pos].fPhase2 = 0;
    nota[pos].CAttack = nota[pos].CDecay = nota[pos].CRelease = 0;
    nota[pos].salida = 0.0f;
    nota[pos].estat = EAttack;
    TeclesActives++;
}


void ASinteti::noteOff (VstInt32 note) {
    int pos;
    pos = pos = conversorNotaenPosicion(note);
    nota[pos].estat = ERelease;
    if(nota[pos].noteIsOn == true)
        TeclesActives--;
}

void ASinteti::noteOffAll() {
    for(int pos=0; pos<85; pos++) {
        nota[pos].noteIsOn = false;
        nota[pos].estat = EOff;
        nota[pos].CAttack = 0;
        nota[pos].CDecay = 0;
        nota[pos].CRelease = 0;
        nota[pos].fPhase1 = 0;
        nota[pos].fPhase2 = 0;
        nota[pos].Delta = nota[pos].Velocity = 0;
        nota[pos].NoteFreq = conversorPosicionenFrecuencia(pos);
        nota[pos].salida = 0.0f;
    }
}
```

6 Bibliografia

- Lindley, Craig A. – Digital Audio with Java – Prentice Hall – 1999 – USA
- Penfold, R.A. – MIDI Avandazado, guía del Usuario – Ra-Ma Editorial – 1992 – España
- Miles Huber, David – The MIDI manual – Focal Press – 1999 – USA

6.1 E-recursos

- Introducción a la adquisición, síntesis y procesamiento del audio - http://www.disca.upv.es/adomenec/IASPA/
- Aprende Gratis - http://www.aprende-gratis.com/produccion/
- Wikipedia - www.wikipedia.org
- Cancionero net - http://www.cancionero.net/escuela/articulo.asp?t= frecuencias_ de_las_ notas_musicales &n=10
- Electrónica y electrónicos - http://electronico.wordpress.com/2008/03/26/frecuencia-de-muestreo/
- Duiops.net - http://www.duiops.net/newage/newsinte.htm
- Aula Actual - http://www.aulaactual.com/especiales/sintetizador/sintetizador.htm
- Aula Actual - http://www.aulaactual.com/equipamiento/hsecuenciadores.php
- Liberation frequency - http://www.liberationfrequency.at/blog/wp-content/uploads/2006/11/introduction_to_vst_plug-in_implementation.pdf
- Hp music - http://www.heikoplate.de/mambo/index.php?option=com_content&task=view&id=427&Itemid=63
- Manual Ego System KeyControl 25 http://www.esi-audiotechnik.com/download/ESI/KeyControl_25/KeyControl_25-English.pdf
- MSDN http://msdn.microsoft.com/es-es/vstudio/products/bb931214.aspx