



NATIVE INSTRUMENTS
SOFTWARE SYNTHESIS

REAKTOR 5 CORE

チュートリアル


オペレーションマニュアル



このマニュアルに記載した事項は、予告なしに変更される場合があります。また、内容についてNative Instruments Software Synthesis GmbHが責任を負うと表明するものではありません。このマニュアルに述べるソフトウェアの使用にあたっては、利用許諾契約を結ぶ必要があります。ソフトウェアを他の媒体に複製することはできません。このマニュアルの一部または全部をコピー、転載し、または伝送、録音、録画するためには、目的のいかんを問わず、Native Instruments Software Synthesis GmbHから事前に書面による許可を得る必要があります。このマニュアルに記載した製品名や会社名は、それぞれの所有者の商標または登録商標です。

マニュアルの執筆者: Len Sasso。

© Native Instruments Software Synthesis GmbH, 2005. All rights reserved. REAKTOR、REAKTOR 5、REAKTOR 5 COREはNative Instruments Software Synthesis の商標です。
日本語版: 株式会社メディア著作権所有。

NATIVE INSTRUMENTS Software Synthesis 
info@native-instruments.com www.native-instruments.com

株式会社メディア 

〒166-0015 東京都杉並区成田東5-17-13 ハーモナイズビル 7F
ホームページ: <http://www.midia.co.jp>
サポートメール: m-support@midia.co.jp (Mac環境)
w-support@midia.co.jp (Windows環境)
サポート: TEL: 03-5335-5862 FAX: 03-5335-5865

目次

1. Reaktor Core: 始めの一步	1
1.1. Reaktor Core とは？	1
1.2. コア・セルの使い方	2
1.3. 実際のコア・セルの使用例	5
1.4. コア・セルの編集	8
2. Reaktor Core の詳細	15
2.1. コア・セルの種類：イベント型とオーディオ型	15
2.2. 基本的なコア・セルの作成手順	16
2.3. オーディオ信号と制御信号	30
2.4. 簡単な Reaktor Core マクロの構築	38
2.5. オーディオ信号を制御信号として使うこと	46
2.6. イベント信号	48
2.7. 論理信号	52
3. Reaktor Core の基礎：コア信号モデル	55
3.1. 値	55
3.2. イベント	55
3.3. 同時イベント	58
3.4. 処理順序	60
3.5. イベント型コア・セル再説	61
4. 内部状態を持つストラクチャー	68
4.1. クロック信号	68
4.2. オブジェクト・バス接続 (OBC)	69
4.3. 初期設定	72
4.4. イベント累算器の作成	75
4.5. イベントのマージ	76
4.6. リセット / 初期設定機能付きのイベント累算器	78
4.7. イベント・シェイパーの問題の解決	85
5. コアにおけるオーディオ処理	88
5.1. オーディオ信号	88
5.2. サンプル・レート・クロック用のバス	90
5.3. 信号の帰還経路	91

5.4. マクロが絡む帰還経路.....	95
5.5. 非正規数.....	99
5.6. その他の特別な値.....	103
5.7. 1 ポール・ロー・パス・フィルターの構築.....	104
6. 条件処理.....	108
6.1. イベントの経路制御.....	108
6.2. 信号クリッパーの作成.....	110
6.3. 単純な鋸波発振器の作成.....	111
7. その他の種類の信号.....	114
7.1. 浮動小数点数値 (Float) 信号.....	114
7.2. 整数値 (Integer) 信号.....	116
7.3. イベント・カウンターの作成.....	119
7.4. 立ち上がり信号カウンター・マクロの作成.....	120
8. アレイ (Array)	125
8.1. 概要.....	125
8.2. オーディオ信号セクターの作成.....	128
8.3. ディレイの作成.....	135
8.4. 表.....	142
9. 効率のよいストラクチャーの作成.....	147
9.1. ラッチと変調マクロ.....	147
9.2. 分岐処理とマージ.....	148
9.3. 数値演算.....	149
9.4. 浮動小数点数と整数の変換.....	150
付録 A. ユーザー・インターフェイス	152
A.1. コア・セル.....	152
A.2. コア・モジュール / マクロ.....	152
A.3. コア・ポート.....	153
A.4. コア・ストラクチャーの編集.....	153
付録 B. Reaktor Core で使われる用語.....	155
B.1. 信号、イベント.....	155
B.2. 初期設定.....	155
B.3. オブジェクト・バス接続.....	156
B.4. 信号経路の分岐.....	156

B.5. ラッチ.....	156
B.6. クロック信号.....	156
付録 C. コア・マクロ用のポート	158
C.1. In	158
C.2. Out	158
C.3. Latch (入力).....	158
C.4. Latch (出力).....	158
C.5. Bool C (入力)	158
C.6. Bool C (出力)	159
付録 D. コア・セル用のポート	160
D.1. In (オーディオ・モード)	160
D.2. Out (オーディオ・モード)	160
D.3. In (イベント・モード)	160
D.4. Out (イベント・モード).....	160
付録 E. 組み込みバス	162
E.1. SR.C (Sample-Rate Clock)	162
E.2. SR.R (Sample-Rate Rate)	162
付録 F. 組み込みモジュール	163
F.1. Const.....	163
F.2. Math > +.....	163
F.3. Math > -.....	163
F.4. Math > *	163
F.5. Math > /	164
F.6. Math > x 	164
F.7. Math > -x.....	164
F.8. Math > DN Cancel.....	164
F.9. Math > ~log	165
F.10. Math > ~exp.....	165
F.11. Bit > Bit AND	165
F.12. Bit > Bit OR	165
F.13. Bit > Bit XOR.....	166
F.14. Bit > Bit NOT	166
F.15. Bit > Bit <<	166

F.16. Bit > Bit >>	166
F.17. Flow > Router.....	167
F.18. Flow > Compare.....	167
F.19. Flow > Compare Sign.....	167
F.20. Flow > ES Ctl	168
F.21. Flow > ~BoolCtl.....	168
F.22. Flow > Merge.....	168
F.23. Flow > EvtMerge	169
F.24. Memory > Read.....	169
F.25. Memory > Write	169
F.26. Memory > R/W Order.....	170
F.27. Memory > Array.....	170
F.28. Memory > Size [].....	171
F.29. Memory > Index	171
F.30. Memory > Table	171
F.31. Macro.....	172
付録 G. 高度なマクロ.....	173
G.1. Clipping > Clip Max / IClip Max.....	173
G.2. Clipping > Clip Min / IClip Min.....	173
G.3. Clipping > Clip MinMax / IClipMinMax	173
G.4. Math > 1 div x.....	173
G.5. Math > 1 wrap.....	174
G.6. Math > lmod	174
G.7. Math > Max / lMax	174
G.8. Math > Min / lMin.....	174
G.9. Math > round.....	174
G.10. Math > sign +/-.....	175
G.11. Math > sqrt (>0)	175
G.12. Math > sqrt	175
G.13. Math > x(>0)^y	175
G.14. Math > x^2 / x^3 / x^4.....	175
G.15. Math > Chain Add / Chain Mult.....	176
G.16. Math > Trig-Hyp > 2 pi wrap.....	176
G.17. Math > Trig-Hyp > arcsin / arccos / arctan	176

G.18. Math > Trig-Hyp > sin / cos / tan	176
G.19. Math > Trig-Hyp > sin -pi..pi / cos -pi..pi / tan -pi..pi	176
G.20. Math > Trig-Hyp > tan -pi4..pi4.....	177
G.21. Math > Trig-Hyp > sinh / cosh / tanh	177
G.22. Memory > Latch / lLatch	177
G.23. Memory > z ⁻¹ / z ⁻¹ ndc	177
G.24. Memory > Read []	178
G.25. Memory > Write []	178
G.26. Modulation > x + a / Integer > lx + a.....	178
G.27. Modulation > x * a / Integer > lx * a.....	179
G.28. Modulation > x - a / Integer > lx - a.....	179
G.29. Modulation > a - x / Integer > la - x.....	179
G.30. Modulation > x / a.....	179
G.31. Modulation > a / x.....	180
G.32. Modulation > xa + y.....	180
付録 H. 標準マクロ.....	181
H.1. Audio Mix-Amp > Amount.....	181
H.2. Audio Mix-Amp > Amp Mod.....	181
H.3. Audio Mix-Amp > Audio Mix.....	181
H.4. Audio Mix-Amp > Audio Relay.....	182
H.5. Audio Mix-Amp > Chain (+/- amount)	182
H.6. Audio Mix-Amp > Chain (dB)	182
H.7. Audio Mix-Amp > Gain (dB)	183
H.8. Audio Mix-Amp > Invert	183
H.9. Audio Mix-Amp > Mixer 2/3/4.....	183
H.10. Audio Mix-Amp > Pan.....	183
H.11. Audio Mix-Amp > Ring-Amp Mod	184
H.12. Audio Mix-Amp > Stereo Amp.....	184
H.13. Audio Mix-Amp > Stereo Mixer 2/3/4	185
H.14. Audio Mix-Amp > VCA.....	185
H.15. Audio Mix-Amp > XFade (lin)	185
H.16. Audio Mix-Amp > XFade (par)	186
H.17. Audio Shaper > 1+2+3 Shaper	186
H.18. Audio Shaper > 3-1-2 Shaper.....	186

H.19. Audio Shaper > Broken Par Sat.....	187
H.20. Audio Shaper > Hyperbol Sat.....	187
H.21. Audio Shaper > Parabol Sat.....	187
H.22. Audio Shaper > Sine Shaper 4/8	188
H.23. Control > Ctl Amount	188
H.24. Control > Ctl Amp Mod	188
H.25. Control > Ctl Bi2Uni	189
H.26. Control > Ctl Chain.....	189
H.27. Control > Ctl Invert.....	189
H.28. Control > Ctl Mix	189
H.29. Control > Ctl Mixer 2.....	190
H.30. Control > Ctl Pan.....	190
H.31. Control > Ctl Relay	190
H.32. Control > Ctl XFade.....	191
H.33. Control > Par Ctl Shaper.....	191
H.34. Convert > dB2AF.....	191
H.35. Convert > dP2FF	192
H.36. Convert > logT2sec.....	192
H.37. Convert > ms2Hz	192
H.38. Convert > ms2sec.....	192
H.39. Convert > P2F.....	192
H.40. Convert > sec2Hz	192
H.41. Delay > 2/4 Tap Delay 4p.....	193
H.42. Delay > Delay 1p/2p/4p.....	193
H.43. Delay > Diff Delay 1p/2p/4p	193
H.44. Envelope > ADSR	194
H.45. Envelope > Env Follower	195
H.46. Envelope > Peak Detector.....	195
H.47. EQ > 6dB LP/HP EQ.....	195
H.48. EQ > 6dB LowShelf EQ	196
H.49. EQ > 6dB HighShelf EQ	196
H.50. EQ > Peak EQ.....	196
H.51. EQ > Static Filter > 1-pole static HP.....	197
H.52. EQ > Static Filter > 1-pole static HS.....	197

H.53. EQ > Static Filter > 1-pole static LP	197
H.54. EQ > Static Filter > 1-pole static LS	197
H.55. EQ > Static Filter > 2-pole static AP	198
H.56. EQ > Static Filter > 2-pole static BP	198
H.57. EQ > Static Filter > 2-pole static BP1	198
H.58. EQ > Static Filter > 2-pole static HP	198
H.59. EQ > Static Filter > 2-pole static HS	199
H.60. EQ > Static Filter > 2-pole static LP	199
H.61. EQ > Static Filter > 2-pole static LS	199
H.62. EQ > Static Filter > 2-pole static N	200
H.63. EQ > Static Filter > 2-pole static Pk	200
H.64. EQ > Static Filter > Integrator	200
H.65. Event Processing > Accumulator	200
H.66. Event Processing > Clk Div	201
H.67. Event Processing > Clk Gen	201
H.68. Event Processing > Clk Rate	201
H.69. Event Processing > Counter	201
H.70. Event Processing > Ctl2Gate	202
H.71. Event Processing > Dup Flt / IDup Flt	202
H.72. Event Processing > Impulse	202
H.73. Event Processing > Random	202
H.74. Event Processing > Separator / ISeparator	203
H.75. Event Processing > Thld Crossing	203
H.76. Event Processing > Value / IValue	203
H.77. LFO > MultiWave LFO	204
H.78. LFO > Par LFO	204
H.79. LFO > Random LFO	204
H.80. LFO > Rect LFO	205
H.81. LFO > Saw(down) LFO	205
H.82. LFO > Saw(up) LFO	205
H.83. LFO > Sine LFO	205
H.84. LFO > Tri LFO	206
H.85. Logic > AND	206
H.86. Logic > Flip Flop	206

H.87. Logic > Gate2L	206
H.88. Logic > GT / IGT	206
H.89. Logic > EQ	207
H.90. Logic > GE	207
H.91. Logic > L2Clock	207
H.92. Logic > L2Gate	207
H.93. Logic > NOT	208
H.94. Logic > OR	208
H.95. Logic > XOR	208
H.96. Logic > Schmitt Trigger	208
H.97. Oscillators > 4-Wave Mst	209
H.98. Oscillators > 4-Wave Slv	209
H.99. Oscillators > Binary Noise	209
H.100. Oscillators > Digital Noise	210
H.101. Oscillators > FM Op	210
H.102. Oscillators > Formant Osc	210
H.103. Oscillators > MultiWave Osc	211
H.104. Oscillators > Par Osc	211
H.105. Oscillators > Quad Osc	211
H.106. Oscillators > Sin Osc	211
H.107. Oscillators > Sub Osc 4	212
H.108. VCF > 2 Pole SV	212
H.109. VCF > 2 Pole SV C	212
H.110. VCF > 2 Pole SV (x3) S	213
H.111. VCF > 2 Pole SV T (S)	213
H.112. VCF > Diode Ladder	214
H.113. VCF > D/T Ladder	214
H.114. VCF > Ladder x3	214
付録I. コア・セル・ライブラリー	216
I.1. Audio Shaper > 3-1-2 Shaper	216
I.2. Audio Shaper > Broken Par Sat	216
I.3. Audio Shaper > Hyperbol Sat	216
I.4. Audio Shaper > Parabol Sat	217
I.5. Audio Shaper > Sine Shaper 4/8	217

I.6. Control > ADSR.....	217
I.7. Control > Env Follower.....	218
I.8. Control > Flip Flop	219
I.9. Control > MultiWave LFO.....	219
I.10. Control > Par Ctl Shaper	219
I.11. Control > Schmitt Trigger	220
I.12. Control > Sine LFO	220
I.13. Delay > 2/4 Tap Delay 4p	220
I.14. Delay > Delay 4p.....	221
I.15. Delay > Diff Delay 4p	221
I.16. EQ > 6dB LP/HP EQ.....	221
I.17. EQ > HighShelf EQ.....	222
I.18. EQ > LowShelf EQ	222
I.19. EQ > Peak EQ.....	222
I.20. EQ > Static Filter > 1-pole static HP.....	223
I.21. EQ > Static Filter > 1-pole static HS.....	223
I.22. EQ > Static Filter > 1-pole static LP.....	223
I.23. EQ > Static Filter > 1-pole static LS.....	223
I.24. EQ > Static Filter > 2-pole static AP.....	224
I.25. EQ > Static Filter > 2-pole static BP	224
I.26. EQ > Static Filter > 2-pole static BP1	224
I.27. EQ > Static Filter > 2-pole static HP.....	225
I.28. EQ > Static Filter > 2-pole static HS.....	225
I.29. EQ > Static Filter > 2-pole static LP.....	225
I.30. EQ > Static Filter > 2-pole static LS.....	226
I.31. EQ > Static Filter > 2-pole static N.....	226
I.32. EQ > Static Filter > 2-pole static Pk.....	226
I.33. Oscillator > 4-Wave Mst.....	227
I.34. Oscillator > 4-Wave Slv	227
I.35. Oscillator > Digital Noise	228
I.36. Oscillator > FM Op	228
I.37. Oscillator > Formant Osc	228
I.38. Oscillator > Impulse	228
I.39. Oscillator > MultiWave Osc	229

I.40. Oscillator > Quad Osc.....	229
I.41. Oscillator > Sub Osc.....	229
I.42. VCF > 2 Pole SV C.....	230
I.43. VCF > 2 Pole SV T.....	230
I.44. VCF > 2 Pole SV x3 S.....	231
I.45. VCF > Diode Ladder.....	231
I.46. VCF > D/T Ladder.....	232
I.47. VCF > Ladder x3.....	232

1. Reaktor Core: 始めの一步

1.1. Reaktor Core とは？

Reaktor では、基本的な機能を提供する部品を組み合わせ、より高度な機能を実現できるようになっています。Reaktor Core では、その中でも中核を成す機能（コア機能）を集め、基本レベルとコア・レベルという階層に整理しました。また、「基本レベルのストラクチャー」とは、どのような部品をどう組み合わせ、インストゥルメントやマクロを実現しているか、その構成を表す言葉です。アンサンブルはさらに上位階層に位置するものなので、基本レベルとしては扱いません。

Reaktor Core の機能は、基本レベルの機能と直接対応していないので、それぞれを対応づけるために「コア・セル」というものを使います。これを基本レベルのストラクチャー内に置き、組み込みモジュールと同じように扱います。次に示すのは「HighShelf EQ」というコア・セルを使ったストラクチャーの例です。基本レベルの組み込みモジュールにも似た機能のものがありませんが、カットオフ周波数や増幅率を調整できるようになっている点が異なります。



コア・セルの中身は、Reaktor Core が提供する中核機能部品を組み合わせ、実装されています。部品を組み合わせる方式なので、独自の低レベル DSP 機能、あるいは複雑な信号処理機能も容易に実装できます。実際の組み合わせ方についてはこのあと詳しく解説していきます。

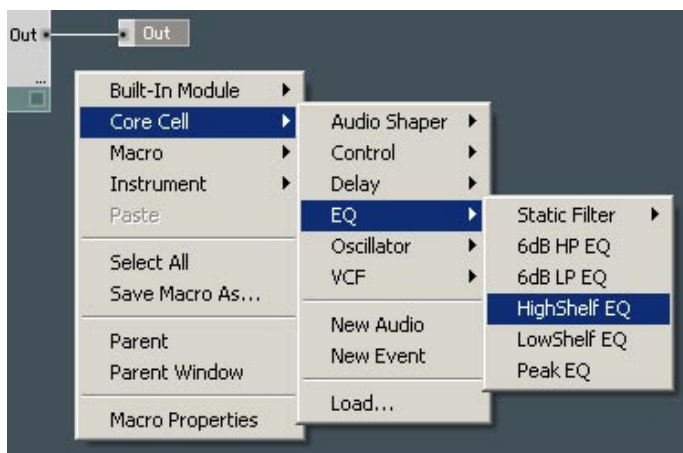
Reaktor Core を使えば、信号そのものを処理するような低レベル DSP ストラクチャーを容易に構築できますが、ほかにも用途はさまざまです。DSP プログラミングの経験がなくても、あらかじめ作成済みのモジュールを揃えたライブラリーがあるので、コア・レベルのストラクチャー内に置いて自由に組み合わせ、使うことが可能です。この考え方は、基本レベルのストラクチャー内でモジュールやマクロを組み合わせるのと同じです。また、コア・セルについても作成済みのものがライブ

ラリーとして提供されており、基本レベルのストラクチャー内で自由に使えます。

今後 Native Instruments は、基本レベルのモジュール群を充実させていく代わりに、Reaktor Core の技術を活用してさまざまなコア・セルを提供していく計画です。ライブラリーには既に、フィルター、エンベロープ生成器、エフェクトなどが多数揃っています。

1.2. コア・セルの使い方

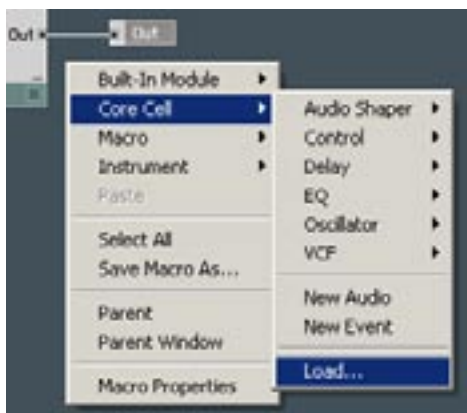
ライブラリーとして提供されているコア・セルは、基本レベルのストラクチャー作成画面で呼び出せます。背景部分を右クリックし、**Core Cell** 以下のサブメニューから選択してください。



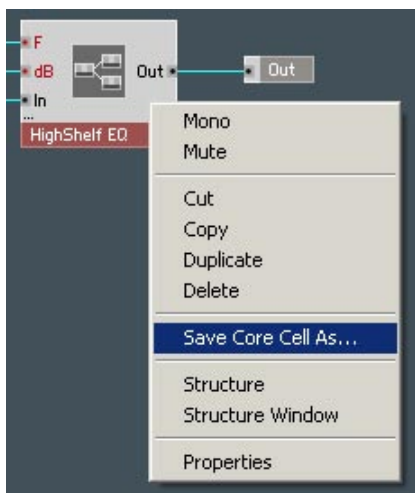
この図からも分かるように、コア・セルにはさまざまな種類があり、組み込みモジュールと同じように使えます。

コア・セルには、イベント・ループ内では使えない、という重要な制限があります。イベント・ループ内に置こうとしてもその時点で検出され、配置できません。

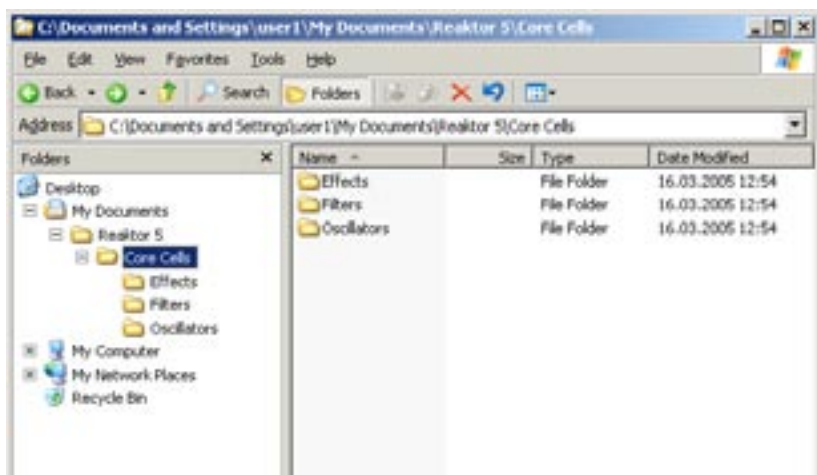
また、ライブラリー以外のコア・セルも挿入できます。**Core Cell > Load...** コマンドを実行してください。



作成 / 修正したコア・セルを保存しておき、別のストラクチャーで流用することも可能です。コア・セルを右クリックすると現れるメニューから、**Save Core Cell As...** コマンドを実行してください。

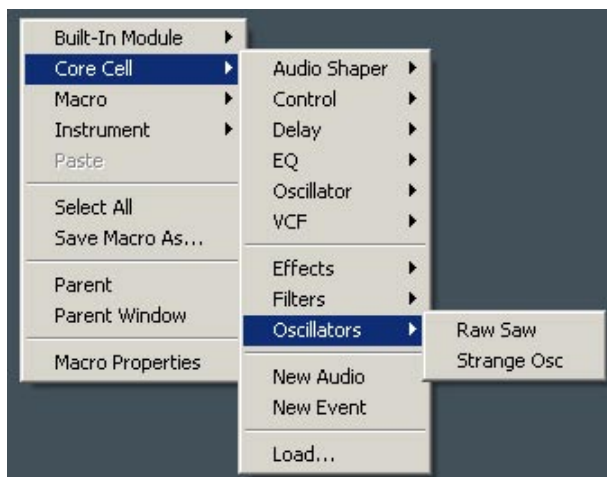


Load... コマンドを使わなくても、ユーザー・ライブラリー・フォルダー以下の「Core Cells」というサブフォルダーにコア・セルを置いておけば、メニューから呼び出して使えます。さらに、次の例のように、機能に応じて分類しておくといでしょう。



この例では「My Documents\Reaktor 5」が Reaktor のユーザー・ライブラリー・フォルダーになっています。実際のパスは Reaktor インストール時の設定によって異なり、環境設定として変更することも可能です。「Core Cells」サブフォルダーはこのすぐ下にあるはずですが、ない場合は手動で作成してください。

ここにコア・セルのファイルを置くのですが、例ではさらに、「Effects」、「Filters」、「Oscillators」というサブフォルダーに整理しています。このフォルダー階層が、**Core Cell** メニューの構成にも反映されます。



メニュー構成は Reaktor の起動時に決まります。ファイルを追加するなどしてフォルダー階層を変えた場合、メニューに反映させるためには Reaktor を起動し直す必要があります。

サブフォルダーがあってもその中身が空であればメニューには表示されません。

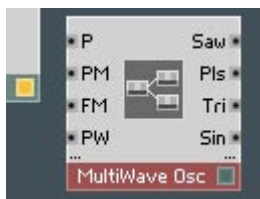
自分で作成したファイルを、システム・ライブラリーには決して置かないでください。アップデートの際に中身を総入れ替えることがあり、独自に作成したファイルは消えてしまう恐れがあります。製品に付属している以外のファイルは、ユーザー・ライブラリーに置いてください。

1.3. 実際のコア・セルの使用例

ここでは例として、基本レベルのモジュールだけを使った Reaktor インストゥルメントに、コア・セルを追加して機能を強化してみましょう。インストール先の「Core Tutorial Examples」フォルダーにある、「One Osc.ens」というアンサンブル・ファイルを開いてください。ここには次のような内部ストラクチャーのインストゥルメントがひとつ入っています。



この図からも分かるように、非常に簡単な減算型シンセサイザーで、発振器 (Sawtooth)、フィルター (2-Pole Filter)、エンベロープ生成器 (ADSR-Env) がひとつずつあります。ここで発振器を、別のもっと機能豊富なものに置き換えてみましょう。画面の背景部分を右クリックし、**Core Cell > Oscillator > MultiWave Osc** を選択してください。



この発振器 (MultiWave Osc) の特徴は、位相が揃ったいくつものアナログ波形を同時に発振できることです。元の発振器は鋸波しか出力できなかったのに対し、いくつかの基本波形を混ぜ合わせた信号を出力できます。信号を混ぜ合わせるためのミキサー・マクロはあらかじめ用意されているので、**Macro > Classic Modular > 02 - Mixer Amp > Mixer - Simple - Mono** コマンドで追加してください。



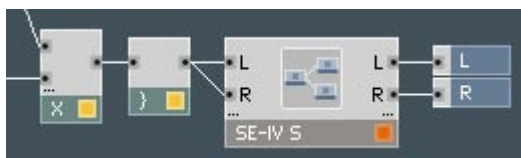
ミキサーと発振器を次の図のように結線し、鋸波発振器の代わりに置きます。



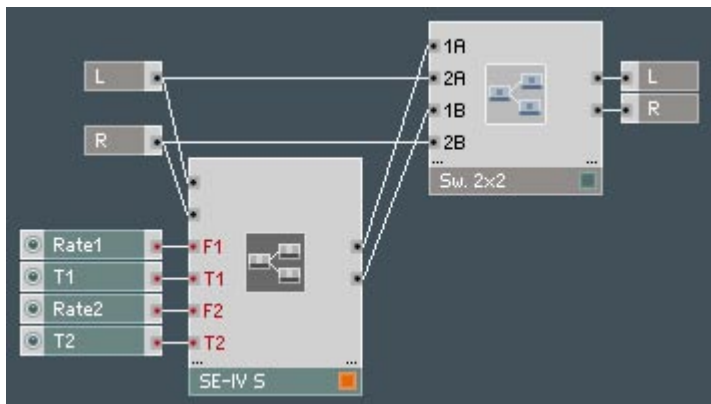
パネル表示に切り替えると、ミキサーに付属している4つのフェーダーで、それぞれの波形を混ぜ合わせる比率を調整できます。

次に、Reaktor Core ベースのコラス・エフェクトを追加してみましょう。「Reaktor Core ベース」というのは、コーラス機能そのものはコア・セルとして実装されていますが、制御パネルは基本レベルの機能で構築されているからです。このような構成になっているのは、現行の Reaktor Core ではストラクチャーに独自の制御パネルを組み込むことができないからです。

Macro > Building Blocks > Effects > SE-IV Chorus コマンドで、次のように Voice Combiner モジュールの後段に追加してください。



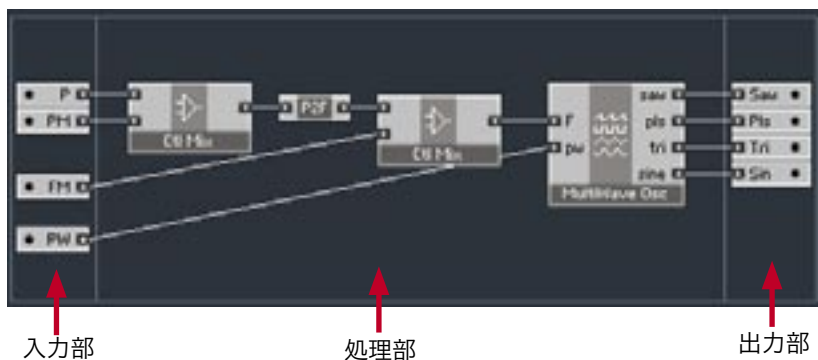
コーラス・エフェクトの中身を見ると、次のように、コア・セルと制御パネルに分かれています。



1.4. コア・セルの編集

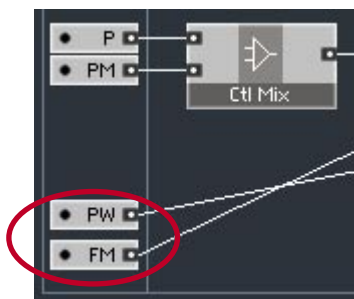
次にコア・セルの編集方法を簡単に紹介します。最初は既存のコア・セルを修正してみましょう。

コア・セル **MultiWave Osc** をダブル・クリックすると、その中身が次のように表示されます。

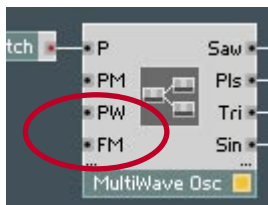


これが Reaktor Core のコア・レベルのストラクチャーです。縦線で、左から順に入力部、処理部、出力部に分かれています。

処理部にあるモジュールはマウスでつかんで自由に位置を動かすことができますが、入力部と出力部にある「ポート」は、上下に動かして並び順を変えることしかできません。試しに「FM」という入力を「PM」の下に移動すると、次のようになります。



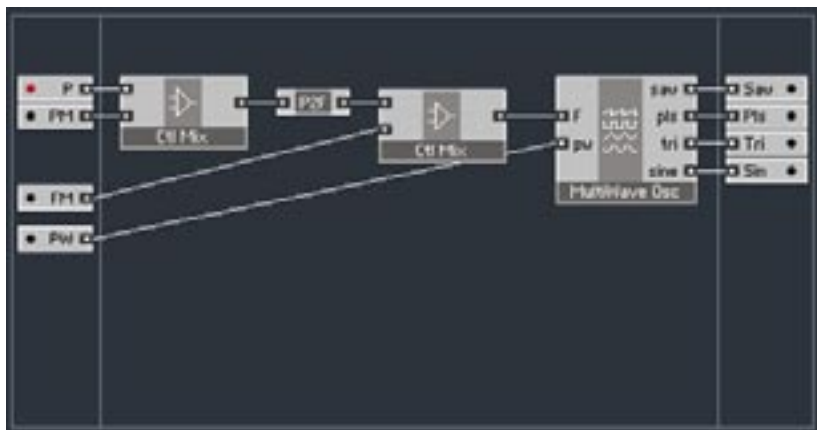
このコア・セルを外側から見たときのポートの並び順も変わっているはずです。実際、背景部分をダブル・クリックして基本レベルのストラクチャーに戻ってみると、順序が入れ替わっていることが分かります。



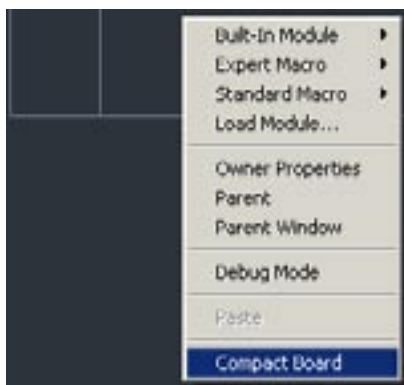
これを確認したら再びコア・レベルに切り替え、並び順を元に戻しておいてください。



もう気づいているかも知れませんが、モジュールを動かした結果場所が足りなくなると、自動的に表示領域が広がります。しかし必要なくなっても自動的に縮むことはないので、無駄に広い領域になってしまうかも知れません。



その場合は背景部分を右クリックし、**Compact Board** コマンドを実行してください。

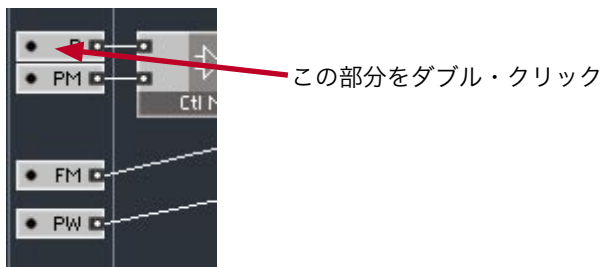



以上、モジュールの位置を動かし、コア・セルのポートの順序を変える方法を解説しました。もう少し機能を紹介しましょう。

コア・セルにオーディオ出力がある場合、入力信号の種類は、オーディオ信号かイベント信号のいずれかに切り替えることができます（この違いについて詳しくは後述）。先に出てきた **MultiWave Osc** モジュールの場合、入力、出力ともすべてオーディオ信号になっていますが、つながっているのはピッチ調整用のノブだけなので、実際にはオーディオ信号でなくても構いません。一部のポートだけでもイベント信号に切り替

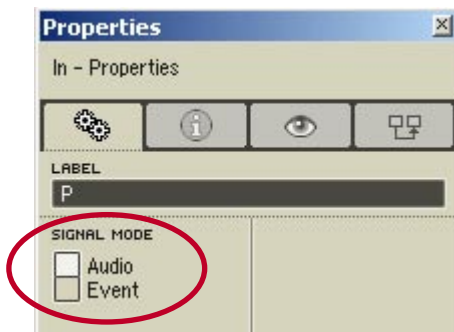
えると、CPU に対する負荷が軽減されます。その手順は次のようになります。

ここでは P と PM の 2 つの入力をイベント信号に切り替えてみましょう。P ポート・モジュールをダブル・クリックして、プロパティ・ウィンドウを開きます。



Properties ウィンドウが開いたら、必要に応じ、タブ  を

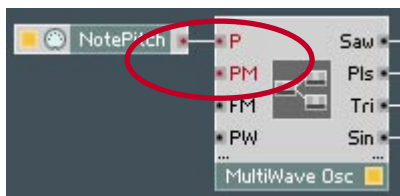
押して画面を切り替えてください。ここに **SIGNAL MODE** という設定項目があります。



これを **Event** に切り替えると、入力モジュールの左側にある丸印が黒から赤に変わるので、イベント信号を受け取るモードになったことが確認できます。分かりにくければ、ポート以外の箇所をクリックして、選択を解除してみてください。



PM 入力も同様にイベント・モードに切り替えてください。残りの 2 つの入力も切り替えて構いません。最後に背景部分をダブル・クリックして基本レベルに戻ってみると、ポートの色が赤に変わっており、CPU に対する負荷が削減されています。



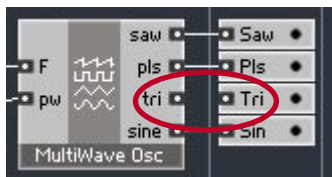
もちろん、モードを切り替えてはならない場合もあります。例えば、エンベロープのようなオーディオ・レートの制御信号ではなく、本当にオーディオ信号を受け取るポートの場合、イベント・モードに切り替えると処理速度が追いつかず、モジュールの機能に支障をきたします。逆にイベント信号を受け取るポートの場合も、オーディオ・モードにすると信号レベルの変化に追いつけず、正常に動作しなくなることがあります。例えばエンベロープ生成器のイベント・トリガー入力 (基本レベルのエンベロープというゲート入力などに相当) がこれに当たります。

このように、適切なモードが明らかに決まっているような場合に加え、問題なさそうに見えるけれども、実際に切り替えると正常に動作しなくなることがまれにあります。その多くは、実装上の制約や不適切なモジュール設計によるものです。以上を除けばモードを切り替えても動作そのものに支障をきたすことはないので、次のような方針で切り替えるとよいでしょう。

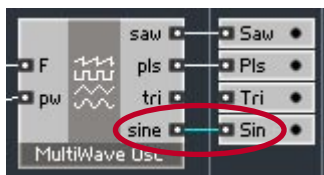
適切に設計されたコア・セルの場合、制御信号が入力されるポートは、それがオーディオ・レートであってもイベント・モードに切り替えて構いません。イベント信号が入力されるポートは、トリガー機能 (その他イベントに依存する機能) がない場合に限り、オーディオ・モードに切り替えても問題ありません。

CPU に対する負荷を減らす手段としては、必要のない出力を切り離し、Reaktor Core ストラクチャーの不要部分を無効にする、という方法もあります。但し、ストラクチャーの内部で切り離すことが大切です。外側で接続を切り離しても効果はありません。

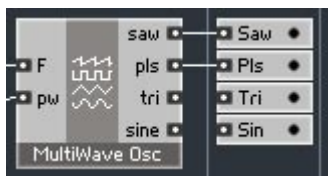
上記の例で、鋸波とパルス波の出力しか必要ないとしましょう。この場合、MultiWave Osc のコア・ストラクチャー上で、必要のない出力の接続を切り離せば、CPU に対する負荷を軽減できます。その手順も簡単で、該当する接続の入力ポートをマウスでつかみ、背景の何もない領域までドラッグして放すだけです。三角波出力 (tri) を受ける入力ポート (Tri) をつかんでドラッグし、何もない領域で放すと次のようになります。



切断の手順はもうひとつあります。MultiWave Osc の正弦波出力 (sine) とコア・セルの出力 (Sin) を結ぶ結線上をマウスで選択してください。次のように、結線の色が青になります。

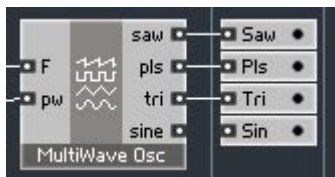


この状態で **delete** キーを押すと結線が消えます。



このようにして不要な結線を 2 つ切断すると、CPU に対する負荷がかなり少なくなります。

あとで切断した出力がやはり必要だということになった場合は、一方のポート上をクリックしてそのままドラッグし、もう一方のポート上で放せば再び接続されます。例えば MultiWave Osc の三角波出力 (tri) 上をクリックし、ドラッグして入力ポート (Tri) 上で放すと、次のように接続されるのです。



もちろんコア・セルは、さまざまな微調整ができるようになっています。どのような調整ができるかは、このマニュアルで徐々に説明していきます。

2. Reaktor Core の詳細

2.1. コア・セルの種類：イベント型とオーディオ型

コア・セルにはイベント型とオーディオ型の2種類があります。これは基本レベル・ストラクチャー上での振る舞いの違いとなって現れます。イベント型コア・セルはイベント信号のみを入力として受け取り、イベント信号のみを出力します。一方オーディオ型コア・セルは、イベント信号、オーディオ信号のどちらも入力として受け取り、オーディオ信号のみを出力します。

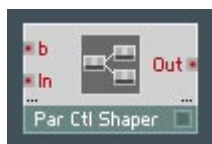
コア・セルの型	入力	出力	クロック源
イベント型	イベント信号	イベント信号	無効
オーディオ型	イベント信号 / オーディオ信号	オーディオ信号	有効

したがって、発振器、フィルター、エンベロープ生成器、エフェクトなどを実装するためにはオーディオ型コア・セルを使い、イベント型はイベント処理のみを行うセルの場合に使います。

HighShelf EQ、**MultiWave Osc** といったモジュールは、オーディオ型コア・セルを使う典型的な例です。オーディオ出力が備わっていることからオーディオ型であることは明らかでしょう。

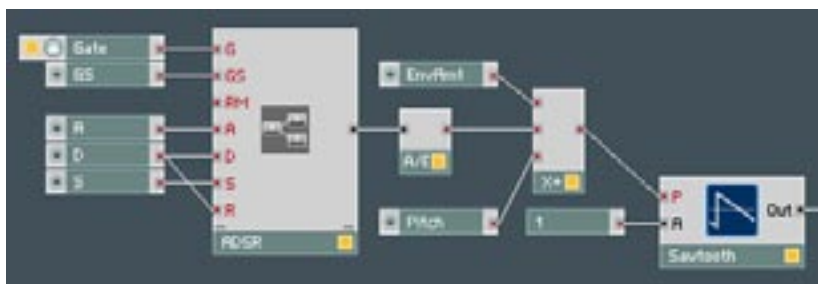


一方、イベント型コア・セルを使った例としては、次のようなものがあります。



この **Par Ctl Shaper** は制御信号を整形する放物型シェイパー (Parabolic Shaper) で、ペロシティー曲線や LFO 信号の波形を対象とします。

先にも説明したように、イベント型コア・セルはイベント処理しかできません。さらに、セル内ではクロック源も使えないので、独自のイベントを生成することもできません。したがってイベント・レートの LFO やエンベロープ生成器は実装できないのです。こういったモジュールはオーディオ型コア・セルで実装し、基本レベルのオーディオ→イベント変換器を介して使うことになります。



上に示したストラクチャーでは、オーディオ型コア・セルとして実装した ADSR エンベロープ生成器の出力をイベント・レートに変換し、この信号を使って別の発振信号を変調しています。

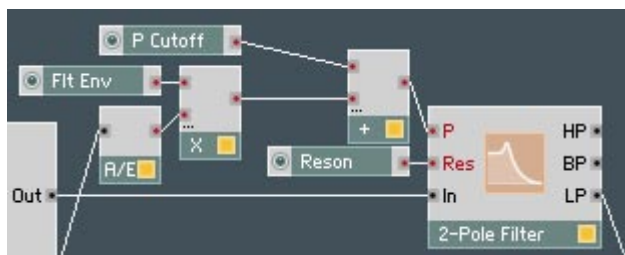
2.2. 基本的なコア・セルの作成手順

基本レベル・ストラクチャー画面の背景部分で右クリックし、作成したいコア・セルの型に応じ、**Core Cell > New Audio** コマンドまたは **Core Cell > New Event** コマンドで作成します。



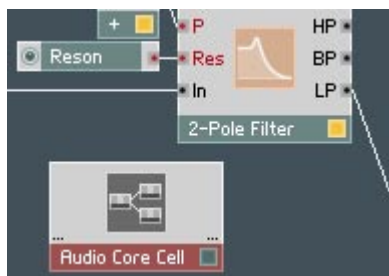
ここでは先に取り上げた「One Osc.ens」アンサンブル内に、既存のフィルター・モジュールを真似て新しいコア・セルを作成し、置き換えてみましょう。前章で発振器とコーラス・エフェクトを追加しましたが、その結果を保存していなくても、以下の手順を試してみるのに問題はありません。

次の図からも分かるように、フィルターのカットオフ周波数を変化させるためにP入力を使いますが、これはイベント信号しか受け取れません。なお、これとは別にFM入力を備えたフィルターもあるのですが、カットオフ周波数を高くしたとき思ったように動作しないため、ここではP入力で変調するフィルターを使っています。また、FM入力の場合、変調信号に対して線形に特性が変化しますが、これはエンベロープ曲線を変調源とした場合、音楽的によい結果が得られません（これを一般に「反応の遅いエンベロープ (slow envelopes)」現象と言いますが、適切な呼称とはいえません）。

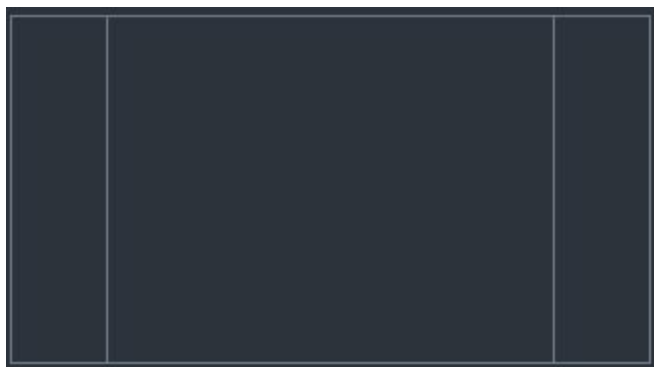


変調信号をイベント入力の形で与える必要があるため、エンベロープ信号を A/E 変換器でイベント信号に変換しています。この変換により、信号の周波数はかなり低くなります。もちろん (CPU に対する負荷を考慮に入れなければ) 高い周波数のまま出力する変換器でも構わないのですが、このあとコア・セルでフィルター機能を実装して差し替える予定なので、周波数を落とすようになっています。実際にはできあいのコア・セルがライブラリーにあるのですが、ここでは練習のため、何もないところから作り上げてみることにします。

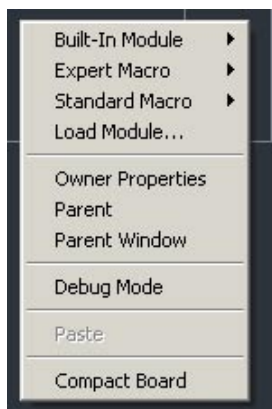
まず、オーディオ型コア・セルを作成します。**Core Cell > New Audio** コマンドを実行すると、次のように、中身の無い状態のオーディオ型コア・セルができます。



これをダブル・クリックすると (コア・レベルの) ストラクチャー表示になりますが、作成したばかりなので何もありません。次のように、入力部、処理部、出力部の各領域に分かれています。



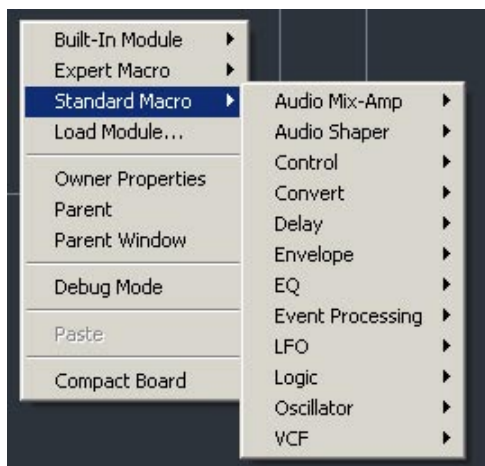
次に、コア・ストラクチャーにモジュールを追加します。処理部の領域内を右クリックすると、次のようなモジュール作成メニューが現れます。



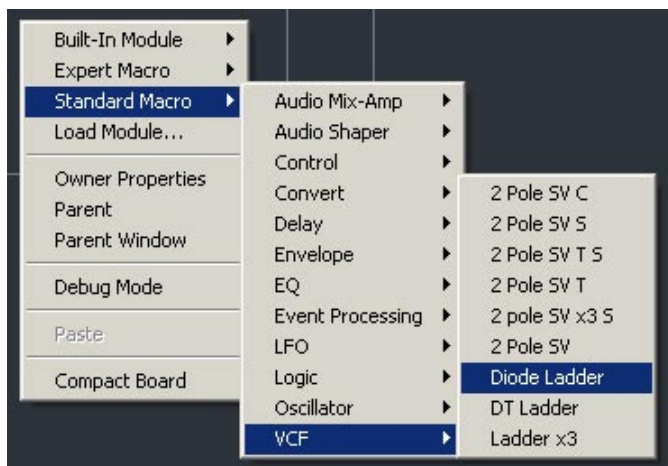
最上段の **Built-In Module** サブメニュー以下には Reaktor Core の組み込みモジュールが並んでいます。これはほとんどが、他のモジュールを構築するための基礎となる、低レベルのモジュールです (詳しくは後述)。

2 番目の **Expert Macro** サブメニューには、組み込みモジュールと組にして使う低レベルのマクロが集まっています。

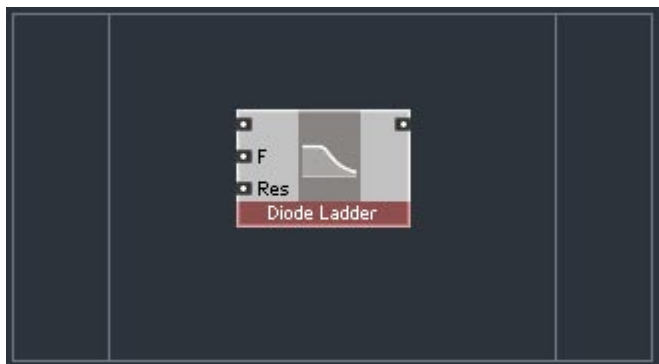
ここでは 3 番目にある **Standard Macro** サブメニューから、必要なマクロを選択することにします。



サブメニューを開き、さらに **VCF** 以下を開くと次のようになっています。



ここから試しに **Diode Ladder** を選択してみましょう。

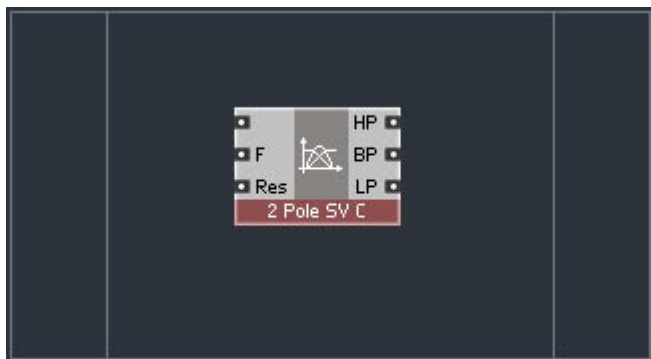


しかしダイオード・ラダーは、今回目標とするフィルターとはかなり違った構成なので、あまり適切とはいえません。何と云っても、ダイオード・ラダーは4ポール(24dB/octave)のフィルターですが、作りたいのは2ポール(12dB/octave)のフィルターなのです。したがってこれを削除することにしますが、その手順は2通りあります。ひとつはモジュールを右クリックし、**Delete Module** コマンドを実行する方法です。



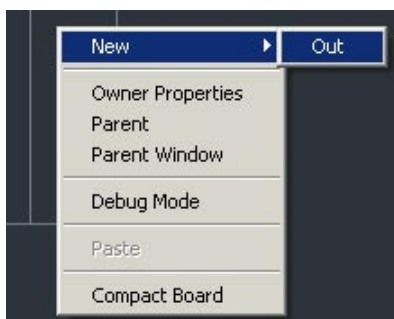
また、モジュールをクリックして選択し、**delete** キーを押すという方法もあります。

削除したら、今度は同じ **Standard Macro** サブメニューの **VCF** 以下にある、**2 Pole SV C** を選択してください。

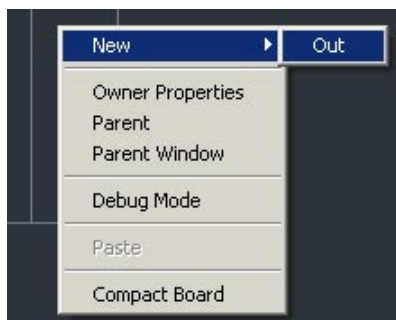


これは 2 ポールの状態可変フィルターで、目標としているフィルター・モジュールによく似ています (まったく同じというわけではありませんが、違いはごくわずかです)。重要なのはオーディオ・レートで変調できる点です。

当然ながらコア・セルには入出力ポートがいくつか必要です。今回必要なのは LP 信号用の出力ポートがひとつだけです。出力部の領域内を右クリックしてください。



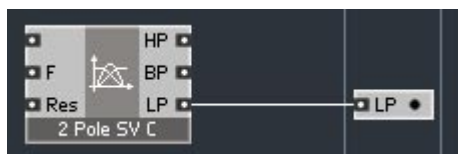
New 以下には作成できるモジュールの種類として **Out** の 1 種類しかありませんが、これを選択します。するとストラクチャーは次のようになります。



出力モジュール上をダブル・クリックして **Properties** ウィンドウを開き、**LABEL** 欄に「LP」と入力してください。



その結果、フィルターの LP 出力と出力モジュールが接続されます。

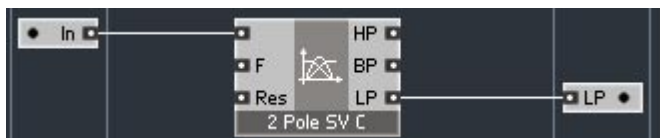


入力側に移って、オーディオ信号の入力ポートを作成します。入力部の背景部分を右クリックし、**New > In** コマンドを実行してください。

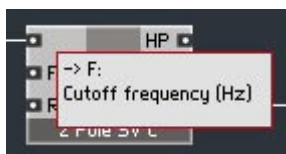


入力は自動的にオーディオ・モードになります。これは黒丸がついていることから分かります。出力モジュール名を「LP」としたのと同様

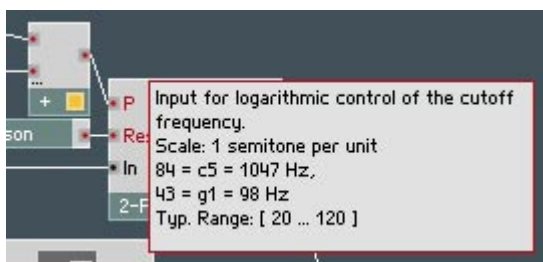
の手順で、「In」という名前に変更してください。するとフィルター・モジュールの一番上の入力とつながります。



2 番目の入力はいくつか複雑です。このモジュールの 2 番目の入力には、図のように「F」という名前がついています。これは Frequency(周波数)の頭文字ですが、実際、このラベル上にマウス・カーソルをしばらく置いたままにしていると、次のように「Cutoff frequency (Hz)」という説明が表示されます。



元のフィルター・モジュールでは、カットオフ周波数は P 入力に与えた信号で制御するようになっていました。同様に説明を表示してみると、次のように、半音単位 (1 semitone per unit) で制御できることが分かります。



これに合わせるためには、半音単位の値を Hz 単位に、すなわち対数目盛を真数目盛に変換する必要があります。これには、基本レベルで **Expon.(F)** モジュールを使う方法と、Reaktor Core ストラクチャー内で処理する方法があります。今はコア・ストラクチャーの構築方法を説明しているところなので、こちらの方法でやってみましょう。処理部の

背景部分を右クリックし、**Standard Macro > Convert > P2F** を選択してください。



名前からも推測できるように、このモジュールには P(Pitch; 音高) を F(Frequency; 周波数) に変換する機能があり、まさに今必要としているものです。そこで、新たに「P」という入力ポートを作り、**P2F** モジュールを介してフィルターに接続してみましょう。



しかしその前に注意があります。元のインストゥルメントには、「P Cutoff」というノブがついています。カットオフ周波数の基準値をこれで指定し、エンベロープから得られる信号でカットオフ周波数を変調する、という構成です。フィルターの P 入力に与える信号は、基本レベル上でイベント型に変換しておく必要がありました。一方、今回作成するフィルターではこの変換が不要なので、A/E モジュールを削除し、オーディオ信号を直接 P 入力に与えることにします。なお、参考のため、別のやり方もあとで紹介します。

P 入力をイベント・モードにし、これとは別にオーディオ・モードの変調入力を追加します。この変調入力ポートを「FM」ではなく「PM」としますが、これは先に説明した「反応の遅いエンベロープ」現象を回避するためです。さらに、元のフィルター・モジュールに合わせて、値を半音単位で設定できるようにします。元のモジュールでは、「P Cutoff」信号にエンベロープ信号を加算して P 入力に供給していました。

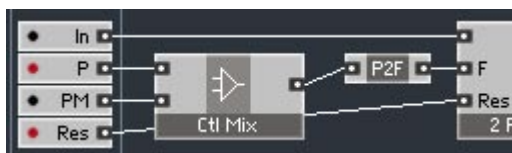
そこで、まず P 入力を (先に紹介した手順で) イベント・モードに変更し、それとは別にオーディオ・モードの PM 入力を追加してみましょう。



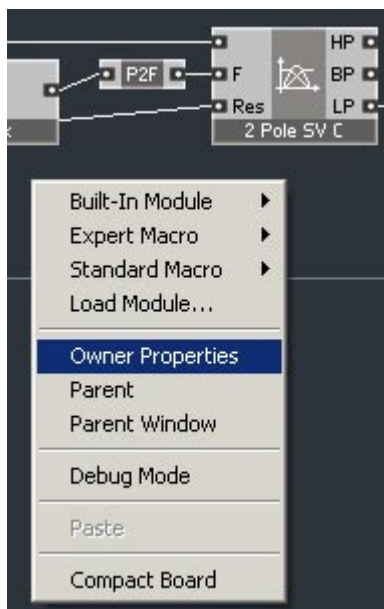
基本レベルの Reaktor を使ったことがあれば、この 2 つの信号を加算するものと予測しているかも知れません。それでもよいのですが、Reaktor Core の場合、**Add** は低レベルのモジュールとして扱うので、低レベルの動作方式をある程度知っておく必要があります。といってもそれ程複雑な話ではないので、あとで概要を解説します。ここではまだ分からなくても構いません。代わりに **Standard Macro > Control > Ctl Mix** の信号ミキサーを使うことにしましょう。



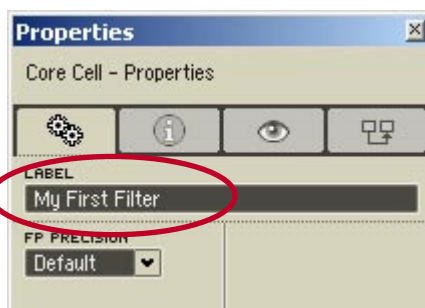
最後に、レゾナンス入力が必要です。これはオーディオ・レートでなくても構わないので、イベント入力モードにしておきます。



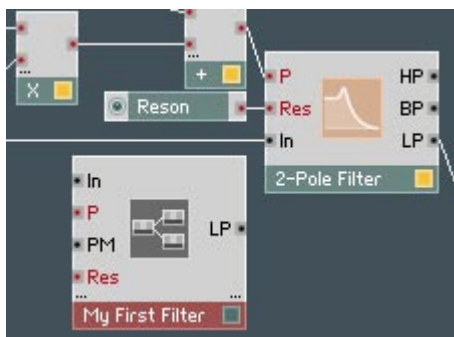
最後に、作成したコア・セルに名前をつけます。**Properties** ウィンドウが開いたままになっていれば、背景部分をクリックするとコア・セル自身のプロパティ設定に切り替わります。ウィンドウが開いていない場合は背景部分を右クリックし、**Owner Properties** コマンドを実行してください。



LABEL 欄に新しい名前を入力します。



背景部分をダブル・クリックすると、その名前が反映されます。



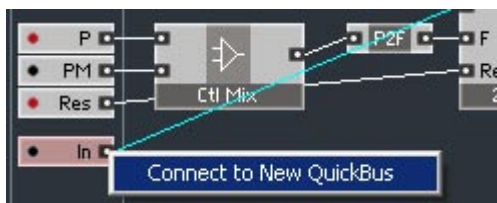
これでほぼ完成しましたが、オーディオ信号入力ポートが元のモジュールでは一番下にあるのに対し、できあがったコア・セルでは一番上になっています。機能上は何も問題ありませんが、修正は簡単ですし、実は既にその手順も説明済みです。実際に修正しながら、ついでに新しい機能も紹介しておきましょう。

コア・レベルの表示画面に戻り、オーディオ信号入力ポートをつかんで一番下までドラッグしてください。

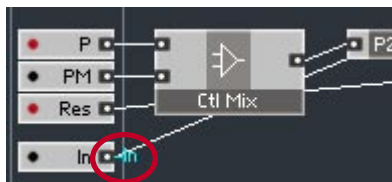


これで並び順は変わりましたが、結線が斜めに横切っていて見た目があまりよくありません。これも修正することにしましょう。

In 入力モジュールの出力部分を右クリックし、**Connect to New QuickBus** コマンドを実行してください。



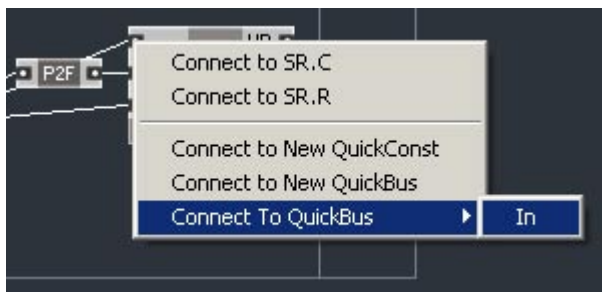
すると次のようになります。



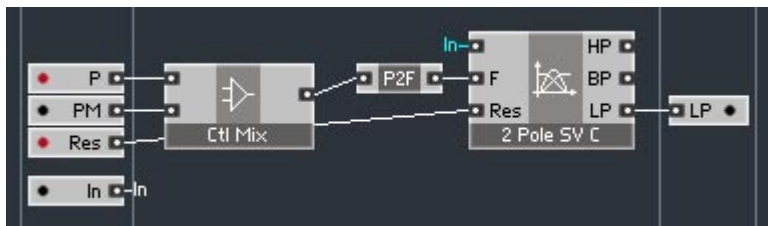
さらに **Properties** ウィンドウも開くので、今作った QuickBus のプロパティを確認できます。ここでは名前を変える機能を最もよく使うことになるでしょう。それ以外のプロパティを変更しなければならないのはごく特殊な場合だけなので、当面はそのままにしておいてください。なお、このウィンドウを閉じてしまっても、QuickBus 上をダブルクリックすればまた開くことができます。

もっとも今回は、初めから QuickBus につながっている入力と同じ名前がついているので、特に変更しなくても構わないでしょう。なお、QuickBus の名前はあるストラクチャー内でさえ区別できればよいので、ほかのストラクチャーや入れ子になっている内側のストラクチャーに同じ名前の QuickBus があっても問題ありません。

次に、**2 Pole SV C** フィルター・モジュールの一番上の入力ポートを右クリックし、接続先として **Connect to QuickBus > In** を選択します。

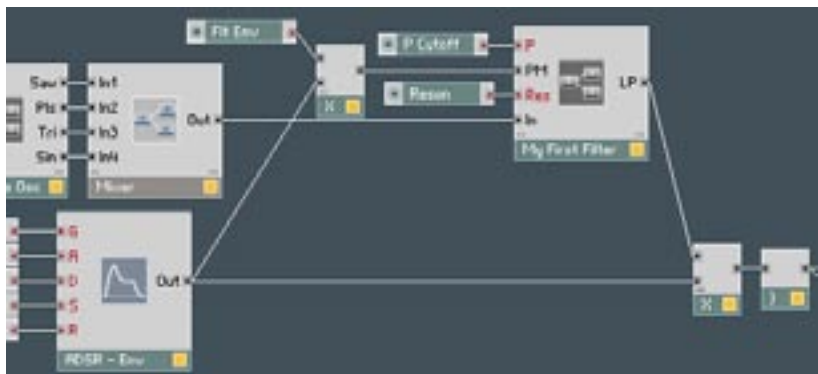


このサブメニューには既存の QuickBus 名が並んでいます。新たに QuickBus を作成しようという場面ではないので、これ以外のメニュー・コマンドはありません。ストラクチャーは次のようになります。



結線は消えていますが、「In」という名前が対応しているので、論理的にはこの間がつながっていることがわかります。

ここで基本レベルに戻り、新しいフィルターを使ってストラクチャーを作り直してみましょう。**Add** モジュール、**A/E** モジュールは不要で、最終的に次のようになります。



CPU に対する負荷は多少増えているようです。変調信号がオーディオ・レートでなので、これはある程度しかありません。これが不都合であればいつでも元のストラクチャーに戻せますし、「反応の遅いエンベロープ」現象には注意が必要ですが、**Multi 2-pole FM** フィルターを使うという選択肢もあります。しかし、これはこれで役立てていただければいいでしょう。さらに、ほかにも有用な機能を備えたフィルターがいくつかありますし、フィルター以外のモジュールも多数揃っているので、試してみる価値はあるはずです。

2.3. オーディオ信号と制御信号

先に進める前に、Reaktor Core ライブラリーの標準マクロ群 (Standard Macros) に特有の約束事を見ておきましょう。これを組み

合わせて作るモジュールの動作を説明するにあたっては、オーディオ、制御、イベント、論理という、信号の種類を意識する必要があります。イベント信号、論理信号についてはあと回しにして、ここではオーディオ信号と制御信号について解説します。

オーディオ信号とは言うまでもなく、音声情報を伝送する信号のことです。発振器やフィルター、増幅器、ディレイ・エフェクトなどの出力がこれに当たります。さらに、フィルターや増幅器、サチュレーター、ディレイなどといったモジュールに対しては、入力としてオーディオ信号を与えるのが普通です。

一方制御信号は、音声情報を運ぶのではなく、他のモジュールを制御するために使います。例えばエンベロープ生成器や LFO(低周波発振器)の出力、MIDI キーボードから得られるノート番号やベロシティなどのデータは、音声そのものではありません。フィルターのカットオフ周波数やレゾナンス、ディレイ・ラインの遅延時間など、さまざまなパラメーターを制御するために使います。当然、フィルターやディレイ・ラインの該当する入力ポートは、制御信号が与えられるものと想定しています。

例えば次に示すフィルター・モジュール **2 Pole SV C** は、既に取り上げました。



一番上の入力ポートにはフィルター処理の対象となる信号を与えるので、ここに入力されるのはオーディオ信号です。一方、F および Res の入力ポートには制御信号を与えます。出力はいずれもフィルター処理を施した音声信号なので、どのポートもオーディオ型ということになります。

一方、次の正弦波発振器 **Sin Osc** には、周波数を調整する制御信号入力が1つと、オーディオ出力が1つしかありません。



さらに別の例として **Rect LFO** モジュールを見てみましょう。これには F および W という、周波数とパルス幅を調整するための制御信号入力があります (ちなみに 3 つ目はイベント信号です)。出力は制御信号です。フィルターのカットオフ周波数や VCA のレベルを調整するのが目的なので、発振器の出力ですがオーディオ信号ではありません。



ミキシングなどのモジュールは、オーディオ信号も制御信号も処理の対象となりえます。このような場合は、同じマクロでも、オーディオ信号用と制御信号用の 2 種類があるはずですが、例えばオーディオ用ミキサーと制御信号用ミキサー、オーディオ用増幅器と制御信号用増幅器、といった具合です。それぞれのマクロはいずれかの種類の信号が与えられることを前提として実装されているので、理解した上であえて使う場合は別として、オーディオ用のマクロに制御信号を与えるようなことはしないでください。

オーディオ信号を制御信号として使えることも珍しくありません。典型的な例としては、発振器の発振周波数やフィルターのカットオフ周波数を、オーディオ信号で変調する、という使い方があります。これは制御の目的でオーディオ信号を使っているわけなので、まったく問題ありません。逆に制御信号をオーディオ信号として使うことは滅多にないでしょう。

信号にはオーディオ、制御、イベント、論理の 4 種類がありますが、Reaktor の基本レベルがイベント型とオーディオ型に分かれているのはまた別の話なので混同しないでください。イベント型 / オーディオ型の区別は処理速度の違いを反映したもので、オーディオ型は CPU に対する負荷がずっと高くなります。さらに、基本レベルとコア・レベルでは、信号の伝播経路にも違いがあります。Reaktor Core でいうオーディオ、制御、イベント、論理の各信号は、処理の種類ではなく、信号の用途にもとづく意味的な違いを反映しています。基本レベルで言うイベント型 / オーディオ型と、Reaktor Core で言うオーディオ / 制御 / イベント

/ 論理の各信号間に、直接的な対応関係はありません。もっとも次のような大雑把な関係はあります。

- ・ 基本レベルのオーディオ信号は一般に、Reaktor Core のオーディオ信号 (発振器出力、フィルター出力など) または制御信号 (エンベロープ生成器の出力など) に対応します。
- ・ 基本レベルのイベント信号は一般に、Reaktor Core の制御信号に対応します。LFO の出力、ノブを回すと得られる制御信号、MIDI キーボードを演奏したときに得られるピッチ (ノート番号) やベロシティ値などがこれに当たります。
- ・ 基本レベルのイベント信号が、Reaktor Core のイベント信号に対応することもあります。MIDI ゲートはその典型的な例です (Reaktor Core のイベント信号については後述)。
- ・ 基本レベルのイベント信号が、Reaktor Core の論理信号に対応することもあります。但し完全に対応しているわけではないので、間に変換処理をはさむ必要があります (詳しくは後述)。例として、**Logic AND** その他の基本レベル・モジュールがあります。

コア・セルの入力ポートはイベント型 / オーディオ型を切り替えることができますが、これは基本レベルの話であって、Reaktor Core 上でイベント信号 / オーディオ信号を切り替えているものではありません。コア・セルのポートは、基本レベルとコア・レベル (Reaktor Core) の境界に位置するため、基本レベルの用語が出てきて混同しがちです。

次にテープ・エコー・エフェクトの動作をエミュレートするストラクチャーを作りながら、この考え方についてもう少し解説しましょう。最初に単純なデジタル・エコーを構築し、これを拡張してテープ・エコーの機能をエミュレートできるようにしていきます。

まず、新規にオーディオ型コア・セルを作成し、コア・レベルの表示に切り替えて、「Echo」という名前をつけます。

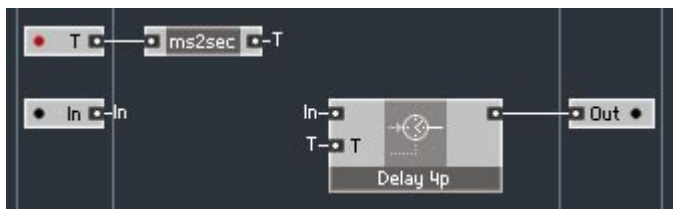
最初にディレイ・モジュールを追加します。ここでは4点補間型のディレイを選択しましょう。2点補間型より品質が高く、また、非補間型のディレイでは今回の目的に合わないからです。**Standard Macro > Delay > Delay 4p** を選択して作成してください。



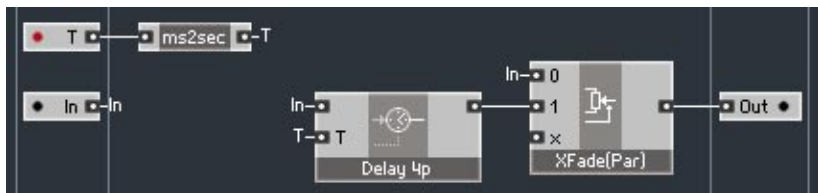
当然これには (コア・セルとして見たときの) オーディオ入力 / 出力が必要です。入力には QuickBus 接続、出力には普通の結線による接続を施すと、次のようになります。



次に、遅延時間を制御するためのイベント入力が必要です。ここで注意しなければならないのは、基本レベルでは遅延時間をミリ秒単位で表しますが、Reaktor Core ライブラリーのディレイ・マクロでは、秒単位で与えられるものと想定している、という点です。とは言っても変換モジュールがあるので問題はありません。**Standard Macro > Convert > ms2sec** で追加してください。



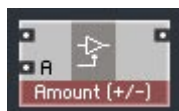
この状態では信号が遅れて聴こえるだけですが、元の信号も聴こえるようにすればエコーが実現できます。そのためには、元の信号と遅延信号をミックスして出力する必要があります。ここで扱うのはオーディオ信号なので、オーディオ・ミキサーを使いましょう (先にフィルター・コア・セルを構築したときに使った、制御信号用のミキサーとは別のものです)。しかも、2つの信号をクロスフェードさせる専用のミキサーがあるので、これを使うことにします。**Standard Macro > Audio Mix-Amp > XFade (par)** で追加してください。



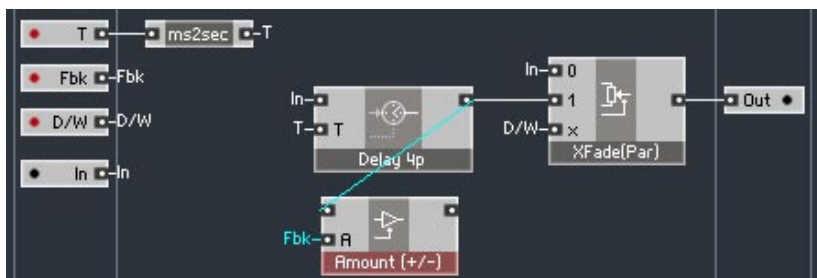
ここで「(par)」は放物型 (Parabolic) を表します。線形のクロスフェードよりも自然に聴こえるという特徴があります。クロスフェードの割合を設定する制御入力 (x) には、「D/W」というイベント入力を用意してその信号を与えます。これは「Dry/Wet」を表し、ディレイ処理前のドライ信号と処理後のウェット信号の比率、という意味の名前です。制御信号の値が 0 であればディレイ処理前の信号のみ、1 にすると遅延のかかった信号のみが聴こえることになります。



これで元の信号とエコー信号の両方が聴こえるようになりましたが、エコーは 1 回しか鳴りません。繰り返しエコーが聴こえるようにするためには、遅延のかかった信号の一部をディレイ入力に戻してやる必要があります。信号の「一部」を取り出すということで、信号を減衰させるため、オーディオ用の増幅器 (増幅率を負にすれば減衰) を使います。**Standard Macro > Audio Mix-Amp > Amount** で追加してください。



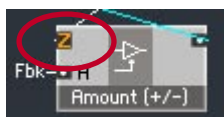
増幅器の中でも特に **Amount** を選んだのは、帰還信号の量を制御したいからです。また、負の値を指定すれば、信号を反転させることができます。逆に **Amp (dB)** などの増幅器は、信号の音量調整には向いていますが、反転信号は取り出せないで今回の目的には合いません。ここでは増幅率 (減衰率) の制御入力 (A) に、帰還の量を制御するイベント入力 (Fbk) をつなぎます。



帰還の量は -0.9 ~ 0.9 程度の範囲で調整するのが適当です。この範囲外の値にすると、すぐに過大な信号レベルになってしまうので注意してください (今回の例では飽和状態にはなりません)。安全のため、値を制限するしくみをコア・セル内に組み込むことも考えられますが、ここでは省略しました。遅延信号が飽和する様子を実験してみても面白いでしょう。



ところで、**Amount** モジュールの一番上の入力には、「Z」という橙色の印がついています。



実は、ソフトウェアの版やその他の条件にもよるのですが、このような入力ポートはほかにもいくつかあります。これはあまり気にする必要はありません。ストラクチャー上にデジタル帰還が発生していることを表すもので、もっと高度なストラクチャーを設計する際には、これがヒントとして役に立つのです。

上記のような単純なストラクチャーでは、1 サンプル分 (サンプル・レートが 44.1kHz ならば 0.02 ミリ秒程度) の遅延が起ころうることを表すだけなので、特に気にする必要はありません。この程度の遅延は、耳で聴いても気がつかないでしょう。

話を戻して、今度は別のモジュールを使い、機能はそのままストラクチャーを小さくしてみましょう。

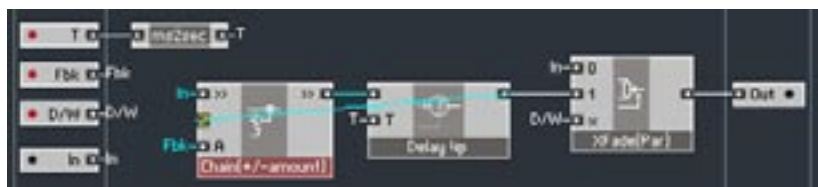
オーディオ増幅器の中に **Chain** というものがあります。これには、信号を増幅し、別の信号 (チェーン信号) とミックスする機能があります。**Audio Mix-Amp > Chain(amount)** はそのような増幅器のひとつで、**Amount** 増幅器と同様の構成ですが、チェーン・ミキシングの機能が追加されています。



このモジュールは、2 番目の入力ポートに与えられた信号を、A 入力に指定された率で減衰させ、チェーン入力 (>>) の信号と混ぜ合わせて出力します。チェーン入力信号は減衰することなくそのまま出力します。この増幅器は本来、一連の信号を次々に混ぜ合わせていく、ミキシング・チェーンを構築するために使うものです。>> のポートを順につないでいくことで、ミキシング・バスができます。



今回の例ではミキシング・バスは必要ないのですが、**Audio Mix** と **Amount** の 2 つを使う代わりに、このモジュールを使って実装することも可能です。先の例と同様に、帰還信号を Fbk 入力で指定される率で減衰し、入力信号と混ぜ合わせます。

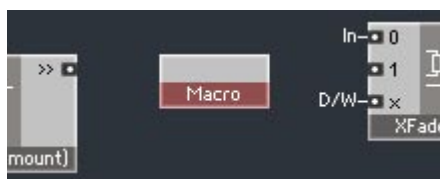


以上、単純なデジタル・エコー・エフェクトを実装してみました。
今度はテープ・エコーの要素を付け加えてみましょう。

2.4. 簡単な Reaktor Core マクロの構築

先に構築したエコー・エフェクトでは、ライブラリーに登録されている **Delay 4p** マクロを使ったので、比較的高音質のデジタル・ディレイを実現できました。しかしそれでも、いわゆるデジタル処理っぽさが前面に出過ぎているようです。音に暖かみを与えるため、テープ・ディレイに見られた、サチュレーションおよびフラッターの効果を追加してみましょう。

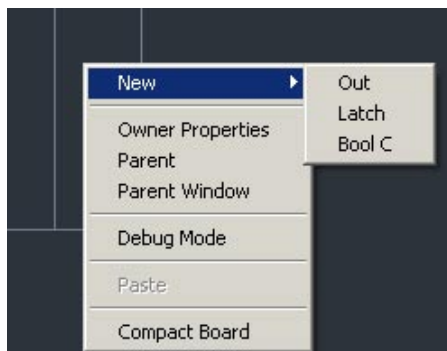
いったんディレイ・マクロを削除し、改めてマクロを作成します。背景部分を右クリックし、**Built-In Module > Macro** を選択してください。



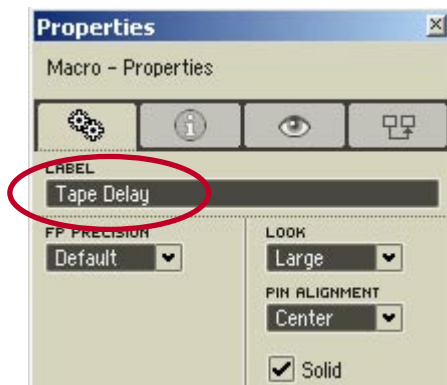
ダブル・クリックしてコア・レベルに切り替えます。これまでと同様の、何もないストラクチャーが現れます。



動作もこれまでとあまり変わるところはありませんが、コア・セルではなくマクロであるということで、いくつか違いがあります。そのひとつとして、次のように、追加できる入力 / 出力モジュールが異なります。



Latch と **Bool C** はやや高度な使い方なので後回しにし、ここでは **Out** のみを取り上げます (入力側は **In**)。これは汎用の入出力ポートで、オーディオ信号、制御信号、イベント信号、論理信号のどれでも扱えます。実際、この区別が必要なのは使う側にとってであって、Reaktor Core の側から見ると、処理の上で何の違いもありません。また、先のストラクチャー上で見ても、オーディオ / イベントというモードの違いはありません。というのも、基本レベルの信号が外に出て行くことはなく、Reaktor Core 内で閉じているからです。最初にマクロ名を変更してみましょう。手順はコア・セルの場合と同じで、背景部分を右クリックし、**Owner Properties** を実行して新しい名前を入力します。



マクロ名 (LABEL) 以外のプロパティーは、外観や信号処理に関するものです。

どのプロパティーもいろいろ試してみても構いませんが、**Solid** パラメーターはオフにしないよう強くお勧めします。また、**FP Precision** も通常は変えないほうがよいでしょう。このプロパティーの意味については、高度な話題としてあとで解説します。

次に **Tape Delay** マクロ用の入出力を作成します。

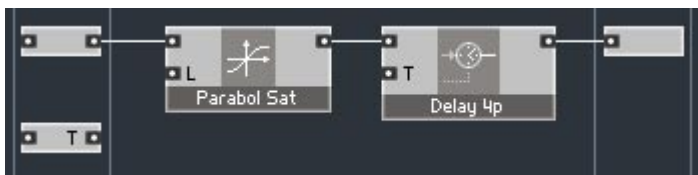


上の入力ポートはオーディオ入力用、下は遅延時間の設定用です。入力モジュールの左側にはほかにもポートがありますが、これについては後述します。

マクロの中心は、同じ **Delay 4p** モジュールです。



サチュレーション・エフェクト (信号の飽和状態) を単純にエミュレートするのであれば、サチュレーター・モジュールをディレイの前段に接続するだけで実現できます。これはオーディオ信号の形状を変える働きがあるので、オーディオ・シェイパーとして分類されています。**Standard Macro > Audio Shaper > Parabol Sat** で追加してください。

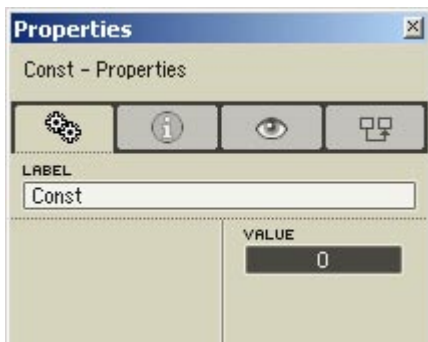


入力信号にサチュレーション処理を施し、その強度を -1 ～ +1 の範囲で調整できるようになりました。強度はモジュールの I 入力で制御しますが、結線しないままにしておいた場合、デフォルト値である +1 が適用されます。普通に考えれば、結線しないと制御信号を受け取れない、あるいは 0 が適用されそうなところなので、少々意外に思われるかも知れませんが。しかし Reaktor Core のストラクチャーでは、入力を結線しなかった場合の動作がモジュールごとに決まっています。サチュレーターの I 入力については、このデフォルト値が +1 になっているということです。

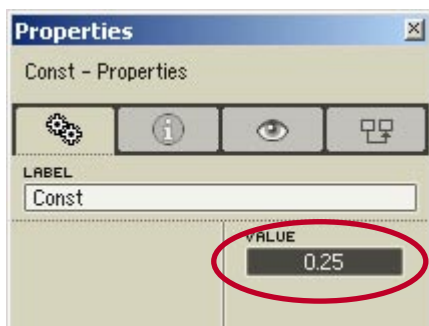
同様に T 入力についても結線せず、デフォルト値を使うことにしましょう。この値を例えば 0.25 秒に変更することができます。T 入力モジュールの左側のポートを右クリックし、**Connect to New QuickConst** コマンドを実行してください。すると次のようになります。



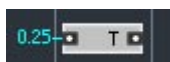
さらに、このデフォルト値を設定する **Properties** ウィンドウが開きます。必要ならばタブを切り替えて、次のような画面にしてください。



VALUE 欄に「0.25」と入力します。



ストラクチャーを見てみると、QuickConst の表示が次のように変わっています。



これは何をしたことになるのでしょうか。入力モジュールの左側のポートは、いわば「デフォルト信号」を表しています。つまり、マクロの外側でこのポートに結線しなかった場合、代わりにデフォルト信号が入力源として与えられるのです。上記の例では、テープ・ディレイ・マクロの T 入力に何も接続されなかった場合、0.25 秒という定数値の信号が与えられたように振る舞います。

デフォルト信号入力の接続先は、QuickConst だけに限りません。ストラクチャー内のどのモジュールにでも接続でき、もちろん他の入力モジュールであっても構いません。

以上でサチュレーションを組み込み、T 入力へのデフォルト値を定めたので、今度はフラッター効果をエミュレートしてみましょう。単純なのは LFO で遅延時間を変調する方法です。発振波形を変えてさまざまなフラッター効果を試すのもよいのですが、ここではライブラリー中から、**Standard Macro > LFO > Par LFO** を選んで使うことにします。



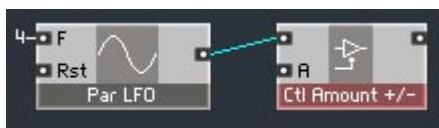
これは放物型 LFO で、正弦波に似た波形の信号を生成しますが、CPU に対する負荷は小さいのが特徴です。F 入力に、発振周波数を指定

する信号を与えます。ここでも QuickConst を使い、4Hz 程度を指定しましょう。



Rst 入力が発振信号の位相をリセットするものですが、ここでは使いません。

次に、LFO 出力の単位を変え、変調強度を指定する信号として与えてやります。LFO 出力の信号レベルは -1 ~ +1 の範囲で変化しますが、これでは変動幅が広すぎます。ここでは制御信号として使うので、**Ctl Amount** で調整します。これは、先にオーディオ用に使った増幅器、**Amount** に似た信号量調整モジュールです。**Standard Macro > Control > Ctl Amount** で追加してください。



変調振幅は 0.0002 程度が適当なので、これを指定して変動幅を調整します。



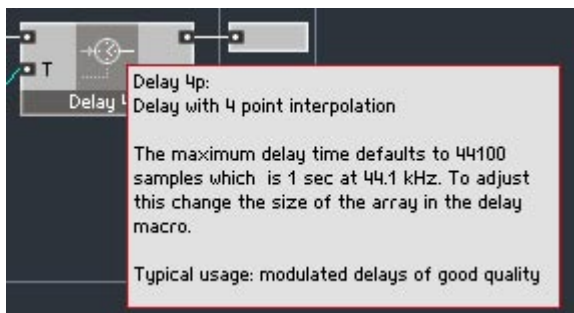
最終的には、2つの制御信号(T入力から得られたものと、**Ctl Amount** モジュールの出力)を混ぜ合わせて、ディレイ・モジュールのT入力に与えるようにする予定です。そのためには、既に何度か出てきた **Ctl Mix** モジュールが使えます。



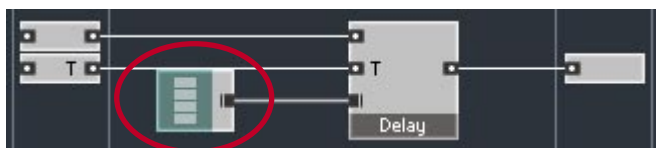
実際にはチェーン型の制御ミキサーも用意されているので、オーディオ信号用にチェーン型のミキサーを使ったのと同様に、**Ctl Amount** と **Ctl Mix** を置き換える形で使うこともできます。これは **Standard Macro > Control > Ctl Chain** で追加します。



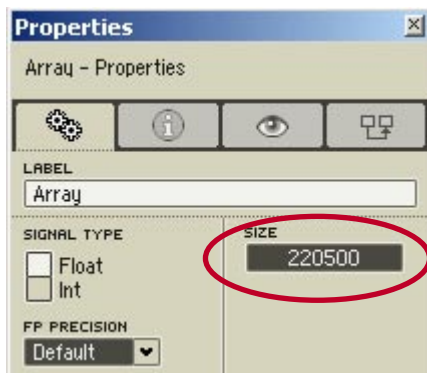
最後にディレイ・モジュールのバッファー長を変更してみましょう。これは最長の遅延時間が決まる要因になります。**Delay 4p** マクロ上にマウス・カーソルを置いて情報を表示すると、サンプル・レートが 44.1kHz の場合に 1 秒分の遅延が可能なだけのバッファー量が設定されていることが分かります。



これを 5 秒分 ($44100 \times 5 = 220500$ サンプル分) に増やしてみましょう。1 サンプル当たり 4 バイトが必要なので、全体では $220500 \times 4 = 882000$ バイト、つまり 1MB 弱のバッファーが必要です。**Delay 4p** マクロ上をダブル・クリックしてください。

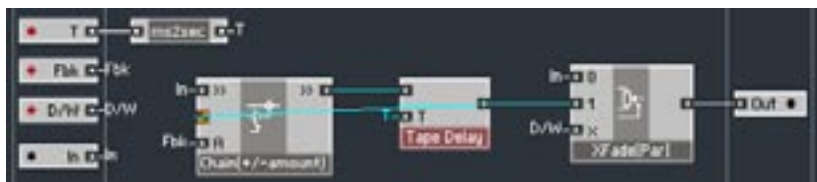


左側にあるのがバッファ・モジュールです。これをダブル・クリックするか、右クリックして **Show Properties** コマンドを実行してください。 **Properties** ウィンドウが開くので、必要ならばタブを切り替え、 **SIZE** として「220500」を入力します。

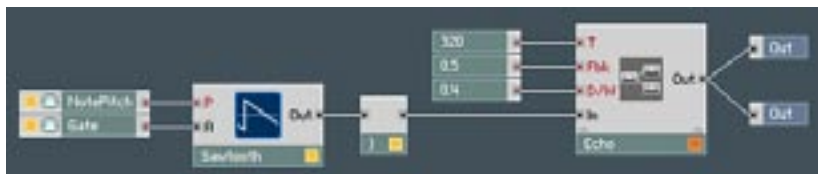


先に計算したように、5 秒分の遅延バッファは約 1MB にも相当するので、変更する際はよく注意してください。さらに、複数の声部を個別に遅延させる場合は、バッファ量も声部数に比例して多くなります。

Delay 4p マクロから元に戻り、さらに (背景部分をダブル・クリックして) テープ・ディレイ・マクロからも抜けて、マクロと他のモジュールを次のように接続します。



参考のため、 **Echo** モジュールも試してみてください。これは基本レベルのテスト用ストラクチャーの中でも最も単純なものです (モノラル用です)。



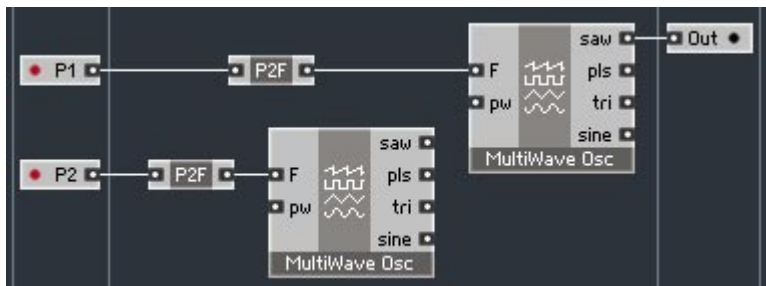
これを元に、エコー・パラメーターを制御するノブを追加する、信号源としてシンセサイザーを使うなど、いろいろ拡張してみるとよいでしょう。

2.5. オーディオ信号を制御信号として使うこと

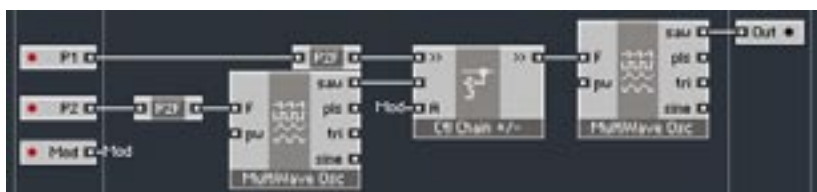
先に述べたように、オーディオ信号を制御用に使うことも可能です。例として、Reaktor Core セルの形で発振器を 2 つ作成し、一方の出力信号でもう一方を変調してみましょう。まず、さまざまな波形を発振できる発振器 (MultiWave Osc) を 2 つ作成します。



発振器のピッチ調整用、および第 2 発振器の出力試聴用に、必要な入出力を用意してください。



次に第 1 発振器の出力を使って、第 2 発振器の周波数を変調します。



Mod 入力の変調強度の調整用です。

ピッチを指定する P1 入力を **P2F** で変換し、変調信号と混ぜ合わせて、第 2 発振器の周波数を制御しています。もちろん **P2F** による変換を省いて、半音単位で設定できるようにしても構いません。

また、変調強度を基準発振周波数に応じて変化させるのも面白いでしょう。



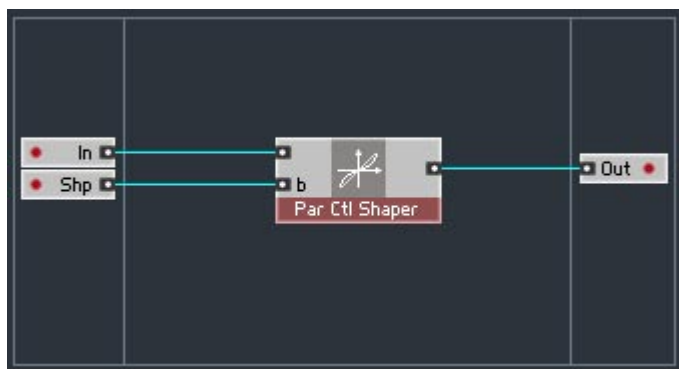
上のストラクチャーを制御信号 / オーディオ信号の違いに注意して見直してみると、発振器の出力以外はすべて制御信号であることが分かります。2つの発振器の出力はもちろんオーディオ信号です。第 1 発振器の出力を **Ctl Chain** に与えています、ここではオーディオ信号を制御信号として使っていることになります。

2.6. イベント信号

先に解説したように、「イベント信号」という言葉の意味は文脈によって異なります。基本レベルでの意味合いは既によく分かっているものと思いますが、これにはさまざまな使い方があります。ひとつは制御信号としての使い方 (LFO 出力、ノブ出力など) で、オーディオ信号に比べて CPU に対する負荷が小さくて済みます。多くの場合、オーディオ信号でも同じ効果を得ることは可能ですが、信号レベルだけではなくタイミングも問題となる場合など、同じ効果が得られない状況もあります。例えば基本レベルのエンベロープ・ゲート信号は、イベントがゲート入力に達した時点でトリガーがかかるため、オーディオ信号を流用したのではタイミングが合いません。

Reaktor Core 上では、信号そのものにオーディオ / 制御 / イベント / 論理という区別はありません。信号の使い方が違っていただけのことです。実際、基本レベルのイベント信号は、制御信号、イベント信号、論理信号のいずれとしても使えますし、先の例にもあったように、基本レベルのオーディオ信号を制御信号として扱うことも可能です。

基本レベルのイベント信号を、Reaktor Core ストラクチャーの制御信号として与える方法については、これまでも説明しました。例えば先に取り上げたフィルターでは、オーディオ型コア・セルの入力ポートをイベント・モードにして信号を与えていました。ほかにもイベント型コア・セルを使って、基本レベルのイベント信号を制御信号として使う状況があります。ここでは「制御信号シェイパー」コア・マクロのラッパーとしてイベント型コア・セルを使う例を紹介しましょう。

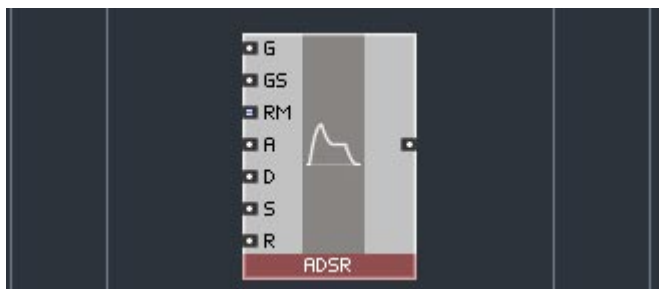


制御信号シェイパーは、イベント・レートの制御信号を基本レベルから受け取り、Shp パラメーターに応じてこれを変形、出力します。MIDI ベロシティ信号、基本レベルの LFO 信号などを変形する、という使い方が考えられるでしょう。

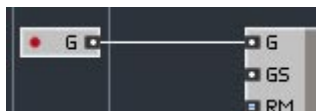
先に述べたように、イベント型コア・セルには、内部では信号源がすべて無効になっている、という制約があります。したがって、発振器やフィルターばかりでなく、エンベロープ生成器や LFO も、イベント型コア・セル内では使えません。こういったモジュールは、Reaktor の基本レベルからイベントを受け取り、処理結果を再び基本レベルに返す、という使い方しかできないのです。

もちろん、基本レベルのイベント信号を、Reaktor Core ストラクチャー内でそのままイベント信号として使うことも可能です。簡単な例を見てみましょう。

ひとつ目の例は、コア・ストラクチャー内でエンベロープ生成器を使うものです。イベント型コア・セルでは信号源が使えないので、オーディオ型コア・セルを作成し、**Standard Macro > Envelope > ADSR** を追加してください。

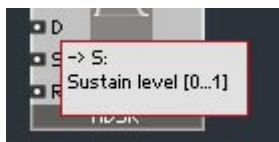


エンベロープの一番上の入力はゲートで、基本レベルの場合と同様、エンベロープ出力の有無を制御することができます。これに対応して、コア・セルにもイベント入力を用意します。

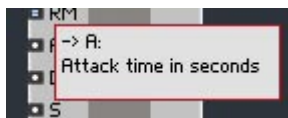


これには、基本レベルのゲート・イベントを、コア・レベルのイベントに変換する役割があります。

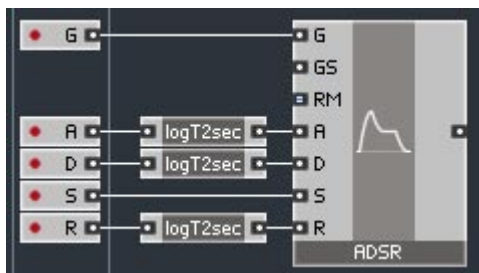
次に A、D、S、R の各入力を見てみましょう。S 入力 (サスティン・レベル) は基本レベルと同様の動作で、0 ～ 1 の範囲の信号が来るものと想定しています。



一方、A、D、R の各入力は、基本レベルと違い、秒単位で指定されるものと想定しています。



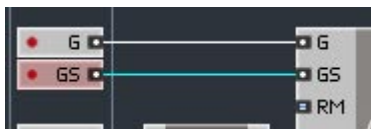
単位の換算用に **Standard Macro > Convert > logT2sec** を使います。



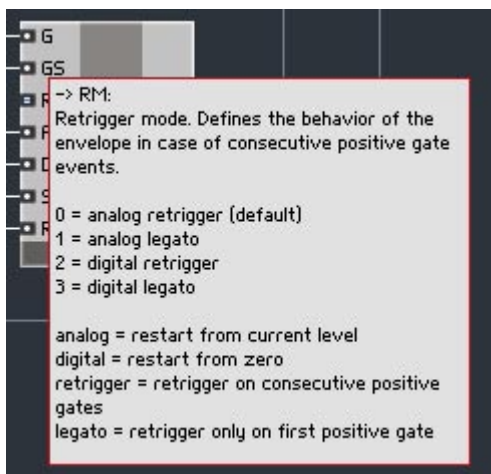
このストラクチャーに対する入力はすべてイベント・モードですが、一番上の入力ポートがイベント信号を出力するのに対し、それ以外のポートは制御信号を出力します。

エンベロープ生成器には未接続のポートが2つ残っています。GS ポートはゲートの感度 (Gate Sensitivity) を表します。これを 0 にすると、G ポートの信号レベルを一切無視して、常に最大振幅で出力ようになります。一方、1 を与えると、G ポートのレベルに応じて振幅を制限

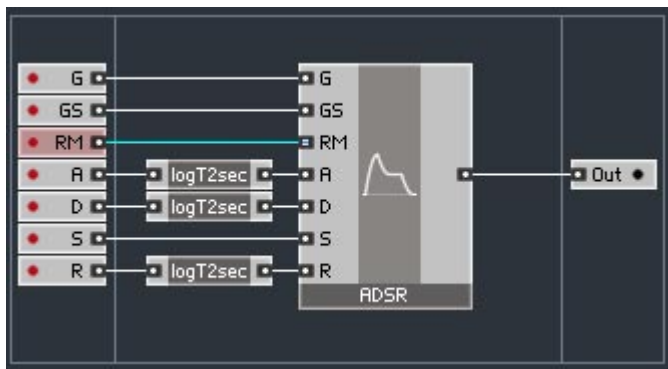
するようになります。これは基本レベルと同じ動作です。次のように入力ポートを追加すれば、ストラクチャーの外部からこれを制御できます。



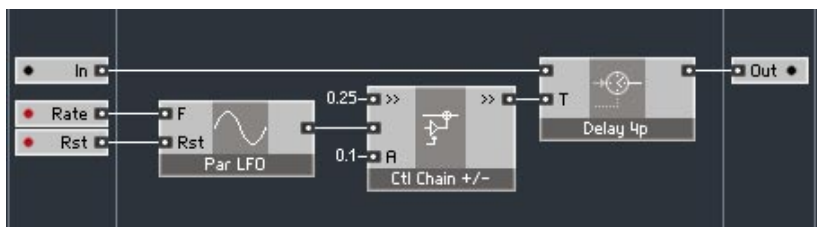
RM ポートはエンベロープに対する再トリガーのモード (Retrigger Mode) を表します。



このポートはほかと違って、整数値が与えられるものと想定しています。と言っても整数値以外の信号が禁止というわけではありません。普通のイベント信号を受け取り、四捨五入して最も近い整数値として扱うだけのことです。



もうひとつイベント信号を使う例を見てみましょう。



このストラクチャーは、ある種のピッチ変調効果を施すものです。機能の中心はディレイ・モジュールで、その遅延時間は 250 ± 100 ミリ秒の範囲で変化します。この変化の速度を制御する Rate 入力 (Hz 単位) は、LFO の発振周波数を指定するポートにつながっています。LFO の出力は純然たる制御信号です。Rst 入力は LFO 出力をある位相にリセットするために使います。値は $0 \sim 1$ の範囲で、これが位相 ($0 \sim 2\pi$) に対応します。実験の際は、決まった値を送信するボタンを接続して試すとよいでしょう。

2.7. 論理信号

制御信号とイベント信号について述べたので、最後に論理信号について解説しましょう。論理信号を処理するモジュールの例を示します。



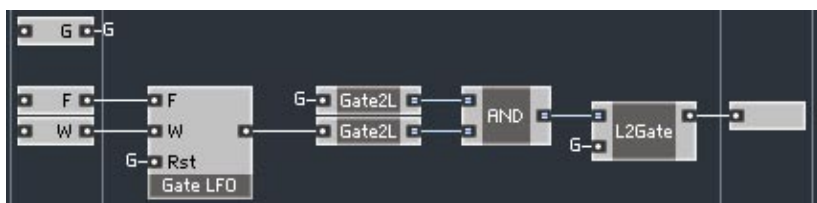
このモジュールの入力は、(エンベロープ生成器の RM 入力と同様に) 整数値を想定しています。一般に論理信号は整数値のみ、しかも 0 と 1 の、2 つの値だけを扱います。

論理信号では、1 は真、0 は偽を表します。具体的に「真」とはどのようなことなのか、「偽」とは何を表すか、については、使う側の解釈次第です。例えば、あるゲートが開いている / 閉じている、という状態を、それぞれ真 / 偽で表します。



この **Gate2L** マクロはゲート信号を受け取り、ゲートが開いていれば真 (1)、閉じていれば偽 (0) を出力します。

論理信号を使って論理演算を行い、例えばクロックに同期して MIDI ゲートを開いたり閉じたりする、ゲート・プロセッサを組み立てることができます。



Gate2L、**AND**、**L2Gate** はいずれも論理モジュールであり、**Standard Macro > Logic** メニュー以下に並んでいます。**Gate LFO** はこの目的のために組み立てたマクロで、ゲートを開く / 閉じる信号を、一定の間隔で交互に生成します。

G (ゲート) 入力と LFO の出力を、**Gate2L** 変換器で論理信号に変換します。ゲートが開いていれば真、閉じていれば偽となります。**AND** モジュールは、どちらのゲートも開いている (真) の場合のみ、真の論理信号を出力します。つまり、LFO がゲートを開く信号を出力している間に、ユーザーがキーを押してゲートを開くと真となります。キーを押さなければなしにしておけば、LFO の周波数に同期して真と偽が交互に入れ替わります。この論理信号を **L2Gate** で再びゲート信号に戻します。この信号のレベル (振幅) は、G (ゲート) 入力と同じになります。

Gate LFO マクロのストラクチャーは次のようになっています。



F 入力 は ゲートを開閉する速度、W 入力 はゲートが開いている時間の割合を表します。W が 0 であればデューティー比が 50% なので、開いている時間と閉じている時間は同じです。-1 ならば 0%、+1 ならば 100% です。Rst 入力 は、あるイベントに応じて、LFO の発振信号をある位相にリセットします。

Rect LFO の Rst 入力につながっているモジュールは **Value** というもので、**Standard Macro > Event Processing** メニュー以下にあります。確実にリセットできるように、イベントが入力されれば常に 0 を出力する、という働きがあります。最後に、LFO の出力をゲート信号に変換するために使う **Ctl2Gate** 変換器は、**Standard Macro > Event Processing** メニュー以下にあります。

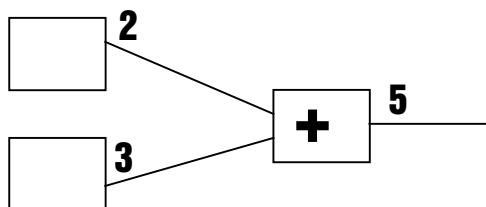
LFO はイベント型コア・セル内では動作しません。このストラクチャーを動かすためには、オーディオ型コア・セルを使う必要があります。

3. Reaktor Core の基礎：コア信号モデル

3.1. 値

Reaktor Core モジュールの出力ポートからは、多くの場合、ある「値」が生成されます。つまり、ある時点で出力ポートを見ると、そこにはある値が設定されているのです。ほかのモジュールは、この出力ポートに自分自身の入力ポートを接続すれば値を参照できます。

次の例で、加算器モジュールの入力ポートには、2 および 3 という値を生成するモジュールの出力ポートがつながっています。

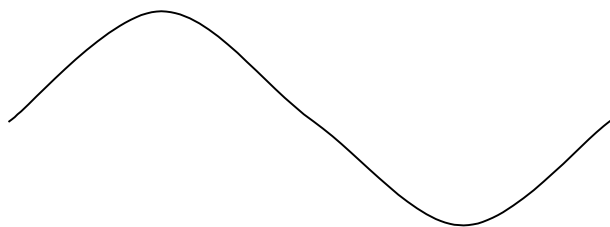


「値」とは、電気回路を念頭に、信号レベル (電圧) のことと考えると分かりやすいかも知れません。発振器、フィルター、エンベロープ生成器など、比較的大規模なモジュールについては、この喩えがよく当てはまります。しかしこれに限らず、「値」の考え方をを使ってさまざまな処理アルゴリズムを実装できます。

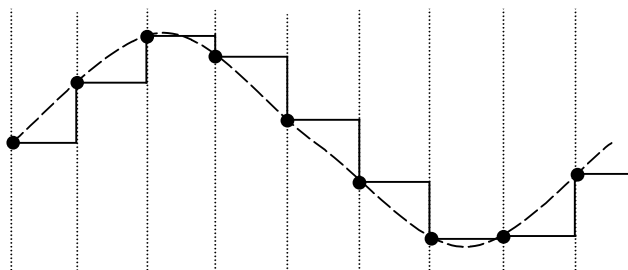
3.2. イベント

ディジタル処理においては、時間も連続したものではなく、離散的な量として扱います。例えばディジタル録音した音声信号には、時間軸に沿って変化する情報がすべて記録されているわけではありません。一定の時間間隔ごとに、その時点の信号レベルを記録しているだけです。1秒あたりの記録回数を「サンプル・レート」と呼びます。

連続量としての信号は次のようになっています。

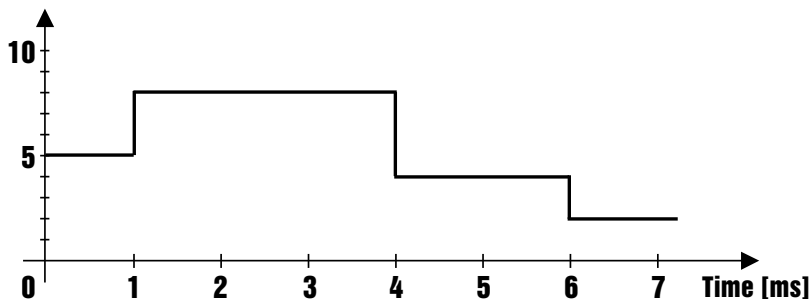


デジタル処理の世界では、これが次のように見えています。



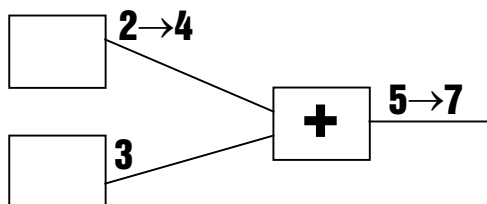
モジュールで実行するのはデジタル処理なので、生成される「値」も連続的に変化するものではありません。しかしだからといって、連続的に変化する信号を与えてはならない、ということもありません。もっと言えば、例えばストラクチャー全体で、一貫したサンプル・レートを適用する必要はないのです。サンプル・レートを決める必要すらないこともあります。この場合、変化は所定の間隔に縛られず、任意の時点で起こりうることになります。

例えば0秒の位置で、加算器が出力する値が5であったとしましょう。次の図のように、最初の変化はそれから1ミリ秒後に起こります。さらに、4ミリ秒後、6ミリ秒後にも変化が起こっています。

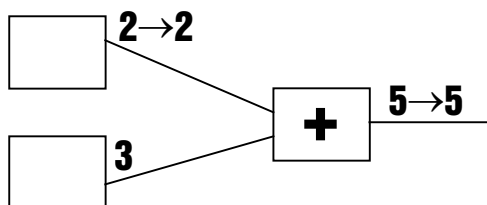


このように、0～7ミリ秒の範囲内でも、加算器の出力値が何回か変わっています。値が変わる時点では、「イベント」が発生します。「イベント」とは、状態が変わって新しい値になった旨を、出力ポートが通知することを表します。

次の図では、左上のモジュールの「値」が2から4に変化し、その旨のイベントを生成しました。加算器モジュールはこのイベントを受けて、自分自身の「値」を変え、新たにイベントを生成します。



一方、「値」が変化しなくてもイベントが発生することがあります。加算器モジュールはそれでもイベントに反応し、自分自身もイベントを生成しますが、「値」はやはり変化しません。



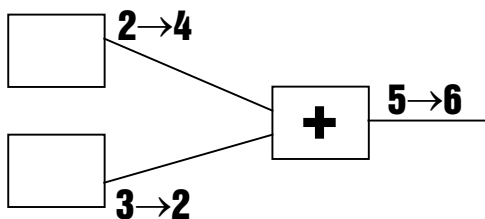
新しい値が元の値と違っている必要はありませんが、逆に値が変わった場合は必ずイベントで通知されます。

この例からも分かるように、あるモジュールの出力ポートでイベントが発生すると、下流のモジュールに順次伝播していきます。図には描いてありませんが、加算器もイベントを生成し、これに続くモジュールに通知しています。ある条件を満たすと伝播は停まりますが、これについては後述します。

Reaktor Core のイベントは、基本レベルのイベントとは振る舞いが異なります。これについても後述します。

3.3. 同時イベント

先の例で、左側の 2 つのモジュールが同時にイベントを生成した場合を考えてみましょう。

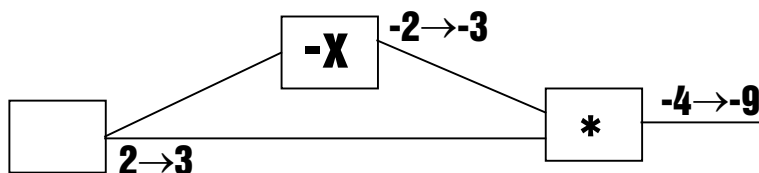


イベントが同時に発生しうることは、Reaktor Core のイベント・モデルを理解する上で鍵となる点のひとつです。2 つのイベントは同時に加算器モジュールに通知されます。重要なのは、これに応じて加算器が生成するイベントは 1 つだけである、という点です。

基本レベルではこれとは違って、イベントが同時に発生することはありません。同様の状況でも、加算器モジュール (イベント・モード) は、イベントを 2 つ生成するはずでは

もちろんどちらのモジュールも 1 基の CPU が処理しているので、厳密に同時というわけではありません。しかしある単位時間内に発生したイベントは、受け取ったモジュール側が、論理的に同時であるものとして扱います。

同時イベントが伝播する例をもうひとつ紹介します。



この例で、左側のモジュールの値が2から3に変わり、その旨のイベントが発生したとしましょう。これが反転器 (-X) と乗算器 (*) に、同時に伝播します。反転器はイベントに反応して値を -3 に変えます。そしてここからが重要なのですが、反転器からのイベントは1段階遅れて届くにもかかわらず、乗算器は2つのイベントが論理的に同時に発生したものとみなします。したがって、乗数と被乗数が同時に変化したと考え、値が -9 に変わった旨を表すイベントを1つだけ生成するのです。

基本レベルではやはり振る舞いが異なり、乗算器モジュールはイベントを2つ生成します。しかも、最初に発生したイベントが、反転器に届くのが先か乗算器に届くのが先かは不定です。この例ではいずれにしても結果は変わりませんが、ストラクチャーによっては問題になることがあります。

2つのイベントが同時に発生したとみなされるかどうかは、次の基準で判断してください。

ひとつのイベントに応じて生成されたイベントは、すべて同時に発生したとみなします。同時に発生した複数のイベントに応じてそれぞれ生成されたイベントも、すべて同時発生とみなします。

後半の例には同時イベントの利点が現れています。すなわち、乗算器は処理が一度で済むため、不必要に CPU を消費することがなくなっています。同時イベントの機構がなかったとすれば、大規模なストラクチャーではイベント数が増えすぎて制御できなくなってしまう恐れがあるため、無駄なイベントが発生しないよう設計時に特別な配慮が必要になります。

CPU への負荷を別にしても、特に低レベルの DSP アルゴリズムを実装する場合、同時イベントの機構があるかないかでストラクチャー構

築の方針がかなり違ってきます。実際に独自のストラクチャーを作ってみれば、ありがたみがよく分かることでしょう。

3.4. 処理順序

前節のように、あるモジュールが生成したイベントは、下流のモジュールに順次伝播していきます。各イベントの発生は論理的に同時ですが、各モジュールの処理も同時というわけではありません。ちょっと考えてみれば、上流のモジュールが処理を終えてから下流のモジュール処理が始まるのが自然であると思われるでしょう。実際にもそのようになっています。

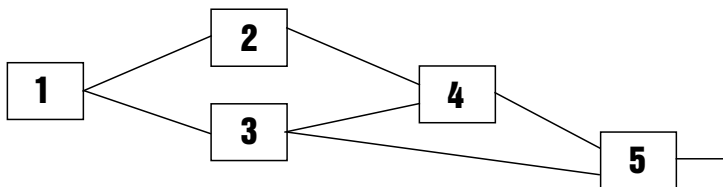
処理順序は一般に、次のように決まります。

互いに接続されている 2 つのモジュールが、論理的に同時に起こったイベントを処理する場合、上流モジュールが先に処理します。同時ではないイベントについては、もちろんイベントの発生順です。

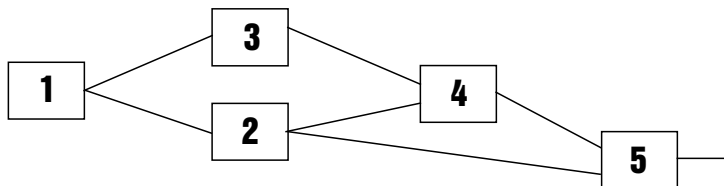
この規則の帰結として、間に他のモジュールがはさまっていても一方方向の接続経路ができていれば、この経路に関する上流側のモジュールが先になります。

逆にモジュール間が一方方向の経路で結ばれていない場合、イベントの処理順序は不定です。実際、さまざまな要因により順序が入れ替わることもあります。順序を変えても結果が変わらないのであればそれでも構いませんが、一般には順序が不定であることのないよう、設計時に配慮しなければなりません。もっとも、オブジェクト・バス接続 (4.2 節を参照) を使っていないのであれば、処理順序は問題にならないのが普通です。

例えば次のようにモジュールがつながっていれば、番号の順に処理されます。

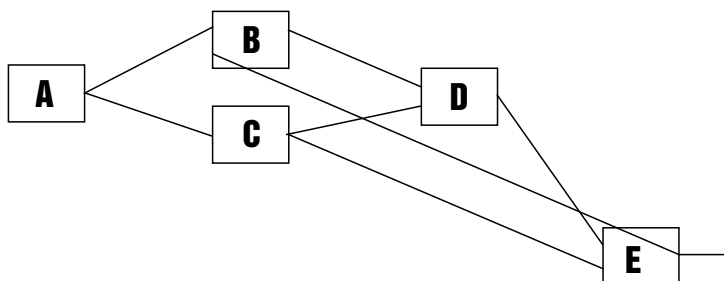


これが次の順序で処理される場合もあります。



どちらの順序になるかを定めることはできません。とは言え、OBC接続を使っていなければ、いずれにしても結果に違いはありません。

上記の順序決め規則は、信号が帰還する経路があれば適用できません。帰還ループ上のモジュールは、どちらが上流であるかを判定できないからです。帰還ループには処理順序以外にも考慮しなければならない問題がいくつかあるので、別途詳しく解説します。



上のストラクチャーでは、例えばモジュール B と D のどちらが上流であるか判定できません。B の下流に D があるように見えますが、D から E を経由して B に到る経路もあるからです。

3.5. イベント型コア・セル再説

イベント型コア・セルについて、イベントの観点から見直してみましょう。

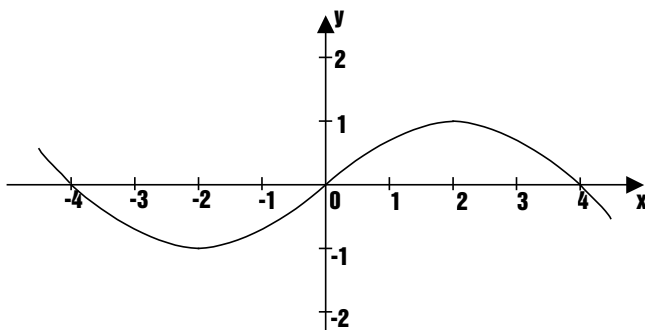
先にも説明したように、イベント型コア・セルにはイベント入力とイベント出力があります。これには基本レベルとコア・レベルの受け渡しをする、インターフェイスとしての役割があります。つまり、基本レベルのイベントとコア・レベルのイベントを、次のように相互に変換しているのです。

イベント入力は、外部から基本レベルのイベントを受け取り、内部にコア・レベルのイベントを送出します。基本レベルのイベントが複数同時に届くことはないので、コア・レベルのイベントを複数同時に送出的ることもありません。

イベント出力は、内部からコア・レベルのイベントを受け取り、外部に基本レベルのイベントを送出します。コア・レベルではイベントが同時に発生することがありますが、基本レベルでは同時に送出的ることができないので、上の方にある出力ポートから順に送り出すようになっています。

練習のため、 $y = 0.25 * x * (4 - |x|)$ という式で信号を変換するイベント処理モジュールを作ってみましょう。

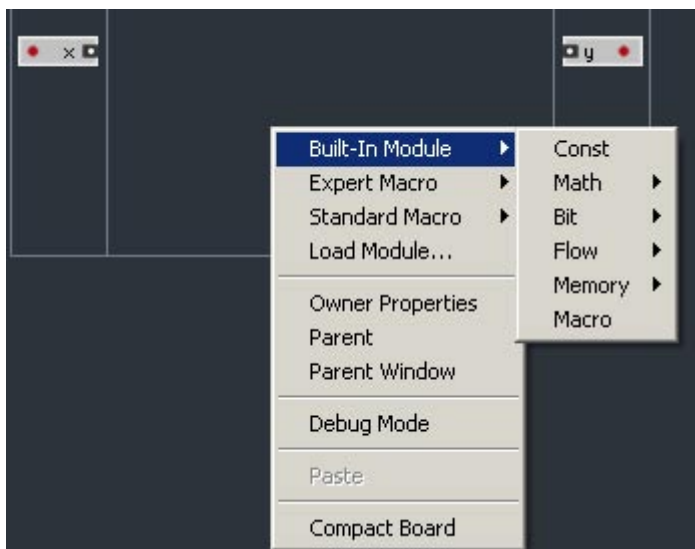
この式のグラフを描くと次のようになります。



イベント型コア・セルを新規に作成し、入力ポートと出力ポートを1つずつ追加して、「x」、「y」という名前にしてください。



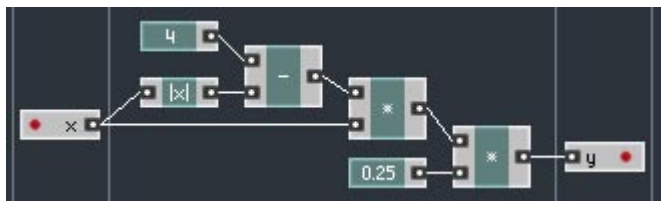
式に従って計算するストラクチャーを組み立てていきます。式から分かるように、絶対値、減算、乗算のモジュールが必要です。これはマクロではなく、Reaktor Core の組み込みモジュールです。処理部の背景部分を右クリックし、**Built-In Module** サブメニューを開いてください。



この中の **Math** 以下に、必要なモジュールがあります。

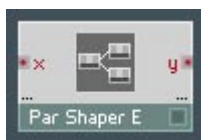


さらに、0.25 および 4 という定数も必要です。先に紹介した QuickConst を使う方法もありますが、ここでは定数モジュールを使ってみましょう。**Built-In Module > Const** を選択してください。定数の具体的な値は、**Properties** ウィンドウで設定します。

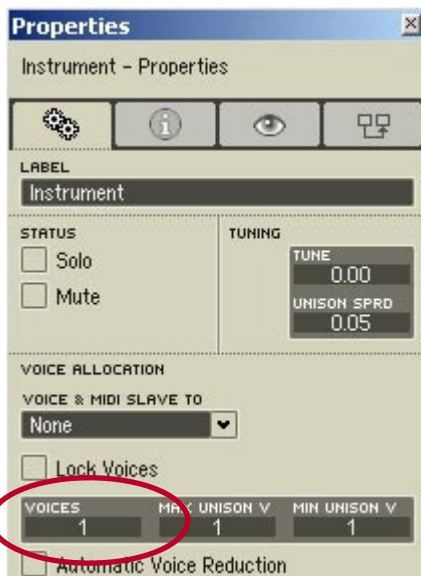


この例では定数モジュールでも QuickConst でも特に違いはありませんが、定数モジュールの方が便利な場合もあります。例えば同じ定数をいくつかのモジュールの入力につなぐ場合、プロパティを 1 箇所を設定し直すだけで具体的な値を変更できます。

上の図のように接続すると、式に従って信号の値を変換できるようになります。実際には不備が残っているのですが、それについてはあとで説明します。当面はこれで完成したものとし、名前をつけて基本レベルに戻ってください。



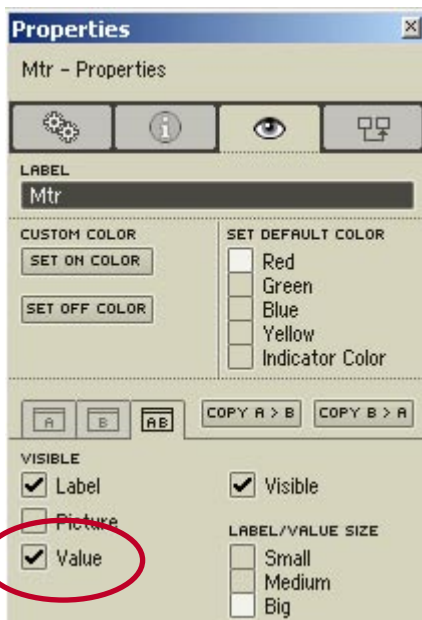
想定通りに動作するかどうか試してみましょう。Reaktor インストゥルメントの声部数を 1 とすると、**Meter** モジュールが使いやすくなります。



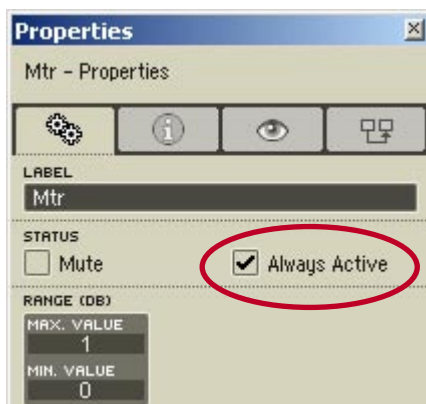
ノブとメーターを用意し、先に作ったモジュールと次のように接続します。



ノブとメーターのプロパティを設定しましょう。特にメーターについては、**Value** をオンにして、値が表示されるようにしなければなりません。



さらに、**Always Active** もオンにしておきます。



ノブを動かして入力値を調整し、出力がどう変わるか、メーターで見
てみましょう。



このように、式に従って信号の値を変換できるようになりましたが、不備が若干残っています。イベント処理の振る舞いについてもう少し説明してから、この不備を解消することにしましょう。

4. 内部状態を持つストラクチャー

4.1. クロック信号

Reaktor Core の各モジュールがイベントをどのように処理するかは、モジュールごとに決まっています。イベントに付随する「値」に対して処理するのが普通ですが、これをまったく無視する場合もあります。例えば**クロック入力**をイベントの形で受け取る場合、値を参照することはほとんどありません。

クロック入力が必要とするモジュールの例として **Latch** があります。これは組み込みモジュールではなくマクロとして実装されていますが、クロックの使用例を紹介するのにはうってつけです。

ラッチには、値を設定する入力と、クロック入力があります。



図の左上、値入力ポートにイベントを与えると、その値が内部のメモリに記憶されます。この時点では何も出力されません。クロック入力ポート (下側、C) にイベントを与えると、直前に記憶された値が出力されます。

モジュールは一般に (特に仕様として述べているものを除き)、クロック入力のイベント値は参照せず、イベントが届いたという事実だけを認識します。

ラッチ・モジュール自身の動作については別に説明することとし、ここではクロック信号についてのみ解説します。

クロック入力のように、値を使うことがない (有用な情報を値として伝えない) 信号もある、という点を改めて明確に述べておきましょう。もっぱらクロック入力に与えることを想定して生成される信号のことを、「クロック信号」と呼びます。

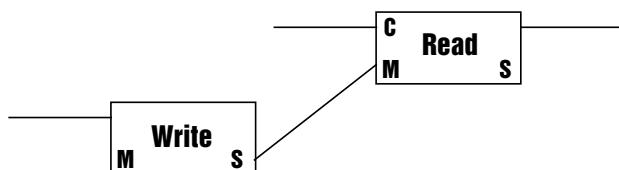
クロック信号の例としては、サンプル・レートに合わせて一定間隔で生成される信号があります。オーディオ・サンプルが生成されること、したがってサンプル・レートが 44.1kHz であれば 1 秒に 44100 回の

割合で生成されます。イベントの値には意味がなく、使うこともないので、(現在の実装では)常に0になっています。

4.2. オブジェクト・バス接続 (OBC)

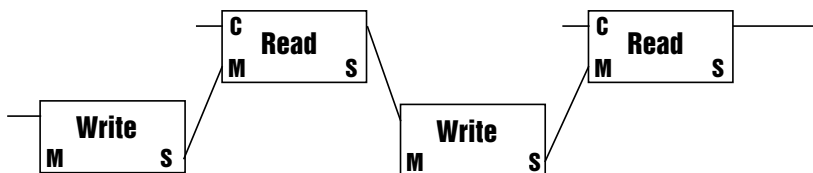
モジュール間の特殊な接続として、「オブジェクト・バス接続」(OBC、Object Bus Connections) というものがあります。モジュール間で内部オブジェクトを共有する形の接続のことで、典型的な例としてメモリー読み込み / 書き出しモジュールがあります。この2つをオブジェクト・バス接続すると、同じメモリー領域が共有されます。

Write モジュールは、与えられた入力の値を OBC 共有メモリーに書き出します。**Read** モジュールは、クロック信号 (C 入力) に合わせて、OBC 共有メモリーの値を読み込んで出力します。



上のストラクチャーは、**Latch** マクロの機能を実装したものです (実際のマクロもこのようになっています)。M はマスター (Master) 入力、S はスレイブ (Slave) 出力を表します。**Read** モジュールのマスター入力を **Write** モジュールのスレイブ出力に、OBC でつなぎます (ほかのマスター / スレイブ入出力は使いません)。これにより、2つのモジュールが同じメモリー領域を共有できるようになります。

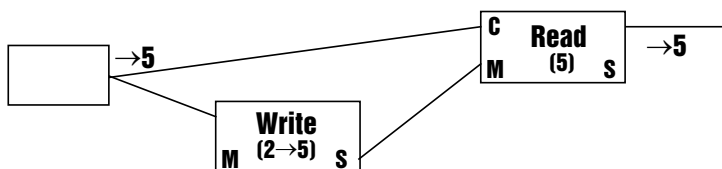
次のストラクチャーには **Read/Write** モジュールが2組あり、それぞれに独立した共有メモリーがあります。**Read** の出力から **Write** の入力につながる、中央の接続は OBC ではありません。



マスターとスレイブの区別は何を表しているのでしょうか。共有オブジェクト (この例では共有メモリ) の所有関係に関して言えば、何の違い也没有ありません。しかし先にも説明したように、同時イベントは上流モジュールから先に処理する、という規則があります。したがっていずれの例でも、マスターである **Write** モジュールが先に処理してから、そのスレイブである **Read** モジュールの処理に移ることになります。逆になることはありません。

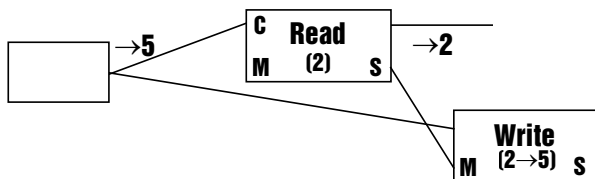
オブジェクト・バス接続されたモジュール間の処理順序も、そうでない場合と同様、上流が先になります。

2つの場合に分けて考えてみましょう。共有メモリの初期状態が2であったとし、値を5とするイベントを、同時に **Read** および **Write** モジュールに送ったとします。すると、**Read** がマスターである場合と **Write** がマスターである場合とで、次のように結果が違います。



上のようなストラクチャーでは、左のモジュールが送ったイベント (値が5) は、先に **Write** モジュールに届くため、共有メモリの状態が5になります。次に **Read** モジュールのクロック入力にイベントが届き、**Read** モジュールはこの5という値を読み取って出力します。ちなみにこれは、Reaktor Core マクロ・ライブラリーにある **Latch** マクロと同じ機能です。

今度は次のようなストラクチャーを考えてみましょう。

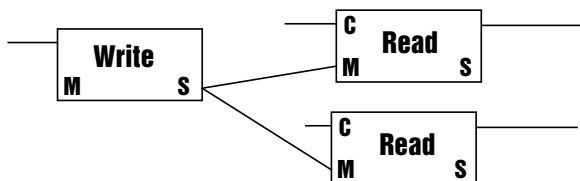


順序が入れ替わって、先に **Read** モジュールにクロック・イベントが届くので、初期状態である 2 という値を読み取って出力することになります。それから **Write** モジュールにイベントが届き、値が 5 に変わります。これは DSP で 1 サンプル分の遅延信号を得るためによく使われる、「Z⁻¹」ブロックの機能に相当します。

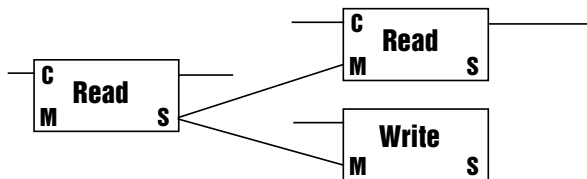
実際に「Z⁻¹」ブロックを実装するためには、ほかにも考えなければならない事項があります。

3 つ以上のモジュールをオブジェクト・バス接続した場合も、同じオブジェクトが全モジュールに共有されます。この場合は処理順序がどうなるかを理解しておくことがさらに重要になります。

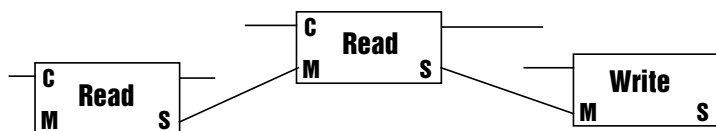
例えば次のストラクチャーでは、2 つの読み込み処理の順序は不定です。しかしいずれにしても書き込み処理のあとなので、結果に違いはありません。



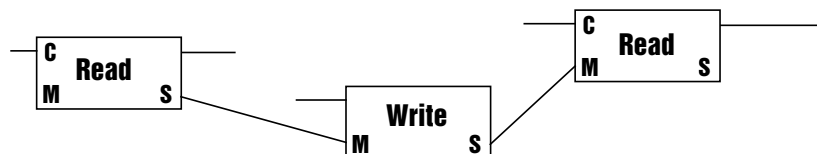
一方、次のストラクチャーの場合、書き出し処理と、右側の読み込み処理の順序が不定です。結果がその都度異なり安定しないので、回避しなければなりません。



例えば次のように組み直せば問題は解消します。



あるいは次のようにしてもよいでしょう。



読み込みと書き出しの順序が問題にならない場合であっても、順序が不定であるような接続はできるだけ避けた方が、多少なりとも安全になります。

書き出し処理の順序は重要です。これが常に一定であれば、読み込み処理の順序は不定であっても一般に問題ありません。

オブジェクト・バス接続と、そうでない一般の信号接続を、単純に置き換えることはできません。同じオブジェクト・バス接続どうしても、共有するオブジェクトの種類が違えばやはり置き換えることはできません。オブジェクトが浮動小数点数であって、その精度が異なるだけであっても同様です。さらに、一般の信号出力と OBC 入力など、種類の異なるポートどうしを接続することもできません。

4.3. 初期設定

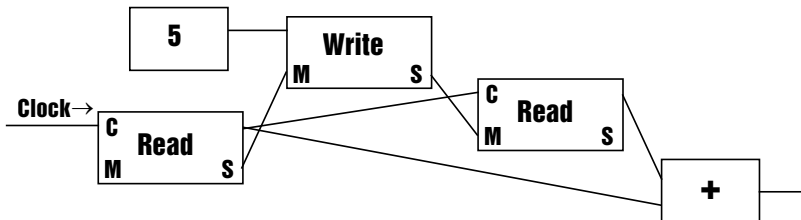
内部状態 (**Read/Write** であれば共有メモリー) を持つオブジェクトについては、その初期状態の設定も大切です。まだ値を書き出していない共有メモリーを **Read** モジュールで読み込むとどうなるか、以下に説明します。また、デフォルトの初期状態では困る場合の対処についても解説します。

これは Reaktor Core の初期設定のしくみに関する問題です。コア・ストラクチャーの初期設定は、次のような手順になっています。

- ・はじめに、状態を持つ要素がすべて、デフォルトの初期状態 (多くは 0) になります。共有メモリーやモジュールの出力値も、特に明記しているものを除き、0 に設定されます。
- ・次に、初期設定イベントが一斉に送られます。(コア・レベル内に) 入力のないモジュールは、ほとんどがこの初期設定イベントの送信元となります。**Const** モジュール (QuickConst を含む) やコア・セル入力がこれに当たります。設定される値は、**Const** モジュールであればその定数、コア・セル入力であれば基本レベルのストラクチャー上で外部から与えられた値になります。

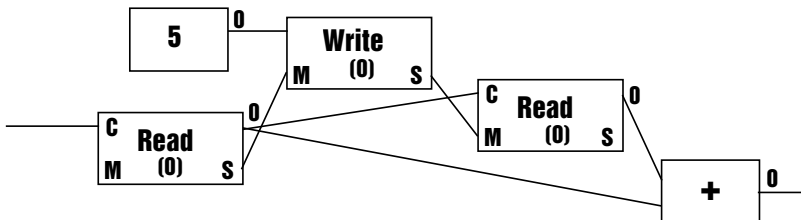
初期設定イベントの送信元になりうるモジュールについては、マニュアルの該当する箇所に、初期設定に関する説明も載っているはずです。それ以外のモジュールは、初期設定イベントを一般のイベントと同じように扱います。送信元になりうるのは、多くが (コア・レベル内に) 入力のないモジュールです。

初期設定の様子を次の例で見ましょう。

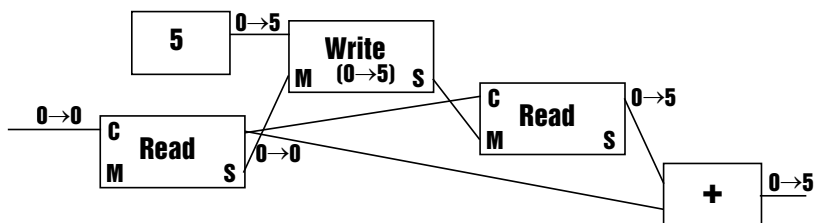


これはあるストラクチャーの一部です。左側の **Read** モジュールにはクロック源がつながっており、これが初期設定イベントも運んできます (クロック源は一般にそうになっています)。

初期状態では、Read-Write-Read という一連のモジュールの内部状態も、右端の信号出力も、0 になっています。

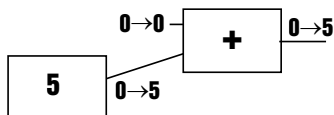


クロック源、および値 5 の **Const** モジュールから、初期設定イベントが同時に送り出されます。



左側の **Read** モジュールの処理が先に始まります。ここにクロック・イベントが届いた時点では、まだ共有メモリの値が変わっていないので、このモジュールは0を出力することになります。次に **Write** モジュールが、共有メモリに新しい値である 5 を書き出します。今度は右側の **Read** モジュールにトリガーがかかって、5 を出力します。最後に加算器モジュールが処理を行い、和として 5 を出力します。

先に説明したように、入力に何も接続されていないければ、0 が入力されたものとして扱います (そうでない振る舞いを指定できるモジュールもあります)。きちんと言えば、値が 0 の **Const** モジュールがつながっていて、「ここから初期設定イベントが届く」とみなされるわけです。



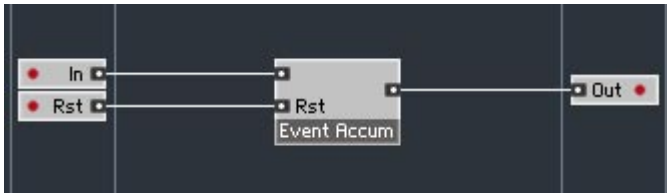
この例で、加算器の一方の入力に何も接続されていない場合、値が 0 の **Const** モジュールがつながっているものとみなされるので、2 つの入力ポートに同時に初期設定イベントが届きます。

信号入力ではない、という意味で、入力ポートに何もつながずにおくこともあります。この場合は、値が 0 の **Const** モジュールもつなげません。たとえば **Write** モジュールのマスター入力に何もつながないでくと、共有メモリの連鎖がここから始まる、という意味になります。

4.4. イベント累算器の作成

この節で作成するイベント累算器 (Event Accum) モジュールには、2つの入力ポートがあります。累積されるイベント値の入力と、これを0にリセットする信号入力です。さらに、現時点までに累積された値を出力するポートもついています。

イベント型コア・セル内に取り込んで使いやすいよう、コア・マクロの形で作成することにしましょう。



マクロの中身が空の状態から始めます。

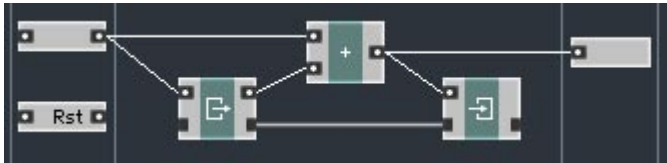


このモジュールには当然、現在までに累積された値を保存するための、内部状態が必要です。ここでは **Read/Write** モジュールを使って実現しましょう。これは **Built-In Module > Memory** サブメニュー以下にあります。



左側の、箱から外に出る矢印がついたモジュールが **Read** です。右側にある、箱の中に入る矢印のついたモジュールが **Write** です。

イベントが入力されると、累算器モジュールは現在の値にこれを加算します。したがって、**Read** モジュールで現在の内部状態を読み込み、加算器で処理した後、**Write** モジュールで書き出すことになります。



Read モジュールにはイベント信号がクロック源として与えられます。下流にある **Write** モジュールとオブジェクト・バス接続することにより、読み込みと書き出しの順序を決めていることに注意してください。

上のストラクチャーは、入力された値を累積し、現時点までの合計値を出力しています。しかしリセット機能や初期状態を決める機能はまだありません。

まずリセット機能を組み込んでみましょう。これはコア・レベルなので、In および Rst の 2 つの入力に、同時にイベントが届く可能性があります。使いやすいコア・マクロにするためには、この点を考慮に入れて設計しなければなりません。実際に同時に届いた場合、どのように動作してほしいでしょうか。リセット処理が先であるか後であるかによって、結果が異なります。これは先に紹介したラッチと「Z-1」の違いにも似ています。

ここではラッチのように動作するものとして作成してみましょう。こちらの方が直感によく合っています。ラッチの場合、クロック信号は値信号よりも論理的に後に届いたものとして処理していました。同様に、リセット信号は加算される値よりも論理的に後に届いたものとして扱います。したがって、何らかの手段で、累積値出力を強制的に 0 にする必要があります。そのためには、イベントのマージという新しい概念を導入しなければなりません。

4.5. イベントのマージ

2 つの信号を入力するモジュールとしては、加算器などいくつかの例を紹介しましたが、単に信号をマージする手段はまだ出てきていませんでした。

これは加算器と同様に 2 つの信号を受け取りますが、何の処理も加えず、後から受け取った方をそのまま出力するという動作です。それには **Merge** モジュールを使います。実際の動作を見てみましょう。

Merge モジュールには 2 つの入力があります。初期状態 (初期設定イベントを受け取る前) では、出力は 0 になっています。



ここで第 2 入力に値 4 のイベントが届いたとします。



するとイベントはそのままモジュールを通過するので、出力値は 4 になります。

次に第 1 入力に値 5 のイベントが届きました。



やはりイベントはそのままモジュールを通過するので、出力値は 5 に変わります。

今度は 2 つの入力に、同時に値が 2 および 8 のイベントが届いたとしましょう。



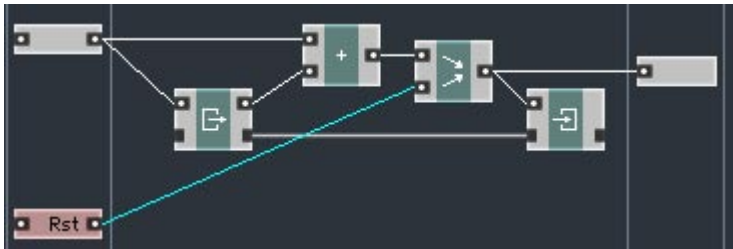
この場合は、**Merge** モジュールに特有の規則が適用されます。

Merge モジュールの複数の入力にイベントが同時に届いた場合、入力ポート番号の小さい方から順に処理されます。Reaktor Core の出力に一般に適用される規則に従い、同時に生成される出力イベントは 1 つだけです。

上の状況では、先に第 1 入力のイベントが処理されますが、すぐに第 2 入力の値で上書きされるので、8 が出力されることになります。

4.6. リセット / 初期設定機能つきのイベント累算器

先に説明したように、リセット機能を組み込むためには、加算器の出力をある初期値で上書きするしくみが必要です。ここでは **Built-In Module > Flow** サブメニュー以下にある **Merge** モジュールを使います。最も簡単なのは、このモジュールの第 2 入力ポートに、Rst 入力をつなぐ方法です。



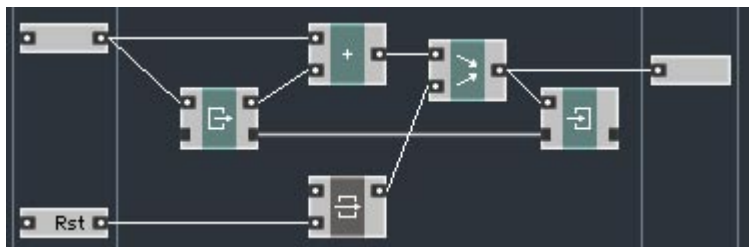
「リセット」イベントは直ちに **Merge** モジュールに入り、加算器の出力が同時に届いたとしても、この値を上書きします。さらに、**Write** モジュールを介して内部状態にも反映されます。

上のストラクチャーで、Rst 入力に与えた値は、累算器をリセットしたときの初期値となります。したがって厳密にはリセット機能というより値設定機能ですが、Reaktor の標準イベント累算器モジュールの動作と同じですし、これはこれで便利に使えるでしょう。本当にリセット機能にしたいのであれば、Rst 入力の値を無視して、定数 0 を **Write** モジュールに与えるようにします。

入力イベントに合わせて特定の値のイベントを送る、という機能は頻繁に必要なため、そのためのマクロが用意されています。**Expert Macro > Memory** メニュー以下にある **Latch** がこれに当たります。



先にも述べたように、**Latch** モジュールには値入力 (上) とクロック入力 (下) があります。値入力に定数 0 を与えておき、Rst 入力をこのクロック入力に供給すると、これに合わせて値が 0 のイベントが発生します。値が 0 なので、値入力に何もつながないでいても構いません。



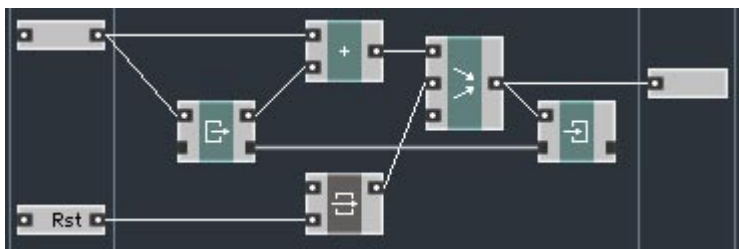
これでリセット機能が想定通り働くようになりました。

最後に、初期設定を正しく行えるようにする必要がありますが、そのためには、「正しい」とはどういうことかをはっきり定義しなければなりません。上記のストラクチャーはどのように初期設定すればよいかを考えてみましょう。

初期設定イベントが、コア・セル自身の In 入力および Rst 入力、および **Latch** に入力される値 0 の **Const** モジュール (接続はしていないがあるとみなされる) から同時に送られたとします。すると、**Latch** は Rst 入力に応じて値 0 を **Merge** モジュールの第 2 入力に送ります。第 1 入力に別の値が与えられていても、第 2 入力の 0 で上書きされるので、内部状態は 0、コア・セル全体の出力も 0 となります。完璧ではないでしょうか。

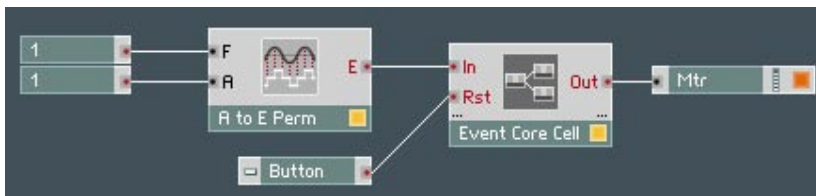
しかし 1 つ問題があります。初期設定イベントが一方の入力ポート、あるいは両方に届かなかったとしましょう。コア・セル自身の入力に初期設定イベントが与えられなければ、このような情況になりえます。また、このマクロがもっと複雑な Reaktor Core ストラクチャーの一部として使われていれば、一部の結線にしか初期化イベントが到達しないこともありえます (実例はあとで紹介)。そこで、こういった場合にも対処できるよう修正してみましょう。

Merge モジュールの **Properties** ウィンドウを開き、入力ポートの個数を 3 に変更してください。

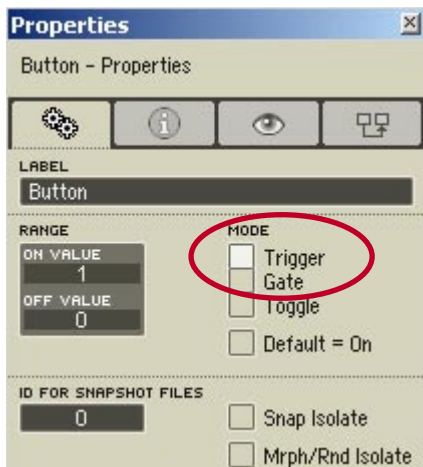


すると、Rst 入力に初期設定イベントが与えられない場合でも、第 3 入力には値 0 の **Const** モジュールがつながっているとみなされるので、正しく 0 が出力されます。

できあがったイベント累算器モジュール **Event Accum** の動作を試してみるため、基本レベルに戻り、次のようなストラクチャーを作りましょう。



前回の例と同様、インストゥルメントの声部数を 1 に設定し、メーターは常に値が表示されるようにしてください。また、ボタン (Button) は次のように、トリガー・モードにしておきます。

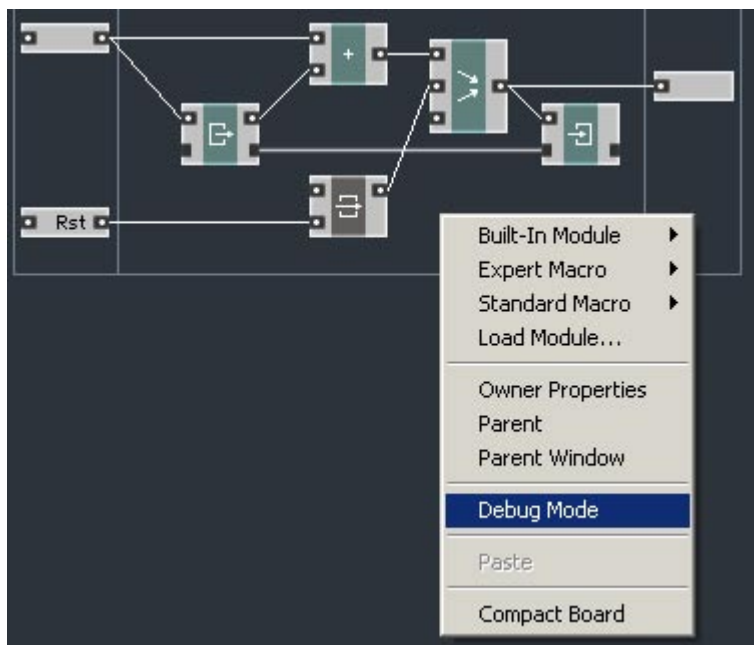



パネル表示に切り替え、1秒ごとに値が増えていくこと、ボタンを押すと0にリセットされることを確認してください。



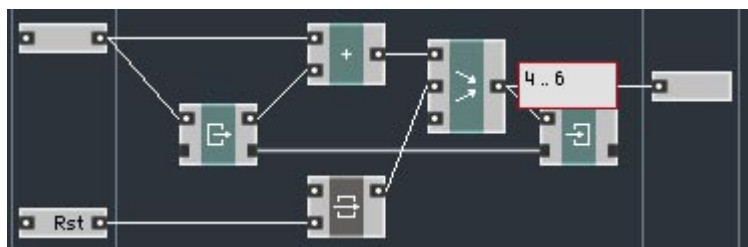
ちょうどよい機会なので、Reaktor Core のデバッグ・モードについても紹介しておきましょう。コア・レベルでは基本レベルとは違い、マウス・カーソルを出力ポート上に置いたままにしても、出力値は表示されません。内部処理を最適化した副作用で、ストラクチャーの外部から値を見ることはできなくなったのです。


しかしこれでは動作確認に支障をきたすかも知れません。そこで、最適化を一時的に解除することにより、値を確認できるようになっています。今作ったばかりのストラクチャーで試してみましょう。背景部分を右クリックし、**Debug Mode** をオンにしてください。

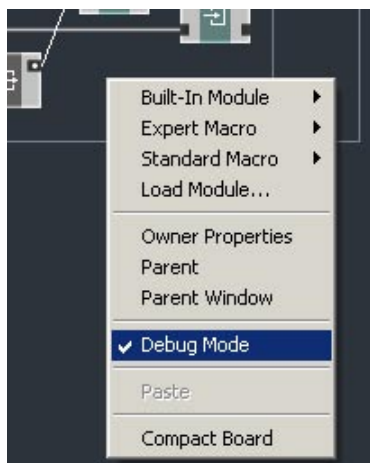


なお、ツールバーの  ボタンでもデバッグ・モードに切り替えることができます。

出力ポート上にマウス・カーソルを置いてしばらく待っていると、値 (またはその範囲) が表示されるようになります。

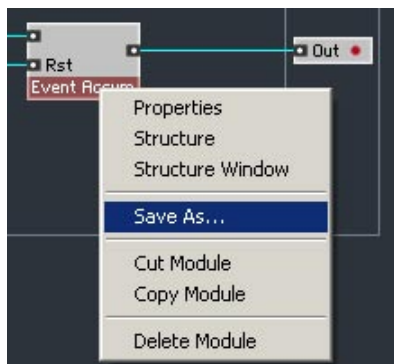


もう一度メニューを開いて **Debug Mode** をオフにするか、 ボタンを押すと、デバッグ・モードが解除されます。

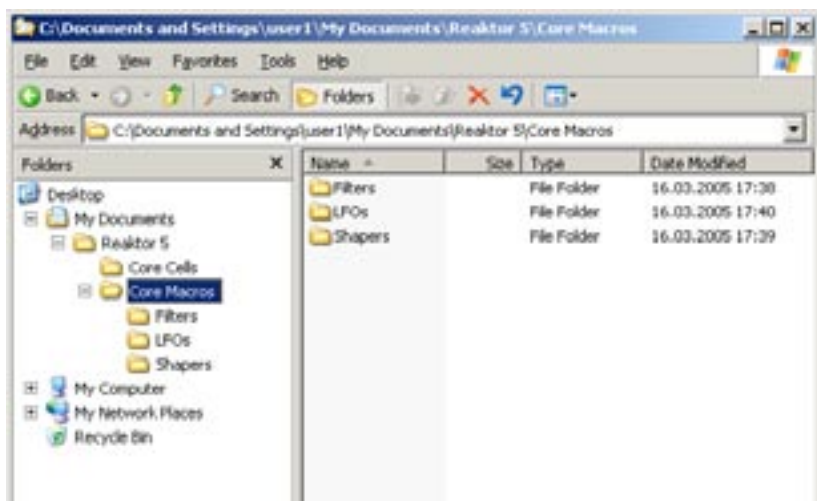


なお、ストラクチャー画面を切り替えれば自動的にオフになるので、別のストラクチャーの動作を調べる際はもう一度デバッグ・モードに切り替える必要があります。

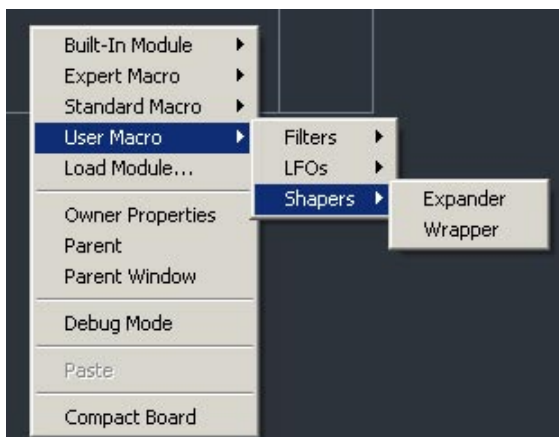
動作確認の済んだコア・マクロは、あとで再利用できるよう、独立したファイルに保存しておくといでしょう。マクロ上を右クリックし、**Save As...** コマンドを実行してください。



コア・セルと同様に、独自に作ったマクロがメニューに現れるようにすることもできます。ユーザー・ライブラリーの「Core Macros」というサブフォルダー以下に、保存したファイルを置いてください。



すると、背景部分で右クリックしたとき、**User Macro** 以下のサブメニューに、フォルダー構成をそのまま反映する形でマクロが列挙されます。

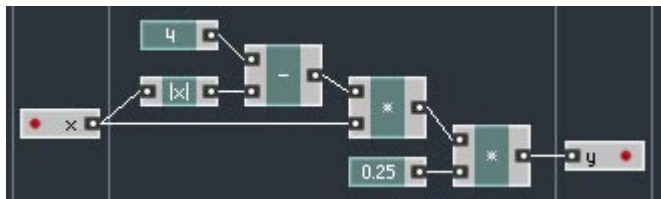


「Core Cells」フォルダーと同様、次の点に注意してください。

- ・中身がないフォルダーはメニュー上に現れません。
- ・システム・ライブラリーには決して置かず、ユーザー・ライブラリーに置いてください。

4.7. イベント・シェイパーの問題の解決

先に作成したイベント・シェイパーには問題があると述べましたが、何が悪いのか、いよいよ説明できるようになりました。



問題は初期設定イベントに関するものです。改めてストラクチャーを見てみれば、次のような点に気がつくでしょう。

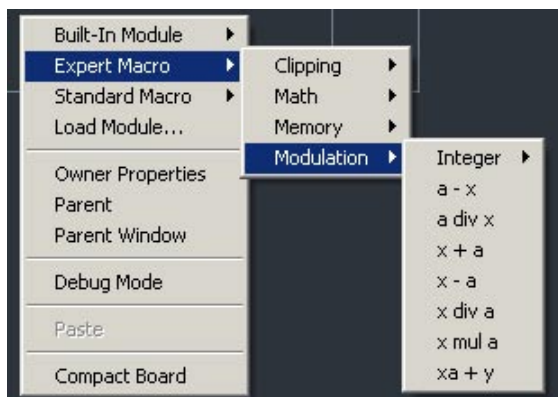
- ・ x 入力初期設定イベントを生成するためには、基本レベル上で、このストラクチャーの外部から初期設定イベントを与える必要があります。これがコア・セルのイベント入力の仕様です。
- ・ 値 4 および 0.25 の **Const** モジュールは必ず初期設定イベントを生成します。

したがって、何らかの理由で入力ポートに初期設定イベントが届かなければ、一番右の乗算器から出力ポートを介して、正しくない値が基本レベル・ストラクチャーに渡される可能性があります。

制御信号の処理であれば、初期設定イベントがなくても入力値は 0 とみなされるため、出力ポートでも初期設定イベントが発生するので問題ありません。しかしイベント処理モジュールの場合、これは期待されるような動作とは言えないでしょう。入力イベントがあったときに限り、出力イベントが生成されるようにしたいはずです。

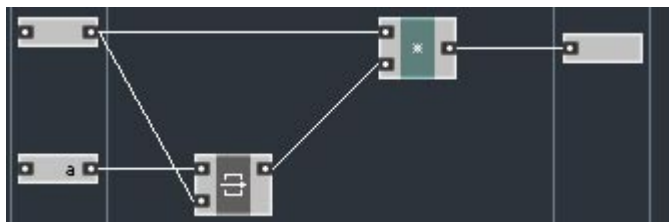
つまり問題は、2つの **Const** モジュールが原因で、入力イベントがないのに、誤ったタイミングでイベントが出力されることがある、という点です。これを解決するためには、減算器および乗算器モジュールに代えて、**Const** モジュールを適切に組み込んだ**変調マクロ**を使うようにすればよいでしょう。

変調マクロは、Reaktor Core ライブラリーの **Expert Macro > Modulation** サブメニュー以下に並んでいます。



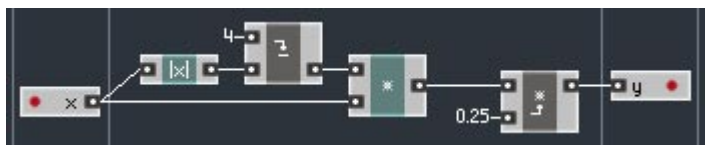
変調 (Modulation) マクロという名前は、この例ではあまり適切とは言えませんが、ある信号で別の信号を変調するために使うことが多いのでこうなっています (実際に変調のために使う方法については後述)。多くの場合、変調信号 (信号波) と被変調信号 (搬送波) の 2 つを入力とします。組み込み算術演算コア・モジュールと違って、被変調信号の入力にイベントが与えられない限り、イベントを生成、出力することはありません。変調信号の入力側にイベントが与えられても、再計算処理のトリガーにはならないのです。

このマクロの内部は非常に単純で、変調信号をラッチに入力し、被変調信号をそのクロック信号として供給しているだけのことです。この方法で構成した乗算器マクロの内部ストラクチャーは次のようになっています。



変調信号入力 (a) がラッチにつながっているので、被変調信号側の入力にイベントが与えられない限り、変調信号が乗算器に送られることはありません。

減算モジュールを **a-x** 型の変調マクロ、乗算モジュールを **x mul a** 型の変調マクロで置き換えると次のようになります。なお、**Const** モジュールも QuickConst に変更しましたが、これは本質的なことではありません。



変調マクロの変調信号入力、アイコンの矢印で識別できます。減算マクロの場合、上の入力側から矢印が出ているので、こちらが変調信号入力です。乗算マクロでは上下が逆になっています。また、変調マクロの出力ポートが、被変調信号入力側に寄っている点にも注意してください。なお、マクロ本体や入力ポート上にマウス・カーソルを置くと、これに関する情報が表示されます。

変調マクロで置き換えた結果、コア・セルの入力にイベントが与えられない限り、イベントが出力されることはなくなりました。

- ・絶対値 (|x|) モジュールは、コア・セルの入力ポートに与えられたイベントにより、直接トリガーがかかります。
- ・減算マクロは、絶対値モジュールからの出力があった時点でのみトリガーがかかります。QuickConst は、トリガーに関しては何の働きもありません。
- ・左側の乗算器モジュールは、減算器マクロまたはコア・セルの入力ポートからイベントが与えられればトリガーがかかります。実際には、既に見たように、この 2 つが同時に発生します。
- ・右側の乗算器マクロは、左側の乗算器モジュールから与えられたイベントによりトリガーがかかります。QuickConst は、トリガーに関しては何の働きもありません。

このように、直感に合う振る舞いのストラクチャーに改善することができました。

5. コアにおけるオーディオ処理

5.1. オーディオ信号

Reaktor Core では、オーディオ信号と言っても何も特別なことはありません。ストラクチャー上、他と変わるところのない普通のイベントとして扱われます。違いと言えば、オーディオ信号経路上にイベントが一定の時間間隔で流れ、その間隔がサンプル・レートで決まるという点です。

一定間隔でイベントを生成するためには、オーディオ・イベントに限った話ではありませんが、何らかのイベント源が必要です。イベント型コア・セルと同様、オーディオ型コア・セルでも、モジュールに対する入力はイベント源となります。しかしオーディオ型の場合、これ以外の種類の入力もあります。

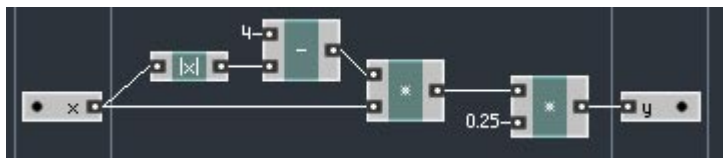
オーディオ入力は、コア・イベントを一定時間ごとに生成し、ストラクチャー内部に送ります。その間隔はサンプル・レートで決まりますが、これはコア・セル外部、基本レベルのストラクチャーで設定するようになっています。あるコア・セルに複数のオーディオ入力がある場合、各イベントは同時に送られます。

また、オーディオ入力には、初期設定イベントをコア・セルのストラクチャーに送る働きもあります。このイベントは、基本レベル・ストラクチャー上の動作とは無関係に送られます。もっとも、初期設定の際に送られる値は、コア・セル外の初期設定処理に応じて決まります。

出力については、イベント出力ではなく**オーディオ出力**になります。

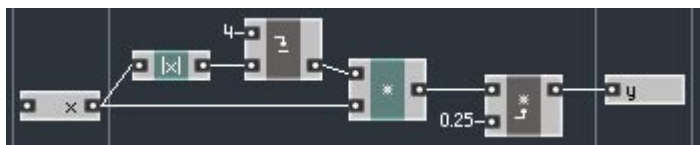
これは、コア・ストラクチャー内の処理によって定まった最終的な値を、コア・セル外部、基本レベルのストラクチャーに対して送り出す働きがあります。なお、基本レベルのストラクチャーに埋め込んだコア・セルのオーディオ出力から、イベントが送出されることはありません。

先にイベント信号用に作成したシェイパーと同様の機能を、今度はオーディオ信号用として作成してみましょう。当然、オーディオ型のコア・セルを作成することになります。ストラクチャーにはほとんど違いはありませんが、イベント入力 / 出力の代わりにオーディオ入力 / 出力を使います。

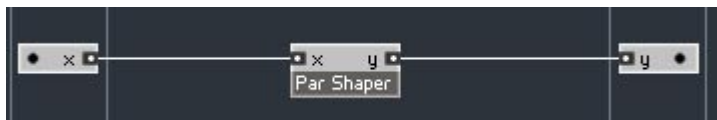


変調マクロを使っていないのはどうしてだろうと思われるかも知れません。これは、扱うのがオーディオ信号であり、これは必ず初期設定イベントを送るので、変調マクロにしなくても安全だからです。もちろん変調マクロに代えても問題が生じることはありません。

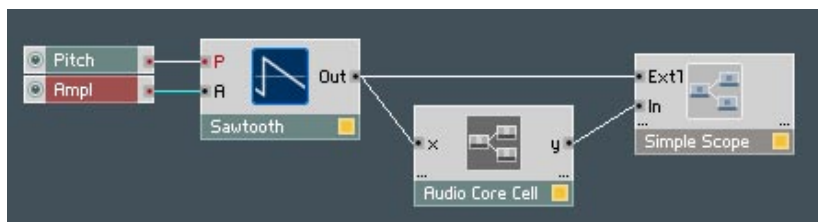
さらに、上のストラクチャーをひとつのマクロに仕立てておけば、もっと大きな Reaktor Core ストラクチャーに埋め込む形で、オーディオ処理用としてもイベント処理用としても使うことができます。この場合は、どちらの信号を処理するのであっても問題が生じないよう、変調マクロを使う方がよいでしょう。



変調マクロを使った場合のストラクチャーはこのようになります。



動作確認のため、鋸波発振器とオシロスコープを接続してみましょう。オシロスコープは、(基本レベルのストラクチャー上で、) **Macro > Classic Modular > 00 Classic Modular - Display > Simple Scope** で追加してください。さらに、インストールメントの声部数を忘れずに 1 に設定してください。



オシロスコープには外部トリガーを与えて、ディストーション・レベルを上げても適切に同期するようにします。そのためにはオシロスコープの制御パネルで **Ext** ボタンをオンにしておく必要があります。**Ampl** ノブを 0 ～ 5 程度の範囲で調整し、シェイパーの動作がはっきりわかるようにしてください。



5.2. サンプル・レート・クロック用のバス

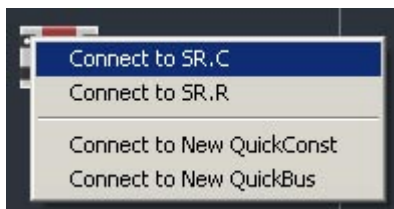
オーディオ・ストラクチャーを構築するには、ほかにも必要な機能があります。まず、オーディオ入力なしのオーディオ型コア・セルを作成する機能です。これはオーディオ・イベント源として使うのが目的です。次に、サンプル・レートがどのように設定されているかを参照できなければなりません。DSP のアルゴリズムを実装しようとすれば、ほとんどの場合これが必要になります。もちろん Reaktor Core には、どちらの機能も組み込まれています。

これはサンプル・レート・クロック用の特殊なバス接続で実現します。このバスは次の 2 種類の信号を運びます。

Clock は、サンプル・レートに合わせて、一定時間間隔のイベントを送出する信号源です。他のオーディオ信号と同様、これにも初期設定イベントがあります。値はすべて 0 になっていますが、将来この仕様が変わる可能性があるので、ストラクチャー側ではこの値を無視するようにしてください。

Rate はサンプル・レートの値 (Hz 単位) を通知する信号源です。初期設定の際のほか、サンプル・レートが変わったときにもイベントが送出されます。

このバスを追加するには、該当する信号入力ポート上で右クリックし、**Clock** ならば **Connect to SR.C**、**Rate** ならば **Connect to SR.R** コマンドを実行してください。



すると入力ポートのそばに、次のように表示されます。



このバスは、イベント型クロック・セル内では動作しません。

5.3. 信号の帰還経路

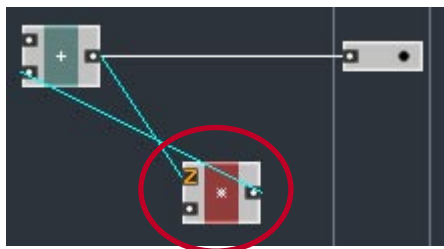
先に説明したように、信号が帰還する経路 (ループ) がストラクチャー内にあると、「上流から下流」という処理順序の規則が適用できません。このような場合は別の規則が必要です。

原則として、Reaktor Core のストラクチャーでは、帰還信号を扱うことができません。

信号が帰還する経路を結ぶことは可能です。しかし Reaktor Core の処理エンジンは、これを適切に処理することができないのです。そこで、(画面上には現れませんが、) 内部的にストラクチャーを修正し、信号の帰還が生じないようにしています。

なぜ適切に処理できないかというと、デジタル信号処理の場合、遅延を起こすことなく信号を帰還させることは原理的に不可能だからです。最低でも 1 サンプル分の遅延は避けられません。そこで処理エンジンは、帰還経路上に **Z⁻¹** というディレイ・モジュールを挿入して解決を試みます。

内部的に **Z⁻¹** モジュールが挿入されると、「Z」という橙色の文字が該当するポート上に表示されます。



この「Z」表示は以前、**Read/Write** モジュールを使ったストラクチャーに出てきました。上の図のストラクチャーは、帰還が生じないように、内部的には次のように修正したものと扱われます。



このように、最初に書き出し、次に読み込む、という順序で処理されます。**Read** モジュールは、オーディオ・ティックごとに 1 回ずつ読み込むよう、サンプル・レート・クロック (SR.C) に同期して動作します。つまり、**Write** モジュールが書き出した値を、常に 1 クロック分遅れて読み込んでいることになるのです。こうすると帰還は生じません。右から左に戻っている信号経路があるように見えますが、接続状態を変えずにモジュールの位置を動かすと次のようになります。



このように、帰還は生じていません。

このように、**Z⁻¹** モジュールを挿入することにより、帰還経路を解消できました。1 サンプル分の遅延は生じますが、論理的な動作は変わっていません。

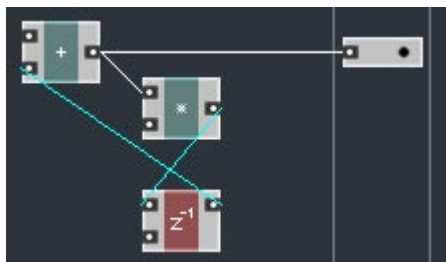
Z⁻¹ マクロの中身は、実際にはもう少し複雑です。これについては次の節で説明します。

Z⁻¹ モジュールの挿入は自動的に処理され、挿入位置などを細かく指定することはできません。具体的な位置は Reaktor Core のバージョンによって異なる可能性があるほか、ストラクチャー上の離れた箇所を変更したり、単に読み込み直しただけでも変わってしまうことがあります。

したがって、**Z⁻¹** モジュールの位置によって結果が異なるようなストラクチャーの場合は、自動挿入の機能に依存しないでください。DSP に不慣れで問題をよく理解していないと、このようなストラクチャーを作ってしまうことがあります。しかしほとんどの場合、自動挿入機能により、適切な結果が得られるはずです。

Z⁻¹ モジュールの挿入位置を細かく指定したい場合は、該当箇所に実際に挿入してください。信号の帰還が生じないので、自動挿入の機能が働くことはありません。

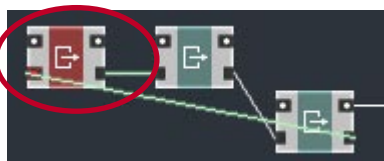
Z⁻¹ マクロを明示的に挿入すると次のようになります。これは **Expert Macro > Memory** サブメニュー以下にあります。



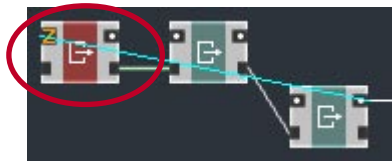
「Z」という橙色の印もなくなっていることが分かります。また、1 サンプル分の遅延が生じる位置も、自動挿入によるものとは違います。乗算器から加算器に向かう経路上で生じていますが、自動挿入によるものでは、加算器から乗算器に向かう経路上でした。

Z^{-1} モジュールにはもうひとつ入力ポートがありますが、これについては後述します。通常は何も接続しないで置いて構いません。

オブジェクト・バス接続など、信号をやり取りするのではない接続については、これを使って帰還経路を作る意味がないので、禁止になっています。何かの拍子にできてしまっても、該当するポート上に「X」型の赤い印がつき、この経路はないものとして扱われます。



これに対し、実際に信号が伝播するのであれば、それ以外の用途と兼用する経路であっても禁止にはなりません。経路上のある位置に、 **Z^{-1}** モジュールを自動挿入する形で問題が解消されます。



信号をやり取りすることのない接続が、**Z⁺-1** モジュールの自動挿入により影響を受けることはありません。もっとも、信号をまったくやり取りすることのない帰還経路を設けても無意味です。

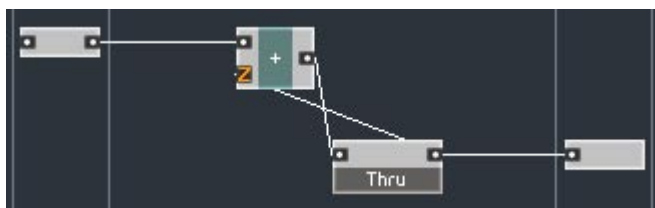
5.4. マクロが絡む帰還経路

帰還経路の解消に関しては、マクロも組み込みモジュールと同様に扱われます。

入ってきた信号をそのまま出力するだけのマクロ **Thru** を考えてみましょう。その内部ストラクチャーは次のようになります。



このマクロを使って、帰還経路のあるストラクチャーを作ってみます。



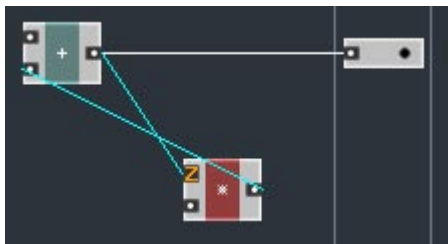
2本の接続路で帰還ループが構成されていますが、実際にはマクロ内部にも接続路が1本あります。この場合、帰還の解消はどの位置で行われるのでしょうか。図を見ると加算器の入力ポートであるようですが、条件によってはほかの位置になることもあります。

仮に **Thru** がマクロではなく組み込みモジュールであったとすれば、この内部で帰還が解消されることはなく、必ずモジュール外で起こります。

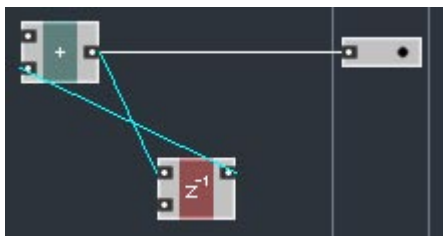
Reaktor Core の設計にあたっては、マクロであってもその振る舞いが、組み込みモジュールとできるだけ同じになるようにしたいと考えました。したがって、特に指定しなければ、マクロ外で解消されるようになっています。具体的にどの位置と確定はしませんが、マクロ内部ということはありません。

一般に帰還の解消は、ループを含む最上位のストラクチャー・レベル上で起こります。

必要であれば、以上のような振る舞いを変更し、マクロ内で起こるようにすることも可能です。実際、**Z⁻¹** もマクロですから、マクロが組み込みモジュールと同じように振る舞うのであれば、どうやって帰還を解消しているのか疑問に思うかも知れません。次のようなストラクチャーを考えてみましょう。

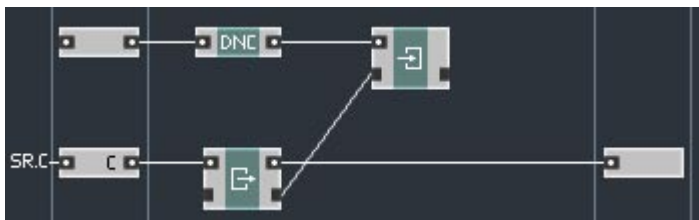


マクロも組み込みモジュールも振る舞いが同じなのであれば、乗算器を **Z⁻¹** マクロに置き換えても、何も変わらないはずです。



しかし実際には、「Z」という橙色の印がなくなっています。これは **Z⁻¹** マクロに特別な設定がしているためです。

マクロ内を見てみると、先に **Z⁻¹** の機能を実装してみたストラクチャーとほとんど同じです。

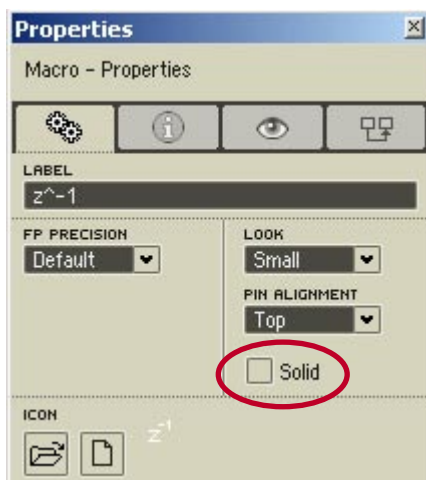


図のように、クロック入力 (SR.C) が、マクロ内部の **Read** モジュールにつながっています。この入力ポートには、値 0 の **Const** モジュール

ルではなく、オーディオ・クロックがつながっています。実際、この状態で使うのがほとんどでしょう。上側の入力ポートと **Write** モジュールの間にもモジュールがはさまっていますが、あとで解説するので当面は無視して考えて構いません。

ここまでの説明では、**Z⁻¹** ストラクチャーを実装したものである、という点を除き、このマクロに特別なことは見当たりません。しかしそうすると、このストラクチャーが帰還ループを解消するためのものであることを、Reaktor Core エンジンはどうやって認識するのでしょうか。ループを解消できることは認識できても、作成者の意図までは分からないはずです。

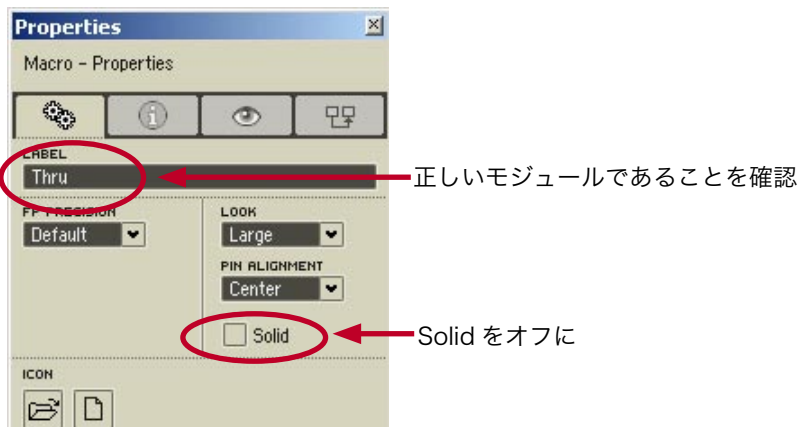
実はこの意図を伝えるためにあるのが、**Solid** というパラメーターなのです。



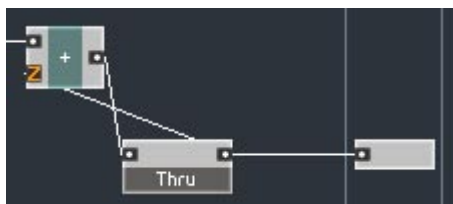
Reaktor Core エンジンは、帰還の解消に関して、マクロを組み込みモジュールのようにひとかたまりのものとして扱うか、それとも内部と外部の境界がないものとして扱うかを、このパラメーターで判断します。特別な理由がない限り、オンのままにしておいてください。マクロの内部で帰還を解消しなければならないことは、減多にないはずです。

その理由のひとつに、マクロ内で解消されても中を覗かない限り目に見えないので、遅延が生じても原因が分かりにくい、ということがあります。例えば先の **Thru** マクロで、**Solid** をオフにしてみましょう（こ

の際、別のモジュールのプロパティを変えてしまわないよう、**LABEL** 欄が「Thru」となっていることを確認してください)。



ストラクチャーを見ても、次のように、何も変わっていないかも知れません。自動挿入機能の振る舞いは一定しないので、実際にやってみると図の通りにならない可能性もあります。



しかしストラクチャーを少し変え、出力を別のモジュールに接続すると、次のようになるかも知れないのです。



このように、「Z」という橙色の印が消えています。もっと大規模で複雑なストラクチャーになると、実際には帰還の解消が行われていること、したがって若干の遅延が生じることなど、簡単に忘れてしまうものです。もちろん実際には、**Thru** マクロの内部で起こっているのです。



Solid プロパティをオンのままにするよう推奨するもうひとつの理由は、オフにすると、帰還経路上に置かれたとき、マクロ内部の処理が変わってしまうことがあることです。したがって、帰還の解消を目的としたマクロを除き、オフにはしないようお勧めします。該当するマクロはごく少数に限られるはずです。

話を **Z^{h-1}** モジュールに戻すと、このマクロでは **Solid** プロパティがオフになっているので、帰還の解消に関しては内部と外部の違いがありません。組み込みモジュールと同様に扱われることはなく、先に解説したように帰還を解消できるわけです。

5.5. 非正規数

これまでも出てきたように、ストラクチャーでは信号の値を処理できるわけですが、コンピューター上ではこの値を、単精度の浮動小数点数として表すようになっていきます。これは広い範囲の数値を表現するのに適しています。

浮動小数点数といっても、具体的な表現方法がきちんと定義されているわけではありません。表現方を指すだけの言葉なので、細かな実装を見るとさまざまな流儀があります。

最近のパーソナル・コンピューターに搭載されている CPU は、IEEE 標準に準拠した形で浮動小数点数を扱うようになっていきます。IEEE 標準では、数値の具体的な表現方法や演算結果、丸め誤差の扱い方などについて細かく定めています。さらに、精度の制限のため普通の方法では表現できない、絶対値がごく小さな値を扱うための特別な表現形式も定義されています。これを IEEE 非正規数 (denormal value) と言います。

32 ビット浮動小数点数の場合、絶対値がおおよそ $10^{-38} \sim 10^{-45}$ の範囲の数 (正負とも) が、非正規数として表現されます。絶対値が 10^{-45} より小さな値は表現できないので、0 として扱います。

非正規数の表現方法は普通の数値と異なるため、CPU の種類によっては、演算処理の上で問題が生じます。普通の数値の演算に比べて 10 倍以上遅くなる場合もあります。

非正規数がある程度以上長い時間にわたって現れる状況としては、エンベロープ関連や帰還が生じるストラクチャーなど、指数函数的に減衰する値を扱う場合が考えられます。入力信号が 0 になったあと、出力信号レベルが徐々に 0 に近づいていきます。0 になることはないけれども、いくらでも 0 に近づいていくという意味で、「漸近していく」という言い回しをします。この場合、(絶対値が 10^{-45} 以下になるまでの) かなり長い間にわたって非正規数が現れ、CPU に対する負荷も高くなりえます。

もうひとつ考えられる状況として、浮動小数点数を 64 ビットの高精度から 32 ビットに落とした場合があります。例えば 10^{-41} という数値は、64 ビットで表す場合は非正規数でないのですが、これを 32 ビットに変換すると非正規数になります (精度の変換については後述)。

例として、アナログ式で 1 ポールのロー・パス・フィルターを考え、カットオフ周波数を 20Hz と想定しましょう。ディジタル信号の値は、アナログ信号の電圧 (V 単位) に対応します。十分に長い時間にわたり、入力信号レベルが 1V を保っていたとします。するとフィルター出力の電圧も 1V です。ここで突然入力電圧を 0 にすると、出力電圧は次の式に従って徐々に減っていきます。

$$V_{out} = V_0 e^{-2\pi f_c t}$$

ここで f_c はカットオフ周波数 (Hz 単位)、 t は時間 (秒単位)、 V_0 は初期状態における電圧 (1V) を表します。

すると出力電圧は次のように変化します。

$$0.5 \text{ 秒後: } V_{out} = 10^{-29} \text{V}$$

$$0.6 \text{ 秒後: } V_{out} = 10^{-33} \text{V}$$

$$0.7 \text{ 秒後: } V_{out} = 10^{-38} \text{V}$$

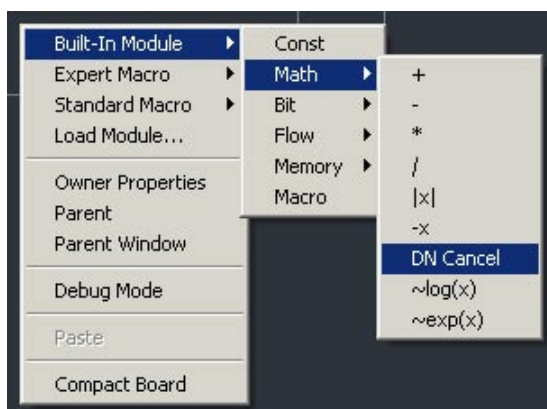
$$0.8 \text{ 秒後: } V_{out} = 10^{-44} \text{V}$$

10^{-38} から 10^{-45} までは非正規数の範囲なので、0.7 ～ 0.8 秒の間は電圧が非正規数で表されていることになります。これはフィルター内部だけの問題ではありません。フィルター出力は下流のストラクチャーで処理の対象となるので、ほかのモジュールでも非正規数を扱うことになります。

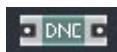
サンプル・レートが 44.1kHz とすれば、0.1 秒間が 4410 サンプルに相当します。一般的な ASIO のバッファー長は数百サンプル程度なので、CPU に対する負荷が高い状態で、バッファーを数回転させる必要があります。100% 近い負荷がかかるとすれば、オーディオ信号が脱落しても不思議ではありません。

このように、非正規数が現れるとオーディオ信号のリアル・タイム処理に悪影響があります。

基本レベルのモジュールは一般に、内部的に非正規数が現れないようなやり方でプログラムされています。もちろん DSP アルゴリズムも、決して非正規数が現れることのないよう修正済みです。Reaktor Core で独自の低レベル DSP ストラクチャーを設計する場合は、非正規数の扱いに注意が必要です。その対策として、**Denormal Cancel** モジュールを用意しました。これは **Built-In Module > Math > DN Cancel** で作成できます。



Denormal Cancel は 1 入力 1 出力のモジュールで、入力された値を若干修正し、出力に非正規数が現れないようにします。



具体的な修正方法は仕様として決めておらず、Reaktor Core のバージョン、あるいはストラクチャー内の位置によっても違ってくる可能性があります。現在は微小な定数を加えることによって非正規数を回避し

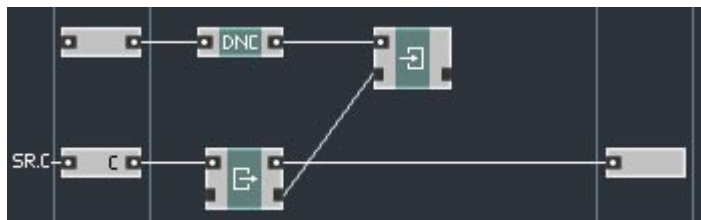
ています。定数を加えると言っても、精度の関係で、ある程度以上の数 (10^{-10} 程度より大きな数) であれば実際には値が変わりません。同じ理由で、加算してもやはり非正規数である、という可能性もほとんどありません。

何らかの理由で、作成したストラクチャー上で **Denormal Cancel** モジュールがうまく働かない場合は、非正規数を回避する別の手段を試みたくなるでしょう。しかし問題は、あるプラットフォーム上ではうまく回避できても、別のプラットフォームでは想定通りに機能しない場合があることです。一方、組み込みの **Denormal Cancel** モジュールは、プラットフォームを問わず動作するよう今後も改善していく予定ですから、どうしても不可能でない限りこのモジュールを使うようお勧めします。別のアルゴリズムを取り入れることも含め、モジュールの改善に努めていますので、提案などがありましたら NI 社サポート・フォーラムにお寄せください。

CPU によっては、非正規数を強制的に 0 にするよう設定できる場合があります (IEEE 標準には非準拠)。Reaktor Core はプラットフォーム非依存を目指しているので、このような CPU を使っている場合であっても、ストラクチャー設計の際は非正規数に配慮するよう強くお勧めします。

非正規数が現れうる典型的な状況としては、指数函数的に減衰していく帰還ループがあります。逆にオーディオ処理中に帰還ループが現れると、信号は指数函数的に減衰していきます。フィルター処理や、ディレイ信号が入力側に帰還するようなストラクチャーがこれに当たります。こういった事情を考慮して、標準の **Z⁻¹** マクロに非正規数を解消する機能を組み込みました。

前にも示したように、このマクロの内部は次のようになっています。



以上で **Denormal Cancel** モジュールの役割が理解できたことと思います。**Z⁻¹** マクロを使う機会が多い帰還ループでは、何も対策しないと非正規数が現れやすいので、最初からマクロ内にこれを解消する機能を組み込んでおいたわけです。

これとは別に、非正規数の解消機能が組み込まれていない、**Z⁻¹ ndc** というマクロもあります。「ndc」は「No Denormal Cancel」を表します。有限インパルス応答 (Finite Impulse Response、FIR) フィルターなど、非正規数が決して現れないと分かっているストラクチャーでは、こちらを使っても構いません。



5.6. その他の特別な値

Reaktor Core ストラクチャー (特に帰還ループ) の内部状態として現れると問題になる値としてはほかに、INF、NaN、QNaN があります。こういった値そのものについてはインターネット上などに解説記事が多数あるので深入りせず、ストラクチャー内に現れないようにする方法のみ解説します。

こういった値は多くの場合、不正な演算の結果として現れます。分かりやすい例として、0 で除算した場合があります。また、浮動小数点数として表現できる最大数 (単精度の場合およそ 10 の 38 乗のオーダー) を超えた、あるいは演算に定義域外の値が与えられた場合にも現れます。

こういった値は、いったん現れるとそれ以降の演算でも解消されない傾向があるので、非正規数よりも扱いが面倒です。非正規数はそうでない数を加算するとほとんどの場合解消できますが、例えば INF が現れると、何らかの数を加算しても結果は INF のままです。

このように、ストラクチャー全体をリセットしなければ解消できないばかりでなく、CPU 処理時間が増える点も問題です。したがって、除算の際は除数が 0 でないことを確認するなど、ストラクチャー設計にあたって細心の注意が必要です。初期設定については特に注意してください。例えばストラクチャーの一部に次のようなモジュールがあるとしましよう。



除算モジュールの下側の入力（除数）に、何らかの理由で初期設定イベントが届かなかった場合、初期設定の際に 0 除算が起こります。この場合、代わりに変調ディレイ・マクロを使うなど、目的に応じた対策が必要です。

5.7. 1 ポール・ロー・パス・フィルターの構築

単純な 1 ポール・ロー・パス・フィルターは、次の再帰的な計算式を実装することにより実現できます。

$$y = b * x + (1 - b) * y_{-1}$$

ここで、

x は入力サンプル、

y は出力サンプル、

y_{-1} は直前の出力サンプル、

b はカットオフ周波数を決定する係数。

係数 b の値は、カットオフ角振動数を正規化したもので、次の式で求められます。

$$F_c = 2 * \pi * f_c / f_{SR}$$

ここで、

f_c はカットオフ周波数 (Hz 単位)、

f_{SR} はサンプル・レート (Hz 単位)、

π は円周率 (3.14159...),

F_c は正規化されたカットオフ角振動数 (ラジアン単位)。

実際には、係数 b と正規化されたカットオフ角振動数 F_c は近似的に等しいだけで、カットオフ周波数が高くなると誤差が大きくなります。しかし精度をそれ程必要としないのであればこれで充分です。

まず、2 入力のオーディオ型コア・セルを作成します。入力オーディオ信号用とカットオフ周波数の設定用です。ここではカットオフ周波数を、イベント入力として扱います。



流用しやすいよう、コア・マクロの形で作成することにしましょう。そこでストラクチャー内にマクロを追加し、上記と同じ入力ポートを用意します。



カットオフ周波数を正規化されたカットオフ角振動数に変換する回路を組み込みます。



定数6.28319は 2π を表します。これをサンプル・レートで割り、カットオフ周波数を掛けて角振動数を求めます。乗算器として変調マクロを使う必要はありません。というのも、Fは制御信号入力なので、初期設定イベントがなくても正しく乗算できるからです。

乗算よりも除算を先に実行しているのは、除算の方がCPUに対する負荷が高いこと、サンプル・レートはそう頻繁には変わらないことを考慮したためです。カットオフ周波数が変わっただけならば、イベントが除算器に送信されず、したがって除算も実行されません。こういった最適化は、コア・ストラクチャーの設計における常套手段です。

次にフィルター処理を実装する回路を組み込みましょう。



オーディオ入力をラッチ・モジュールにつないでのは、オーディオ・クロックと非同期にイベントが届く場合があるからです。常に同期して

いると分かっている場合は必要ありませんが、コア・マクロに汎用性を持たせたいのであれば、これに備えておくべきでしょう。

2つの乗算器には変調マクロを使って、F入力にいつイベントが与えられても、帰還ループ内の処理にトリガーがかからないようにしています。この例ではカットオフ周波数に応じた信号で帰還経路上の利得を変化させているので、「変調マクロ」という用語を使う理由が分かりやすいでしょう。

イベント入力により適切でないタイミングで計算処理が始まるのを防ぐ手段として、Reaktor Core の設計においてはラッチが頻繁に使われます。同じ状況で、変調マクロもよく登場します。

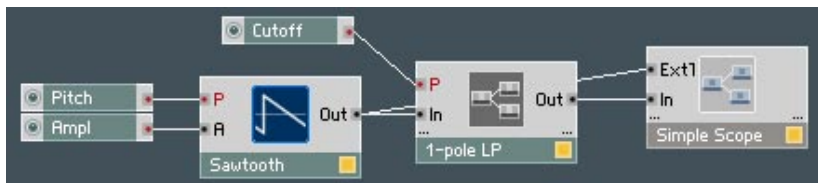
Z⁻¹ モジュールは、1 サンプル前の出力値を記憶しておくために使います。オーディオ・クロックに合わせて乗算器に送り出しています。また、非正規値を解消する役割もあります。DSP の設計に慣れている方は、このストラクチャーが標準的な DSP フィルターの回路に似ていることが分かるでしょう。

加算器の出力につながっている **Merge** モジュールは、入力信号の値が 0 でなくても、初期設定直後のフィルターの状態を確実に 0 にするために必要です。

最後にピッチを周波数に変換するモジュールを追加すれば、いよいよ実際に動作を試してみることができます。



動作確認のため、次のようなストラクチャーを使います。インストゥルメントの声部数を忘れずに 1 に設定してください。



Cutoff ノブを例えば 0 ～ 100 の範囲で調整します。カットオフ周波数を高くすると、係数の誤差が大きくなるため、フィルターの動作が不安定になるので注意してください。

実際には、動作が不安定にならないよう、カットオフ周波数の範囲を制限するしくみを組み込むべきでしょう。例えば係数 b の範囲を 0 ～ 0.99 に制限します。その具体的な手順については次の章で解説します。

制御パネルは次のようになります。



Cutoff ノブを調整し、出力信号がどう変化するか確認してください。

6. 条件処理

6.1. イベントの経路制御

イベントは常に同じ経路を伝わるわけではなく、動的に変えることも可能です。そのためには **Router** モジュールを使います。これは **Built-In Module > Flow** メニュー以下にあります。



Router モジュールは、信号入力 (下側) で受けたイベントを、出力 1 (上側) または出力 0 (下側) に送ります。どちらの出力に送るかは **Router** の状態によって決まり、これを Ctl 入力 (上側) で制御できるようになっています。

Ctl 入力に与える信号は BoolCtl というもので、これまでに出てきた種類の信号とは異なり、もちろんオブジェクト・バス接続とも違います。真 / 偽 (あるいはオン / オフ、1/0) の 2 つのうちいずれかの値を取り、これが真であればイベントは出力 1 に、偽であれば出力 2 に送られます。

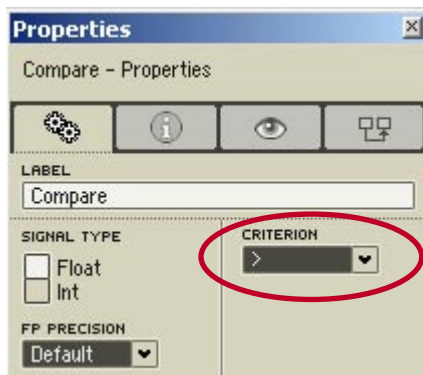
Reaktor Core の制御信号は、一般の信号と大きく異なり、イベントを運ぶことがないため、それ自身が何らかの処理を起動することはありません。

Router の制御用に制御信号源が必要ですが、多くの場合、**Built-In Module > Flow > Compare** メニュー以下にある比較モジュールを使います。



このモジュールは 2 つの入力信号を比較し、その結果を BoolCtl 信号として出力します。上側の入力と比較演算子の左辺、下側の入力がある右辺に当たります。したがって「>」モジュールの場合、上側の値が大きい場合に真の制御信号を出力することになります。

比較方法は **CRITERION** プロパティで設定します。

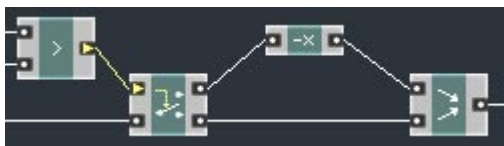


選択肢は次の通りです。

=	等しい
!=	等しくない (≠)
<=	より小さい、または等しい (≤)
<	より小さい
>=	より大きい、または等しい (≥)
>	より大きい

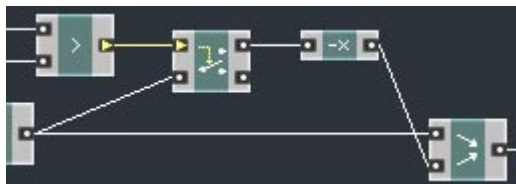
もちろんいくつかの **Router** モジュールを組み合わせることも可能です。それぞれの状態は同時に変わります。

Router モジュールを使うと、イベントの経路を枝分かれさせることができます。もっとも、多くの場合、最終的にそれぞれの経路をマージすることになります。



上のストラクチャーは、比較結果に応じ、信号を反転し、またはそのままで出力します。

同じ機能を次のストラクチャーでも実現できます。



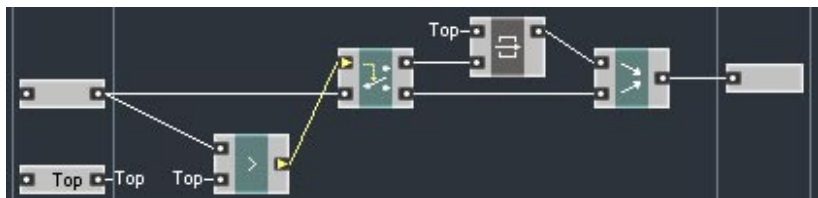
このストラクチャーの **Router** モジュールは、条件が偽の場合の出力には何もつながっていません。したがってゲートとして働き、条件が真の場合のみイベントが通過することになります。反転信号は **Merge** モジュールの第2 入力に入り、したがって条件が真の場合、反転していない第1 入力を打ち消してこの反転信号が出力されます。一方、偽の場合は反転信号が **Merge** モジュールに届かないので、第1 入力そのまま出力されるのです。

枝分かれしたイベントの経路をマージするには多くの場合 **Merge** モジュールを使います。しかし、加算器、乗算器などのモジュールで経路をマージすることも、理論的には可能です。

Router モジュールは初期設定イベントを他の一般のイベントと同じように扱います。したがって条件によっては、初期設定イベントが後続のモジュールに伝わらないことも起こります。

6.2. 信号クリッパーの作成

入力オーディオ信号のレベルが過大にならないよう制限 (クリップ) するマクロ・ストラクチャーを作ってみましょう。

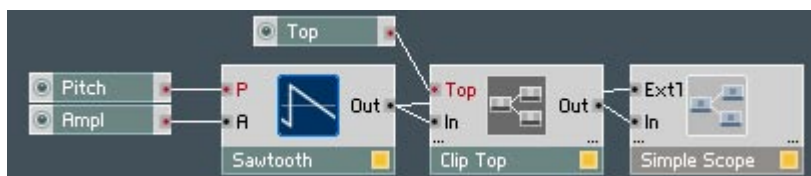


入力信号レベルが所定の閾値以下であれば、**Router** の「偽」出力から **Merge** モジュールを経由してそのまま出力されます。そうでなければ **Router** の「真」出力が **Latch** モジュールに入り、レベルがどんなに大きくても閾値に等しい信号が出力され、これが **Merge** モジュール

を経由して最終出力に到達します。初期設定の際にも同様に処理されません。

このストラクチャーは、閾値を変えてもその時点では出力が変化しません。新しい閾値に従って処理されるのは次のイベント以降です。これは、入力値が変わってもその時点ではイベントが出力されない、変調マクロの振る舞いに似ています。

クリッパーの動作を確認するため、次のようなストラクチャーを作りましょう (オーディオ型コア・セルを使用)。



制御パネルは次のようになります。



実際には同様の「変調」クリッパー・マクロが **Expert Macro > Clipping** メニュー以下にいくつか並んでいます。

6.3. 単純な鋸波発振器の作成

単純な鋸波発振器を作ってみましょう。振幅は 1 に固定、周波数は調整可能なものとします。出力信号レベルを一定の勾配で増加させ、1 を超えたら 2 を減らして -1 にする、という方針で作ります。

2 を引く代わりに -1 にリセットする方法も考えられますが、発振周波数の精度を保つのが難しいのでここでは採用しません。

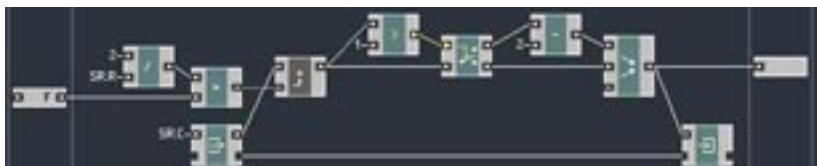
レベルを増加させる速度は、指定された周波数から次の式で計算します。

$$d = 2 f / f_{SR}$$

まず、この速度 (1 サンプル相当時間あたりの増分) を計算する回路を作ります。



次に信号レベルを増加させるループを作ります。累算器の例に出てきたのと同じ、**Read/Write** モジュールの組を使います。



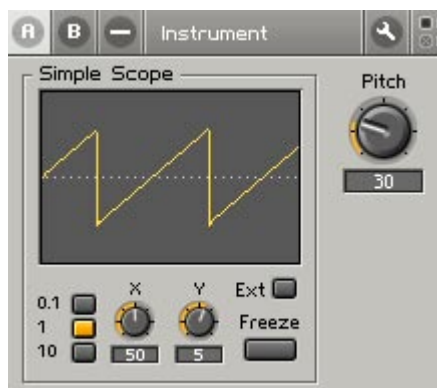
Read モジュールはオーディオ・イベントごとに、レベルを増加させる処理に対してトリガーをかけます。元のレベルに増分を加えた値を出力すると同時に、**Write** モジュールにも渡してその値を保存します。但しその間に、値を 1 と比較する処理が入っています。

Merge モジュールの入力 3 は、発振出力の信号レベルを確実に 0 に初期設定するために必要です。2 を引く働きの減算器は変調マクロにする必要がありますが、**Merge** モジュールの側に初期設定の機能を持たせているので、その必要はありません。

動作確認用のストラクチャーを示します。**P2F** 変換器がコア・セル内にあることを思い出してください。



また、制御パネルは次のようになります。



7. その他の種類の信号

7.1. 浮動小数点数値 (Float) 信号

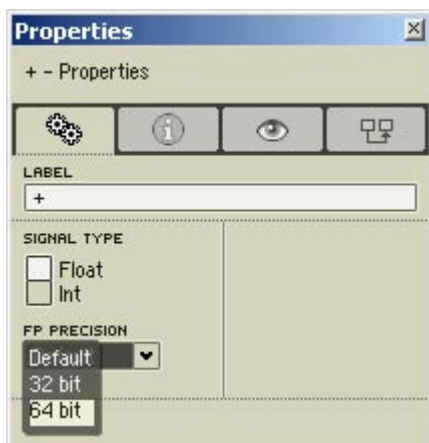
DSP(デジタル信号処理) に最も多用されるのは浮動小数点数型の信号でしょう。32 ビット単精度ならば 10^{38} 、64 ビット倍精度ならば 10^{308} のオーダーまでの、幅広い範囲の数値を表現できます。但し精度には問題があり、64 ビットならば精度が高いとはいえ、やはり限界があります。

精度に限界があるのは技術的な理由によります。無制限に精度を高めようとするれば、メモリー量も CPU に対する負荷も無限に増えてしまいます。 π などの無理数を 10 進展開すれば、小数第何位まで求めても終わることはありませんから、有限の面積しかない紙に書き下すことは不可能です。有理数であれば有限の桁で終わる (あるいは同じ並びの繰り返しになる) とはいえ、ある面積の紙には書き切れない数がいくらでもあります。

Reaktor Core では一般に、数値を 32 ビット浮動小数点数として表現します。64 ビット倍精度も使えますが、32 ビットでも 10^{38} から 10^{38} まで広い範囲の数値を扱えるので、多くの場合これで充分です。

表面上は 32 ビット浮動小数点数で表していても、内部的には 64 ビットで処理している場合があります。

この精度はモジュールごと、あるいはマクロごとに、変更することもできます。モジュールについては、**FP PRECISION** プロパティで設定します。



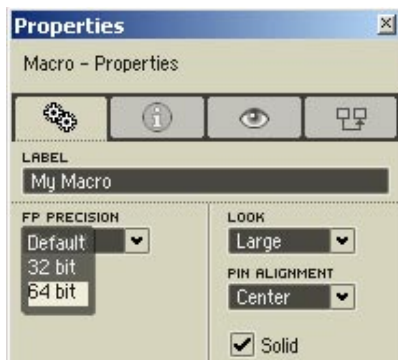
Default ストラクチャーに関して設定されたデフォルトの精度を援用

32 bit 少なくとも 32 ビットの精度で処理

64 bit 少なくとも 64 ビットの精度で処理

モジュールについて精度を変更すると、その内部処理ばかりでなく、出力値にも影響があります。

ストラクチャー全体に適用するデフォルトの精度も変更できます。背景部分を右クリックし、**Owner Properties** コマンドで次の **Properties** ウィンドウを開いてください。



この設定は、ストラクチャー内のモジュールやマクロ全体に適用されます。但し、特定のモジュールについて別に設定している、あるいはマ

クロ口についてそのマクロ口内のみ適用されるデフォルトの精度を設定している場合を除きます。

32 ビット数と 64 ビット数には互換性があり、変換して自由に入れ替え可能です (非正規数を除く)。但しオブジェクト・バス接続で、精度の異なる数を共有することはできません。32 ビット数と 64 ビット数をいっしょに記憶しておくことはできないからです。さらに、あるマクロ口に含まれるモジュールで精度を「Default」と設定し、これがたまたま 32 ビットであったとしても、明示的に「32 bit」と設定した信号との互換性はありません。マクロの側でデフォルトの精度を変えることにより、実際の精度が 64 ビットになってしまう可能性があるからです。

コア・セルの最上位ストラクチャーにあるモジュールの入出力では、常に 32 ビットの値を受け取り、または送り出します。基本レベルではイベント信号、オーディオ信号とも常に 32 ビットで扱うからです。

7.2. 整数値 (Integer) 信号

多くの CPU では、浮動小数点数よりも基本的なデータ型として、整数値を扱えるようになっていきます。精度は無限大、すなわち丸め誤差が生じる余地はありませんが、値の上限と下限はあります。32 ビット整数であれば最大値は 10^9 のオーダーです。

精度が無限大というのは、小数点以下の桁がないため、有限個の数字で必ず表現できるからです。例えば 1 時間あたりの秒数という場合、3、6、0、0 と 4 個の数字を並べれば終わりです。一方、 π の値を書き下そうとすれば、3、1、4、1 と並べてもまだ終わりません。桁を増やして 5、9 と追加していても、これで完全ということはないのです。このような意味で、整数値は精度が無限大である、と言っているに過ぎません。

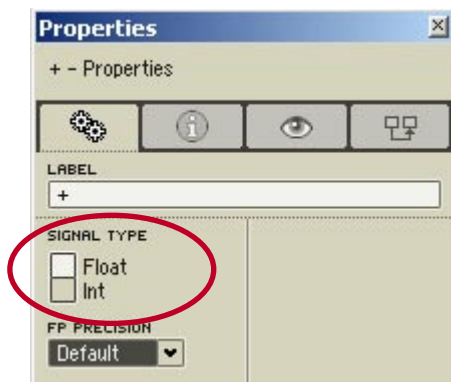
連続的に変化する値を表す場合は浮動小数点数を使うのが自然ですが、カウンタのように離散的に増減する量を扱うには整数が向いています。

Reaktor Core のモジュールの多くは、入力として整数値信号を想定する Int モードに切り替えることができます。この場合、出力も整数値信号になります。このモードに切り替えて使える代表的なモジュールと

しては、加算器、乗算器などといった算術演算モジュールがあります。もっぱら整数値を扱うモジュールも若干あります。

Reaktor Core では、整数値が少なくとも 32 ビット長であることを保証しています。

整数値モード / 浮動小数点数値モード (Int/Float) は、**SIGNAL TYPE** プロパティで切り替えます。



このプロパティを Int にすると、入力値、出力値とも整数になります。入出力ポートを見ると、次のように整数値である旨の印がついています。

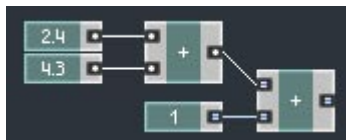


マクロの入出力信号については、このようなモード切り替えはありません。一般に、マクロの場合、ごく簡単なストラクチャーであっても、整数値を処理する場合と浮動小数点数値を処理する場合とで、具体的な実装がまったく同じということはほとんどないからです。

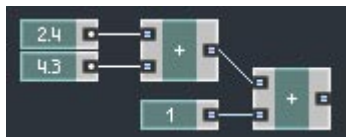
整数値信号を Float モードのモジュールに与えることも不可能ではありませんが、変換処理のため CPU にかなりの負荷がかかります。このマニュアルを執筆している時点の実装では、PC 上ではこの違いが若干気になる程度ですが、Mac の場合はかなり大きな差となって現れます。また、オブジェクト・バス接続はもちろんです。

変換により情報が損なわれる場合もあります。特に大きな整数値を浮動小数点数に変換すると精度が落ちますし、逆に浮動小数点数を整数に変換すれば小数部分が欠落します。さらに、整数値として表現できる最大数よりも大きな浮動小数点数を整数に変換することはできず、結果は未定義です。浮動小数点数から整数への変換に当たっては、最も近い整数値への丸めが起こります。但し、0.49 は 0、0.51 は 1 に丸められますが、0.5 が 0 になるか 1 になるかは不定です。

ある演算モジュールを Int モードにした場合と、Float モードのままで演算してその出力を整数に変換した場合とでは、結果が異なることがあるので注意してください。例えば次のストラクチャーで、2.4 と 4.3 を加算すると 6.7、これを整数に変換すると 7 になります。したがって最終的な出力は 8 になります。



一方、左側の加算器を Int モードにすると、2.4 と 4.3 ではなく、2 と 4 を加算することになります。その結果は 6、したがって最終的な出力は 7 になります。



クロック入力の場合は値を参照することがないので、Float モードのままであり、もちろん型変換が起こることもありません。

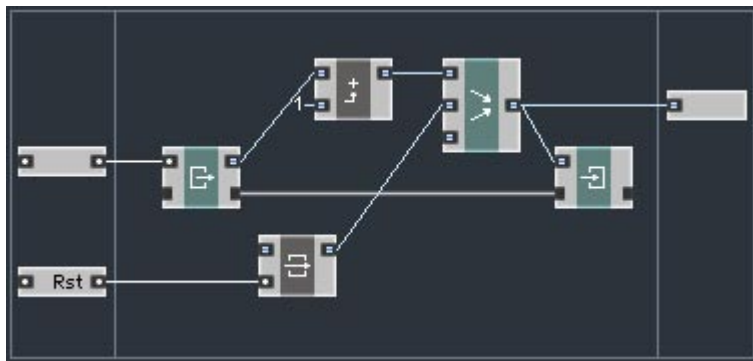


Read モジュールを Int モードにしても、クロック入力 Float モードのまま動作します。すなわち、OBC ポートは Float モードでも Int モードでも同じように動作します。

帰還の解消には、Int モードであっても Float モードの場合と同じく、**Z^{h-1}** モジュールを使います。もちろん非正規数の解消処理は不要です。

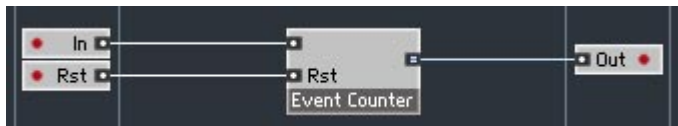
7.3. イベント・カウンターの作成

「イベント・カウンター」マクロを作成してみましょう。イベント累算器と似た構成ですが、値を加算する代わりに、イベントの個数を数えます。扱うのが個数なので整数値を使うことにします。



出力ポートおよび組み込みモジュールをすべて Int モードにします。**ILatch** マクロは個数のリセットに使います。**Latch** と同様の機能で、メニュー上も同じ所に並んでいますが、整数値信号を扱う点が違います。さらに、変調マクロも整数値用のものを使います。これは **Expert Macro > Modulation > Integer** メニュー以下にあります。入力ポートにはクロック信号を与えるだけなので、Int モードにする必要はありません。

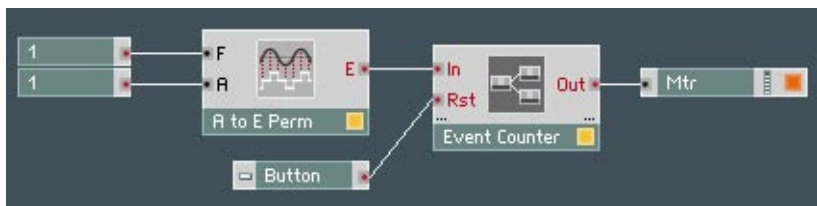
このマクロを組み込んだイベント型コア・セルのストラクチャーを示します。



このジュールの出力ポートは、Int モードにはなっていない点に注意してください(しようとしても不可能)。というのも、基本レベル上のモジュールとしてコア・セルを見たとき、その出力は基本レベルで言うと

この正常なイベント、すなわち浮動小数点数値でなければならないからです。

動作確認用のストラクチャー例を示します。



制御パネルは次のようになります。



7.4. 立ち上がり信号カウンター・マクロの作成

ここで符号比較演算について説明しましょう。これは Reaktor Core ストラクチャーを作成する上で時々必要になる技法です。2つの数値を比較する演算ですが、値は無視し、符号(正/負)にのみ着目します。当然ながら、正の方が負よりも大きいと定義します。したがって、例えば次のようになります。

3.1 は -1.4 よりも、符号比較の意味で大きい

2.1 は 5.0 と、符号比較の意味で等しい

4.5 は -2.9 と、符号比較の意味で等しくない

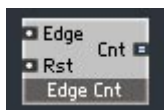
0 の符号は未定義なので、ある数と 0 を符号比較した結果は不定です。

符号比較演算は、普通の比較演算モジュールやルーターを組み合わせで実装することも可能ですが、専用の **Compare Sign** モジュールをが **Built-In Module > Flow > Compare Sign** メニュー以下にあるので、こちらを使った方がよいでしょう。



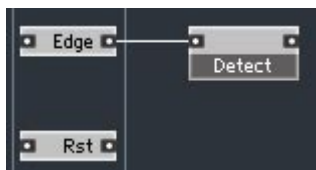
このモジュールの出力は BoolCtl 信号なので、ルーターに直結できません。

符号比較演算の応用例としては、信号の立ち上がりを検出する、という使い方があります。実際に、立ち上がりを検出し、個数を数えるマクロを作ってみましょう。

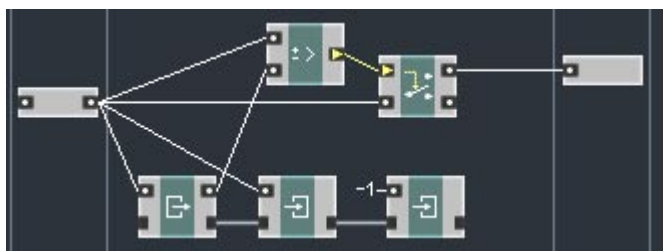


出力値は個数を表すので、整数値 (Int) モードにしています。

まず必要なのは、信号の立ち上がりに応じてイベントを出力するマクロです。



この中身は次のようになっています。

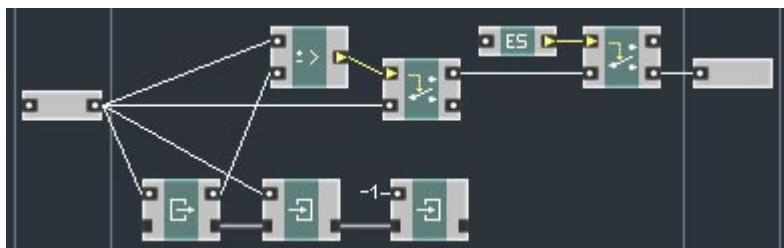


オブジェクト・バス接続を利用して直前の信号値を保存しています。符号比較用に、直前の値を読み込んだあとで、新しい値を書き出していることが分かります。右側の **Write** モジュールは、信号値を保存するメモリーの初期設定用です。-1 に初期設定しているので、正の値の信号が届いた時点で 1 回目の立ち上がりと認識することになります。

メモリーの初期設定については、**Merge** モジュールを使う方法を先に紹介しましたが、上記のような方法もあります。この場合、初期化用の **Write** モジュールをオブジェクト・バス接続の末尾に置いて、上流側の **Write** モジュールの出力にかかわらず、想定通り初期化されるようにする必要があります。

Router モジュールは、**Sign Comparison** モジュールの結果にもとづき、符号が負から正に変わったときだけイベントを通過させます。

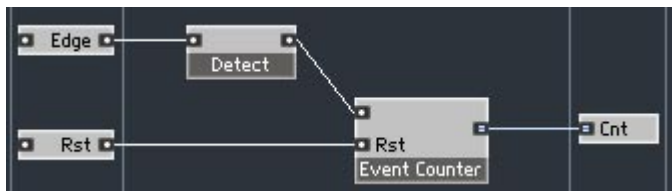
初期設定イベントを処理している時点では、メモリー上の値は 0 であり、その符号は未定義なので、イベントが送られるかどうかははっきりしません。そこで、初期設定中にイベントが送られることのないよう、ストラクチャーを次のように修正します。



新たに追加した **ES Ctl** モジュールの名前は、「Event Sensitive Control」に由来しています。入力ポートにイベントが届いた場合にのみ、真の制御信号を出力します。しかし上のストラクチャーでは入力に何もつながっていないので、定数 0 がつながっているものとして扱われ、したがって初期設定のときのみ真になります。この振る舞いを利用して、初期設定の際にイベントが発生しても、それを出力しないよう一番右側のルーターで制御しています。

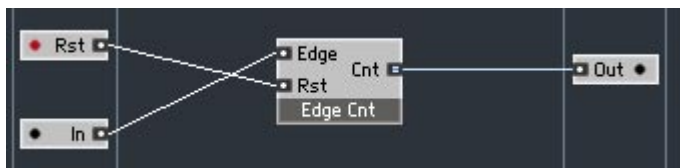
これは、初期設定中にはイベントが出力されないようにしたい場合の常套手段です。

これで信号立ち上がりを検出する **Detect** モジュールができたので、カウンターと組み合わせます。

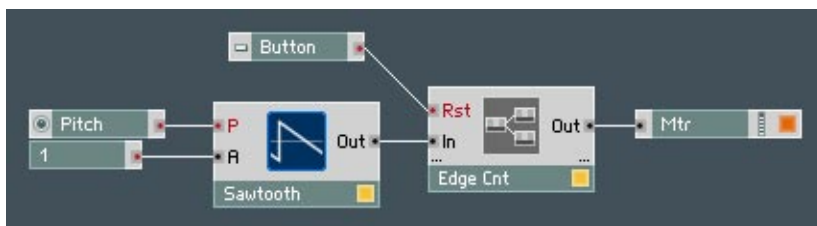


動作は単純で、**Detect** モジュールが信号立ち上がりを検出すること
にイベントを送り、それを **Event Counter** モジュールが数えるように
なっています。

動作確認のため、オーディオ型コア・セルにこのマクロを組み込み、
鋸波を与えて立ち上がりを数えてみましょう。コア・セルの内部ストラ
クチャーは次のようになります。



また、基本レベルのストラクチャーは次のようになります。



なお、**Meter** パラメーターを先の例のように設定してください。
制御パネルは次のようになっています。



メーターの数字が切り替わる速度は鋸波の周波数によって決まり、これは**Pitch** ノブで調整できます。ピッチを 0 とすれば約 8Hz になるので、1 秒に 8 回の割合で数字が増えていくはずです。

8. アレイ (Array)

8.1. 概要

4つのオーディオ信号から、制御信号にもとづいて1つを選択する、という機能のモジュールを考えてみましょう。



Router モジュールを組み合わせる方法もありますが、こういった目的には **Array** モジュールが向いています。

1次元アレイとは、同じ型の複数のデータ項目に、順序番号をつけて管理できるようにしたもののことです。例えば次の5つの浮動小数点数があるとしましょう。

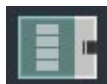
5.2 16.1 -24.0 11.9 -0.5

Reaktor Core では、アレイの各要素には0から始まる番号をつけて管理するようになっています。したがって、先頭の要素である「5.2」は、0番というインデックスで参照できます。同様に、インデックス1番には16.1、2番以降には-24.0、11.9、-0.5が順に対応します。

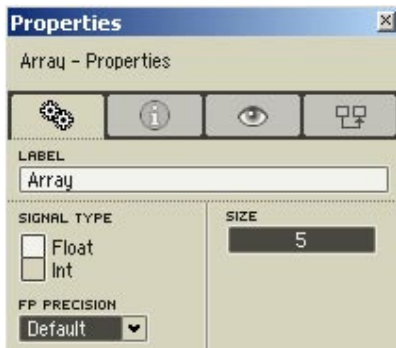
表を使って表すと次のようになります。

インデックス	0	1	2	3	4
値	5.2	16.1	-24.0	11.9	0.5

アレイの作成には **Array** モジュールを使います (**Built-In Module > Memory > Array** メニュー)。



Array モジュールには出力が1つあり、これは **Array OBC** という型になっています。アレイの大きさ(要素数)、要素データの型は、プロパティとして設定できます。



例えば上記の5つの要素データを扱う場合、**SIGNAL TYPE**としてFloat、**SIZE**として5を指定します。

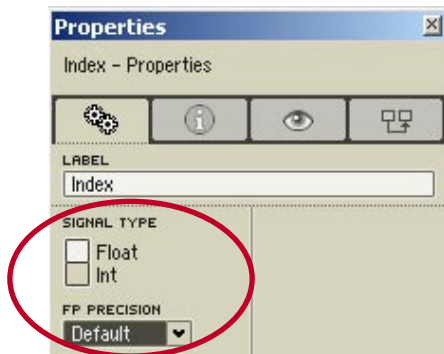
インデックスは0から始まるので、要素数が5であれば、各要素は順に0～4の番号で参照します。

Array OBC 信号は、要素データ型が異なれば互換性がありません。

Array モジュールのある要素にアクセスするためにはインデックスを指定しなければなりません。それには **Index** モジュールを使います (**Built-In Module > Memory > Index**)。



下側の入力ポートはオブジェクト・バス接続のマスター側になっており、これを **Array** モジュールの出力 (スレーブ) と接続します。この信号の型は、アレイ側の各要素の型に合わせて、**SIGNAL TYPE** プロパティで指定します。



Array モジュールとは次のように接続してください。



Index モジュールの上側の入力には、整数 (Int) 型の信号でインデックス値を与えます。例えば次のようにすると、インデックス 1 の要素が出力されます。



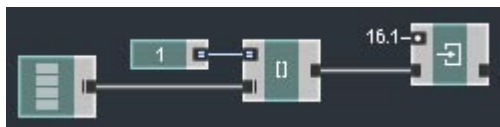
定数モジュールの出力ポートが Int 型になっている点にも注意してください。実際には Float 型のままだでも、自動的に変換されます。

次のように QuickConst を使っても構いません。



Index モジュールの出力は **Latch OBC** という型になっているので、**Read/Write** モジュールにも直接接続できます。もちろんデータ型 (Float/Int) は合わせなければなりません。

インデックス 1 の要素を 16.1 に初期設定するストラクチャー例を示します。



インデックスとしてアレイの範囲外の番号を与えた場合の結果は未定義です。ストラクチャーが壊れることはありませんが、どのような結果が得られるかは分かりません。必要であればルーターやマクロを組み合わせ、このような番号が与えられても信号がアレイに到達しないようにしてください。

8.2. オーディオ信号セクターの作成

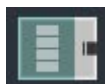
話を戻して、オーディオ信号のセクターを作成してみましょう。



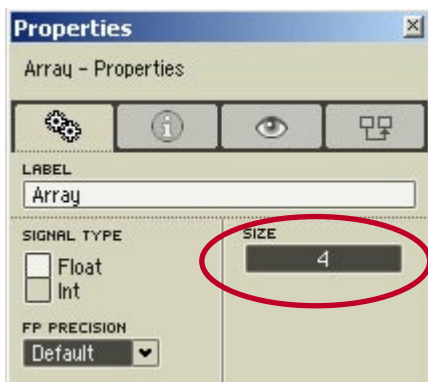
まず、新しくコア・セルを作成し、次のように入出力ポートを置いてください。



Float 型 4 要素の **Array** モジュールを使ってオーディオ信号を保持することにします。



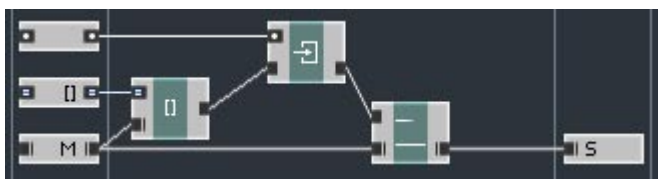
したがってプロパティは次のようになります。



入力された値をアレイに書き込むために、**Write []** マクロを使います (Expert Macro > Memory > Write []).



このマクロは **Index** モジュールと **Write** モジュールを組み合わせたもので、あるインデックスの要素に値を書き込む働きがあります。

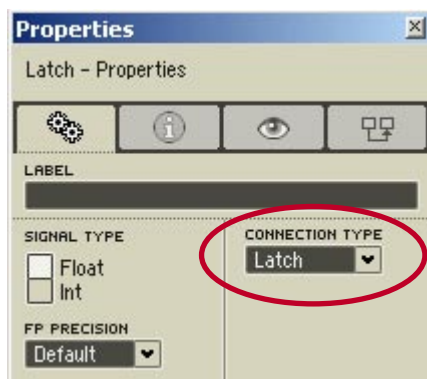


一番上の入力に書き込みたい値、「[]」入力にインデックスを与えます。M 入力には、Float 値アレイ (デフォルトの精度) をオブジェクト・バス接続でつなぎます。S 出力は、**Read/Write** などの OBC モジュールと同様のスルー接続です。

M 入力と S 出力は、これまでに紹介したものとは違う種類のマクロ・ポートです。ポートを追加する際、**New > Latch** を選択してください。ちなみにこのメニューの 3 つ目にあるのは、BoolCtl 型のポートを追加するためのものです。



「ラッチ」ポートは、Array OBC ばかりでなく、(Read/Write 間の) Latch OBC にも使えます。これはポートのプロパティーで切り替えます。



CONNECTION TYPE プロパティーには **Latch** と **Array** の選択肢があり、オブジェクト・バス接続の種類を表します。Write [] マクロ・ポートについては、もちろん **Array** を選択してください。

水平線が 2 本引かれたモジュールは **R/W Order** です (Built-In Module > Memory > R/W Order)。



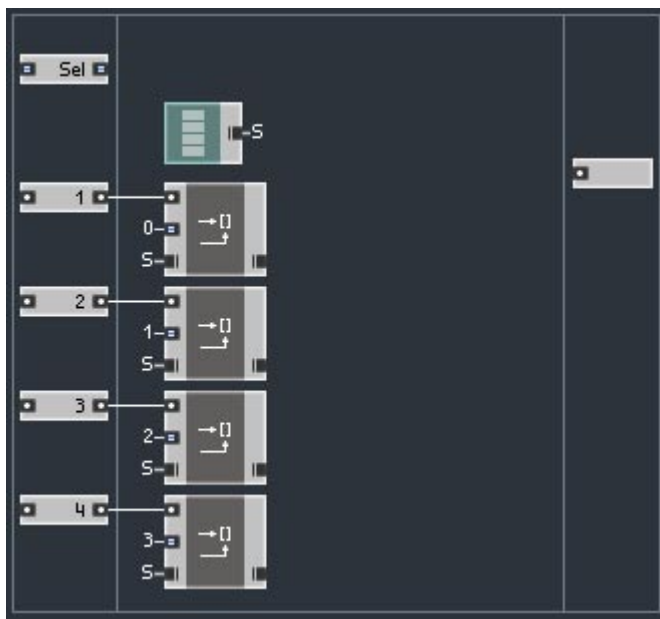
マスター入力 (下側) をそのままスレイブ出力に渡すだけのモジュールです。上側の入力に対しては何もしません。しかしここに接続するこ

とにより、モジュールの処理順序を変える効果があります。先に示したストラクチャーでは **Write** モジュールの出力がつながっているので、スレイブ出力以降のモジュールはすべて、**Write** モジュールよりもあとで処理されることになります。

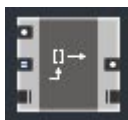
Write □ マクロの S 出力以降にあるモジュールは、マクロ自身よりもあとで処理されるものと考えるのが自然でしょう。しかし **R/W Order** モジュールを入れておかなければ、この順序が保証されません。こういった問題はオブジェクト・バス接続が絡む場合にしか起こりませんが、マクロの設計に当たってはこの点に注意し、必要に応じて **R/W Order** モジュールを組み込んでください。

OBC ポートと同様、**R/W Order** モジュールにも **CONNECTION TYPE** というプロパティがありますが、これは M ポートと S ポートの型にのみ影響します。上側の入力には常にラッチ・モードです。詳しくは付録を参照してください。

いよいよ入力信号をアレイに書き込むストラクチャーを組み立てます。

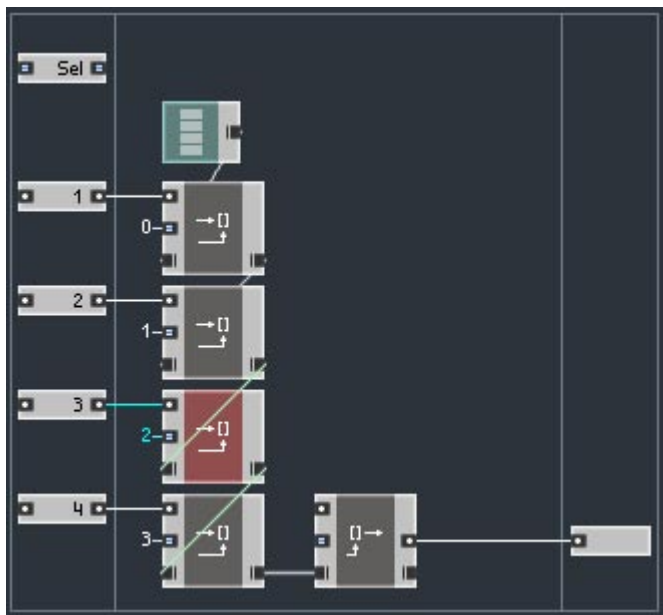


4つ並んでいる **Write []** モジュールは、オーディオ信号の値をアレイにそれぞれ保存するために使います。次に、この保存した値を読み取る仕組みが必要です。それには **Read []** マクロを使うのがよいでしょう (**Expert Macro > Memory > Read []**)。

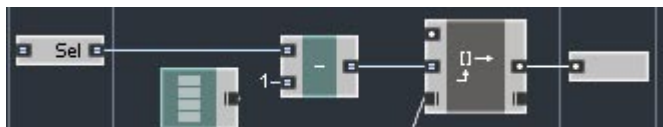


このマクロはアレイの要素を読み取って出力します。インデックスは2番目の入力に、整数値として与えます。一番上はクロック入力で、ここに与えられるイベントに合わせてアレイ要素の値を出力するようになっています。一番下の入出力ポートは、マスター / スレイブ接続用に使います。

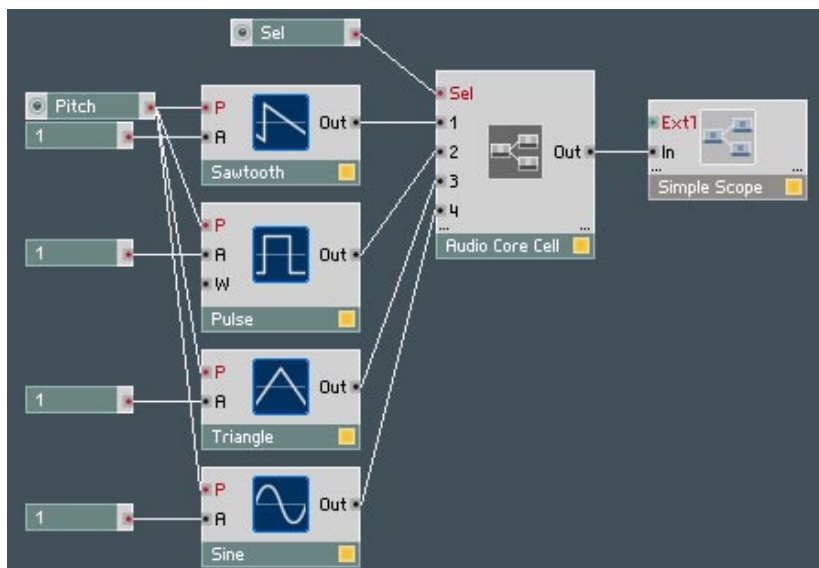
しかしこのマスター入力に何をつなげばよいのでしょうか。もちろんアレイ・モジュールを直接つなぐではありません。というのも、書き出し処理のあとで読み込むよう、順序を制御しなければならないからです。これをきちんとしておかないと、求める値が得られるまでに1サンプル分の遅延が生じたり、値そのものが信頼できないものになったりする恐れがあります。もちろん、対称性を考えると、**Write []** モジュールのうちいずれか1つにつなぐのではないことも明らかです。**Write []** モジュールの出力をすべて集めて **Read []** モジュールにつなげる、という方法も考えられますが、むしろ **Write []** モジュールを直列につなげ、その末端モジュールから **Read []** モジュールにつなげる方法をお勧めします。



Read [] モジュールのインデックス入力はどう扱うべきでしょうか。選択信号には 1 ～ 4 の番号が割り当てられているので、アレイに合わせて Sel 入力値から 1 を引いておきます。ここでは整数値として計算しますが、Sel は制御入力なので、変調マクロを使う必要はありません。

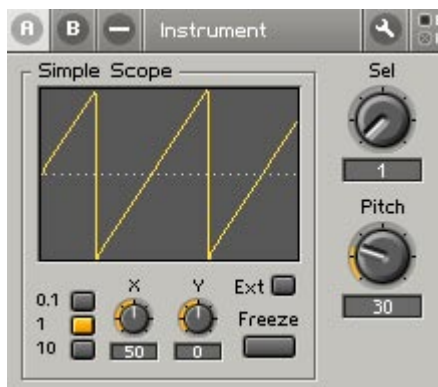


最後に、**Read []** モジュールに制御クロックを与えます。ここで作っているのはオーディオ信号のセクターなので、サンプル・レート・クロック (SR.C) に合わせます。



Sel ノブで 1 ～ 4 の選択番号を切り替えてください。

制御画面で見てみると、ノブの設定に応じて出力波形が切り替わるのが分かります。



8.3. ディレイの作成

アレイを使うもうひとつの例として、オーディオ信号用の簡単なディレイ・マクロを作ってみましょう。モジュールの外観は次のようになります。

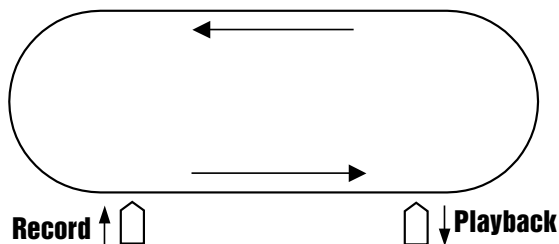


あるいは次のように、出力ポートの位置を上側の入力ポートに揃えた方がよいかも知れません。これはマクロの **Port Alignment** プロパティで調整します。

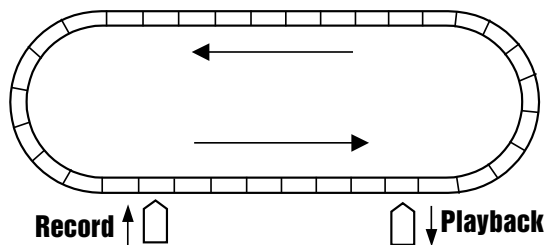


T 入力は遅延時間を秒単位で指定します。

アナログ式のテープ・ディレイ装置には、輪になったテープと、録音用 / 再生用のヘッドがついています。実際には消去ヘッドもついているのですが、話を簡単にするため、録音用のヘッドに消去機能もあるものとしましょう。

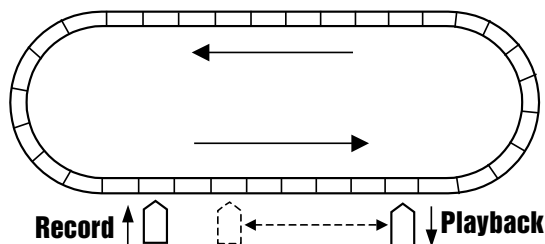


この動作をデジタル処理でシミュレートしようとするれば、テープの機能に相当するしくみが何か必要です。離散量しか扱えないという性質上、時間を有限個の区間に区切って、一定時間 (サンプル・レート) ごとにサンプルを録音 / 再生する形になります。



輪になったテープのシミュレートには、アレイを使うのが自然です。何サンプル分を録音 / 再生するか、その個数がアレイ長に相当します。

アナログ・テープの場合、遅延時間は録音 / 再生ヘッド間の距離とテープ走行速度によって決まります。ほとんどの場合、単なる技術上の理由により、距離を固定しておいて、走行速度に違いにより遅延時間を調整するようになっています。しかしデジタル処理の場合、テープ走行速度に相当するサンプル・レートを変に可変にするのは困難です。一方、ヘッド間の距離を変えるのは比較的簡単なので、その方針で作ることしましょう。



違いはもうひとつあります。アナログ・テープの場合、動いているのはテープの方です。しかしデジタル処理でこれをシミュレートしようとすれば、オーディオ・クロックごとにアレイの要素をすべて隣にコピーする、という処理が必要で、CPU に対する負荷が膨大です。そこでヘッドの方を動かすことにします。

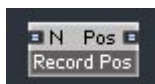
以上をまとめると次のようになります。

- | | |
|-------------|-------------------|
| アレイ | - 輪になったテープをシミュレート |
| 書き出し用インデックス | - 録音ヘッドの代用 |
| 読み込み用インデックス | - 再生ヘッドの代用 |

書き出し / 読み込み用インデックスは、1 オーディオ・サンプル相当の時間ごとに増加し、アレイ上の要素を順次指していきます。アレイの末尾に達すれば 0 に戻すことにより、ループを実現します。2 つのインデックスの差が、サンプル単位で測った遅延時間に相当します。

これはごく一般的なプログラミング技法で、「循環バッファ」あるいは「リング・バッファ」と呼ばれています。

まず録音ヘッドを作成します。先に作った鋸波発振器と同様のしくみですが、整数値モードで動作します。オーディオ・ティックごとに、0 から N-1 の範囲で循環的に値が増えていきます (N はアレイの大きさ)。これを **RecordPos** マクロとして実装しましょう。

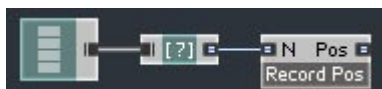


N 入力にアレイの要素数を与え、Pos 出力から書き出し用インデックスの値 (録音ヘッドの位置) を取得します。その実装は次のようになっています。鋸波発振器の実装と比べてみると分かりやすいでしょう。

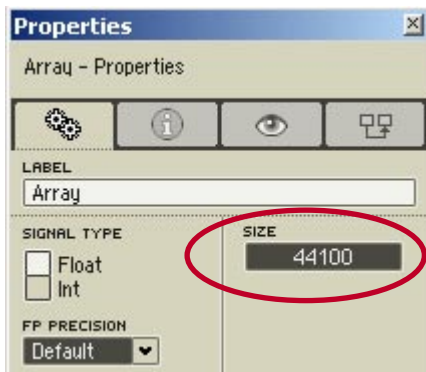


ここで比較モジュールが「>=」になっている点に注意してください。鋸波発振器の場合は「>=」でも「>」でも大きな差はありませんでしたが、今回は整数値演算なのでこの違いが重要です。アレイのインデックスとして N は範囲外になるので、値が N に達しないよう、「>=」という条件で比較しなければなりません。

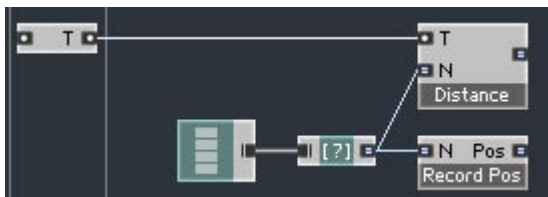
N 入力に与える値は、アレイの大きさを出力する、**Size []** モジュールを使って求めます (**Built-In Module > Memory > Size []**)。



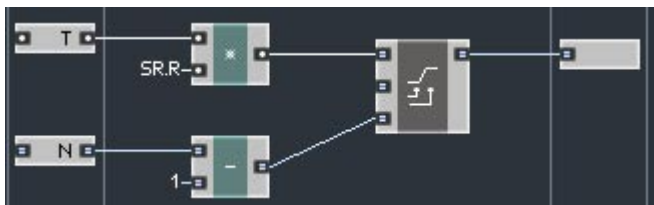
アレイ長をここでは 44100 と設定します。サンプル・レート
を 44.1kHz とすれば、最大 1 秒の遅延が可能です (厳密には 1 秒よりも 1
サンプル分少ない時間)。



次に読み込み用インデックスを計算するしくみを、2つのマクロを使っ
て作成します。ひとつは指定された遅延時間 (秒単位) をサンプル単位
に変換する **Distance** マクロです。



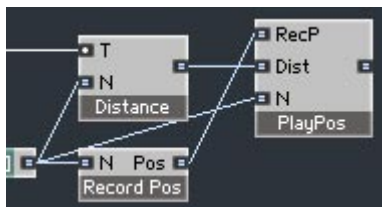
これはサンプル・レート (Hz 単位) を掛けて求めます。但しこ
の結果が N 以上にならないよう、**Expert Macro > Clipping >**
IClipMinMax マクロを使って制限します。



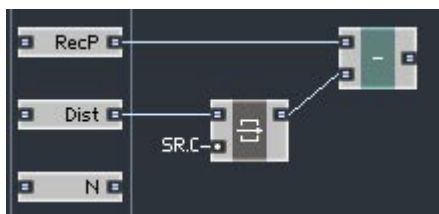
最大値を N-1 に抑えているのは、アレイの要素間の距離が最大で N-1
だからです。なお、整数値への変換は乗算のあとで行います。

乗算結果ではなく、指定された遅延時間の方で範囲をチェックする方法もあります(もちろん具体的な上限値は違います)。しかし、浮動小数点数を整数に変換した結果が一定しない場合があるので、乗算後にチェックする方が多少優っています。

次に、**RecordPos** と **Distance** をもとに、読み込み用インデックス(再生ヘッド)を計算する **PlayPos** マクロを作ります。



再生ヘッドの位置は録音ヘッドよりも遅延相当分うしろですから、引き算をすればよいことになります。



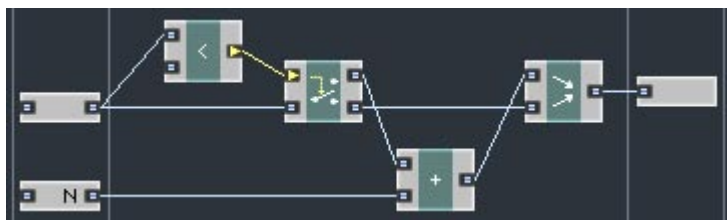
Dist 入力に与えられるのは制御信号なので、ラッチを挿入しています。オーディオ・クロックに同期しないタイミングで値が変わった旨のイベントが届くと結果が不安定になるので、それを回避するためです。

単純に引き算するだけだと、録音ヘッド位置が循環して 0 になったあとしばらく、結果が負になってしまいます。したがって、結果が -1 になったら N-1、-2 になったら N-2 などと変換しなければなりません。

そのため、次のように **Wrap** マクロを組み込んで剰余演算を施します。



今回は減算の結果が $-N+1$ より小さくなることはない (RecordPos の最小値が 0、Distance の最大値が $N-1$) ので、単に N を加算するだけでも充分です。

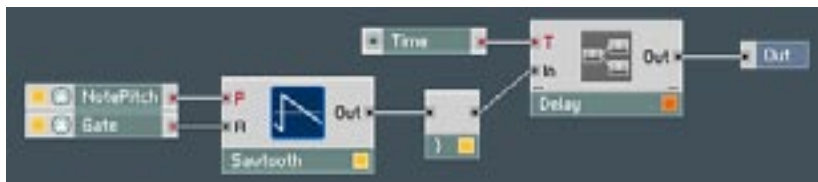


これで書き出し用 / 読み込み用インデックスを求めることができたので、最上位レベルに戻って全体を組み立てましょう。



読み込みは必ず書き出しのあとであること、SR.C をクロック源とし、これに同期して処理されることに注意してください。

動作確認用のストラクチャー例を示します。ミリ秒から秒への変換器を **Delay** セル・コアに置くこと、**Delay** セル・コアを単声モードにすることを忘れないでください。



声部ごとに 200K 程度のメモリーを消費するので、節約のため単声モードにしてください。1 秒に 44100 サンプルで、1 サンプルあたり 4 バイト (32 ビット) が必要なので、 $44100 \times 4 = 176400$ バイトとなります。これは 172K 強に相当します (1K は 1024 バイト)。

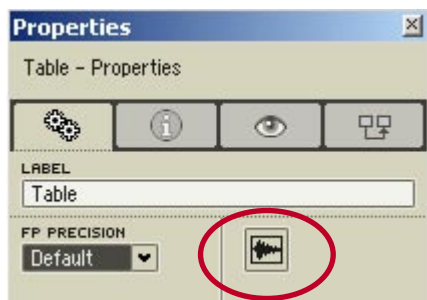
MIDI キーボードで何かを演奏し、**Time** ノブで指定した時間遅れて聴こえるのを確認してください。

8.4. 表

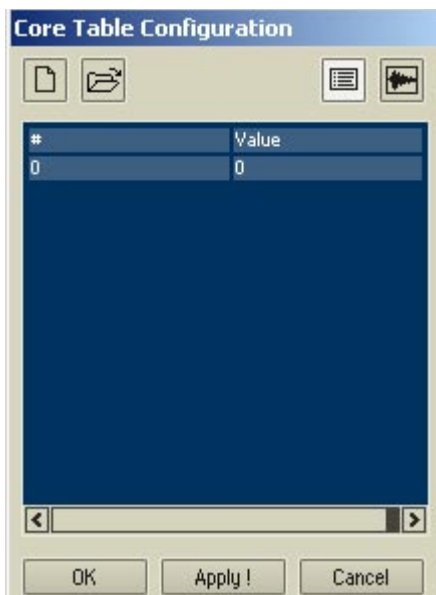
Table モジュールは、アレイに似た使い方ができます。**Built-In Module > Memory** サブメニュー以下にあります。




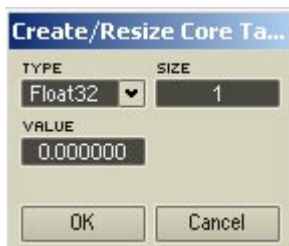
但しアレイと違って、読み込みしかできません。値はプロパティーとして事前に設定しておきます。値を表示、編集するには、**Properties** ウィンドウを開き、次のボタンを押してください。




すると次のようなダイアログ・ボックスが開きます。



この状態では、表の中身は空です。要素が1つだけあり、その値は0という状態です。ここに値を直接入力するほか、別にファイルに作成しておき、インポートすることも可能です。直接入力する場合は、 ボタンを押して、次のダイアログ・ボックスを開いてください。



ここで表に格納する値の型、要素数、各要素の初期値を設定します。
 ファイルからインポートする場合、オーディオ・ファイル (WAV/AIFF)、テキスト・ファイル (TXT/ASC)、独自形式のファイル (NTF; Native Table File) が使えます。 ボタンを押すとファイル選択ダイ

アログ・ボックスが開きます。ここでファイルを選択すると、表に格納するデータ型を訊ねられるので指定してください。

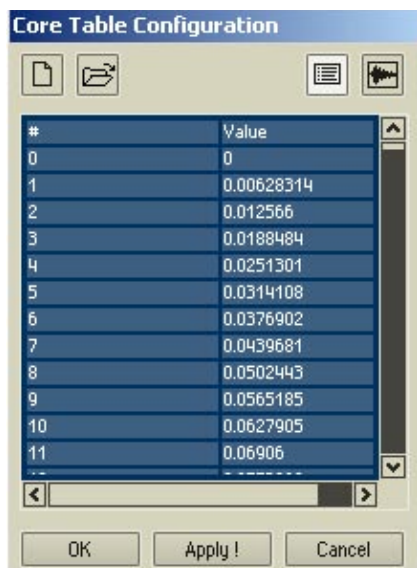
表の活用例として、正弦波発振器マクロを作ってみましょう。




ここに表モジュールを組み込みます。



表の値設定には sinetable.txt というファイルを使います。これは Reaktor のインストール先、「Core Tutorial Examples」というフォルダー内にあります。正弦関数の値が 1 周期分載っているので、Float32 型の数値としてインポートしてください。



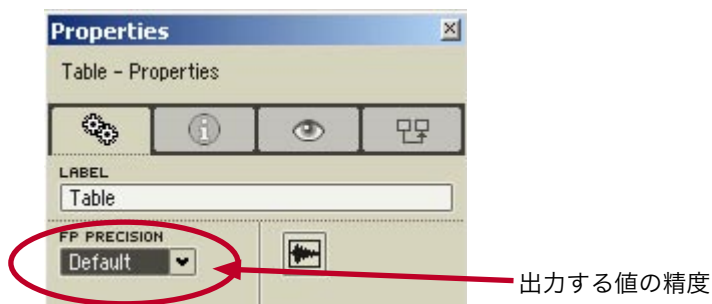
この値を波形グラフの形で表示することもできます。  ボタンと



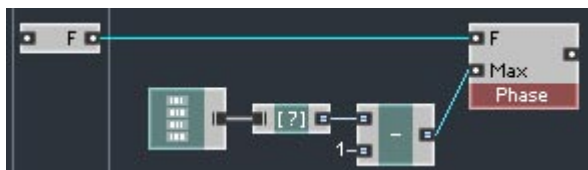
ボタンで、表示形式を切り替えてください。

OK ボタンを押してダイアログ・ボックスを閉じると、表の値が確定します。

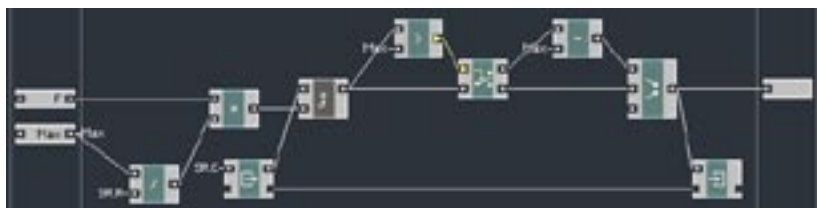
Properties ウィンドウには **FP PRECISION** というプロパティーもあります。これは表に格納した値そのものの精度ではなく、値を出力する際の精度です (値そのものの精度は別に設定しました)。通常は「Default」のままにしておいてください。



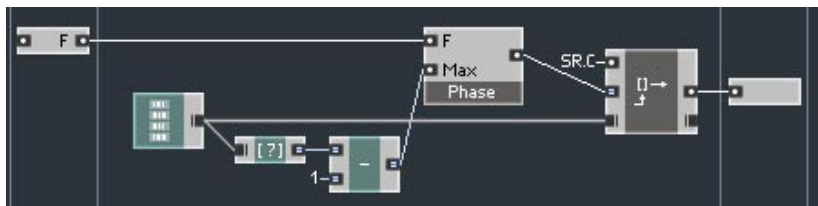
この表を使って発振器を実装します。中心となるのは位相発振器 **Phase** で、表の大きさを N としたとき、 0 から $N-1$ の範囲でインデックスを増やしていくために使います。



その中身は、テープ・ディレイで録音ヘッドの位置を制御するために使った、鋸波発振器と同様です。



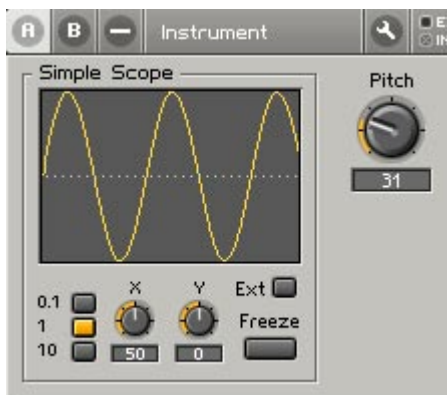
位相発振器につながっている **Read**  モジュールは、SR.C に同期して、該当するインデックスの値を出力するために使います。



動作確認用には次のようなストラクチャーがよいでしょう (コア・セルに P2F 変換器を忘れないでください)。



制御パネルは次のようになります。



こうして得られた正弦波は、それほど滑らかな曲線にはなりません。解消するには補間処理が必要になりますが、その実装については省略します。

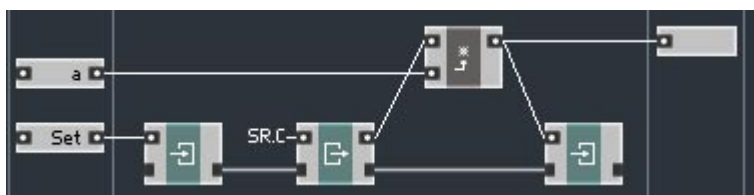
9. 効率のよいストラクチャーの作成

あらゆる面で理想的なツールというものはありません、Reaktor Core の技術もその例外ではありません。その能力を十全に引き出すために知っておくべきコツをいくつか紹介しましょう。

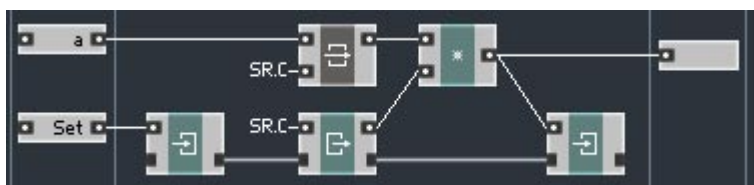
9.1. ラッチと変調マクロ

イベントを実際に必要なタイミングで確実に処理できるよう、ラッチや変調マクロを適切に使ってください。

次のストラクチャーでは、クロックに同期してオーディオ信号を処理するループで、乗算器として変調マクロを使っています。a 入力には非同期にイベントが与えられますが、変調マクロを使っているので、不適切なタイミングで演算が起こることはありません。



ラッチを使って同じ効果を得ることもできます。



これについては今までにいくつも例を挙げてきました。

ラッチを使うことにより性能を改善することはもちろんですが、仕様と異なるストラクチャーになってしまっただけでは何にもなりません。イベントを間違ったタイミングでモジュールに送る、という間違いがよくあります。

ラッチを使うと処理性能を損なう、という心配はいりません。ほとんど処理時間を要せず、CPU に対する負荷は実質的に 0 と言ってよいほどです。

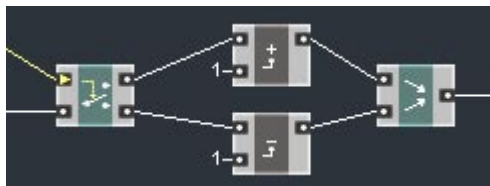
したがってイベントの振り分けにもラッチが向いています。ラッチとルーターのどちらを使っても構わないような場合は、ラッチを使うとよいでしょう。

9.2. 分岐処理とマージ

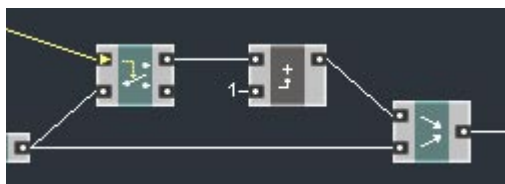
Router モジュールによるイベントの振り分け（分岐）は、状況によって一概には言えませんが、CPU に対する負荷がかなり高い傾向があります。ほかに負荷の高い処理を追加することなく **Router** を取り除けるのであれば、そのようにしてください。

ES Ctl モジュールによる処理は、ラッチで置き換え可能な場合が少なくありません。

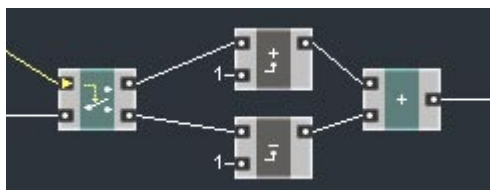
Router モジュールでイベントを振り分けた場合は、できるだけ、それぞれの信号経路をマージしてください。



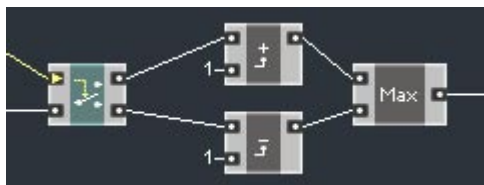
また、入力信号を **Router** モジュールに通した場合も、元の入力信号とマージするとよいでしょう。



マージ処理は **Merge** モジュール以外に、算術演算モジュールなどで行うことも可能です。



さらに、マクロ内のストラクチャーにもよりますが、マクロを使ってマージできることもあります。



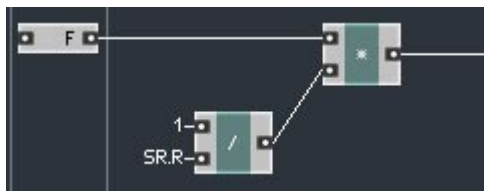
複数の **Router** モジュールでいくつにも分岐した信号も、原則として、最終的にはすべてマージするようにしてください。ただしこの場合、CPU に対する負荷も高くなります。

9.3. 数値演算

浮動小数点数の加算、乗算、減算、絶対値、符号反転などといった演算は、CPU にそれほど負荷がかかりません。整数の加算、減算、符号反転も負荷がかかりませんし、絶対値も同様です。 **Denormal Cancel**(非正規数の解消) は、現状では単なる加算を使って実装されています。

浮動小数点数の除算、整数の乗算および除算は、CPU に相応の負荷がかかります。

したがって、演算順序などを工夫して、負荷の高い演算が発生する頻度をできるだけ抑えるようにしてください。例えば、周波数 (Hz 単位) をサンプル・レートで割って正規化した周波数を求めたい場合、サンプル・レートの逆数をあらかじめ求めておき、これに周波数を掛けるようにするとよいでしょう。



上のストラクチャーで除算が発生するのは、サンプル・レートを変えた場合に限ります。周波数が変わっても、乗算だけで結果を得ることができます。

何も考えなければ次のようなストラクチャーにするかも知れません。

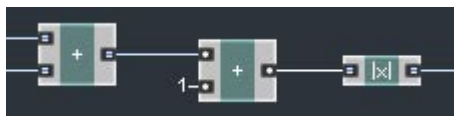


この場合、周波数が変わるたびに除算が発生してしまいます。

9.4. 浮動小数点数と整数の変換

浮動小数点数と整数の変換は、できるだけ避けるようにしてください。プラットフォームにもよりますが、CPU にかなりの負荷がかかります。処理の上でどうしても必要な場合は、もちろんきちんと使うべきです。

次のストラクチャーは一応正しく動作しますが、本来は不要な変換が2回入っています。



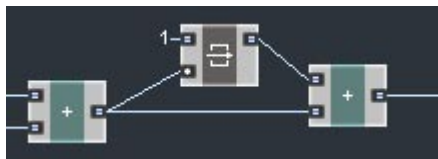
まず、中央の加算器モジュールの入力で変換が起こります。このモジュールは Float モードになっていますが、実際に届くのは整数値信号だからです。次に、絶対値モジュールの入力でも変換が起こります。このモジュールは Int モードですが、浮動小数点数値信号が届きます。

このストラクチャーは次のように作り変えるべきでしょう。



モジュールがすべて Int モードなので、変換は起こりません。

クロック信号は、一般には Float ですが、Int であっても問題ありません。



このストラクチャーでは、整数値信号を処理する **ILatch** モジュールに Float のクロック信号が入力されていますが、値を使うわけではないので変換は起こりません。

付録 A. ユーザー・インターフェイス

A.1. コア・セル

コア・セルの作成は、(アンサンブル以外の)基本レベル・ストラクチャー上で行います。背景部分で右クリックし、オーディオ型かイベント型かに応じて、**Core Cell > New Audio** または **Core Cell > New Event** を実行してください。

コア・セルのライブラリーには、システムに同梱されているものとユーザーが作成するものがありますが、いずれも同じ **Core Cell** メニュー以下に並んでいます。また、所定の場所がないライブラリーは、**Core Cell > Load...** コマンドで読み込めます。

不要になったコア・セルは、選択して **delete** キーを押すか、右クリックして **Delete** コマンドを実行してください。複数のコア・セルをまとめて削除することも可能です。

コア・セルをファイルに保存するには、右クリックして **Save Core Cell As...** コマンドを実行します。

コア・セルの内部ストラクチャーを編集したい場合は、当該コア・セル上をダブル・クリックしてください。逆に上位ストラクチャーに戻るには、画面の背景部分をダブル・クリックします。

コア・セル自体のプロパティーは、これを右クリックし、**Properties** コマンドで設定ウィンドウを開いて設定します。既にこのウィンドウが開いている場合は、コア・セルをクリックするだけでそのプロパティーに切り替わります。

コア・セルの内部実装に関するプロパティーを設定したい場合は、ストラクチャー内部を開いて背景部分を右クリックし、**Owner Properties** コマンドを実行してください。既に設定ウィンドウが開いている場合は、背景部分をクリックするだけで切り替わります。

A.2. コア・モジュール / マクロ

モジュールやマクロの作成には、コア・ストラクチャー編集画面の中央部分(処理部)で右クリックし、対象に応じて **Built-In Module**、**Expert Macro**、**Standard Macro**、**User macro** のいずれかのメニュー以下から選択してください。また、所定の場所がないモジュールやマクロは、**Core Cell > Load Module...** コマンドで読み込めます。

中身のないマクロは、**Built-In Module** メニュー以下で作成できます。作成したモジュール / マクロをファイルに保存するには、右クリックして **Save As...** コマンドを実行します。

不要になったモジュール / マクロは、選択して **delete** キーを押すか、右クリックして **Delete** コマンドを実行してください。複数のモジュール / マクロをまとめて削除することも可能です。

マクロの内部ストラクチャーを編集したい場合は、当該マクロ上をダブル・クリックしてください。逆に上位ストラクチャーに戻るには、画面の背景部分をダブル・クリックします。

モジュール / マクロのプロパティを設定したい場合は、ストラクチャー内部を開いて背景部分を右クリックし、**Owner Properties** コマンドを実行してください。既に設定ウィンドウが開いている場合は、背景部分をクリックするだけで切り替わります。

ストラクチャー内部を開かなくても、右クリックして **Properties** コマンドを実行すればプロパティを設定できます。既にこのウィンドウが開いている場合は、モジュール / マクロをクリックするだけでそのプロパティに切り替わります。

A.3. コア・ポート

入出力ポートは、コア・ストラクチャー編集画面の左側区画（入力部）、または右側区画（出力部）で右クリックし、**New** メニュー以下から必要なものを選択して追加します。

不要になったポートは、選択して **delete** キーを押すか、右クリックして **Delete** コマンドを実行してください。複数のポートをまとめて削除することも可能です。

A.4. コア・ストラクチャーの編集

コア・モジュールの位置は、マウスでつかんでドラッグすれば自由に調整できます。一方、ポートは上下方向に動かすことしかできません。この並び順が、上位レベルで外部から見たときの並び順にも反映されます。

あるモジュールの入力と別のモジュールの出力を接続するには、一方のポートをクリックしてそのままドラッグし、もう一方のポート上で放してください。

切断（接続の削除）は、その結線上をクリックして選択状態にし、**delete** キーを押してください。別法として、結線が出ている入力ポー



トをクリックしてそのままドラッグし、背景部分で放す、という操作でも切断できます。

QuickConst を作成するには、モジュールの入力ポート上を右クリックし、**Connect to New QuickConst** コマンドを実行してください。追加された QuickConst 上をクリックするとプロパティ画面が開きます。

QuickBus も同様に、モジュールの入力 / 出力ポート上を右クリックし、**Connect to New QuickBus** コマンドで作成します。作成済みの QuickBus に別のモジュールの入力 / 出力ポートを接続するには、該当するポートを右クリックし、**Connect to QuickBus** メニュー以下から接続先のバスを選択してください。

付録 B. Reaktor Core で使われる用語

B.1. 信号、イベント

信号には浮動小数点数 (Float) 型と整数 (Int) 型があります。Float 型信号を入出力するポートは 、一方、Int 型信号用のポートは  と表示されます。

信号はイベントの形で、あるモジュールの出力ポートから、接続先モジュールの入力ポートに伝播します。イベントは所定の要因によってあるモジュールの出力ポートに発生し、出力値の変化という形で現れます。但し、イベントの前後でたまたま値が変わらないこともあります。

同じイベント源から発生したイベントは、論理的にすべて同時に発生したとみなします。したがって、同じモジュールに複数の入力ポートがあって、それぞれにイベントが届いた場合、実際には時間差があっても、同時に届いたと考えて処理することになります。

イベント源が同じであるとは、発生元の出力ポートが同じということですが、ある条件を満たせば、異なる出力ポートで発生した場合でもイベント源が同じとみなします。例えば、コア・セルに対するオーディオ入力や、標準のサンプル・レート・クロック供給源は、すべて同じイベント源と考えます。また、初期設定の際に送出されるイベントも、同じイベント源から届くものとして扱います。この場合、イベント源は同じでも値が同じとは限りませんが、すべて同時に発生したものとして処理します。

自分自身がイベント源である場合を除き、入力値に対して処理を起動できるのは、入力ポートにイベントが届いたときに限ります。複数のイベントが届いた場合でも、同時に届いたとみなされるため、最終的に出力されるイベントは 1 つだけです。

B.2. 初期設定

ストラクチャーの初期設定は、値をすべて 0 にすることから始まります。次いで初期設定イベント源から、イベントが同時に送り出されます。イベント源としては、定数モジュール、コア・セルの入力、クロックなどがあります。

B.3. オブジェクト・バス接続

オブジェクト・バス接続 (Object Bus Connection、OBC) は、モジュール間を接続するものですが、それ自身が信号をやり取りすることはありません。モジュール間で記憶域を共有する、という働きがあります。典型的な使い方の例として、**Read/Write** モジュール間を接続し、共有メモリーを介して値を受け渡しする、というものがあります。

B.4. 信号経路の分岐

Router モジュールを使うと、条件によって信号の経路を振り分けることができます。出力ポートが2つありますが、入力イベントはそのどちらかにのみ出力されます。もう一方にはイベントが発生せず、したがって値が変わることもありません。

どちらに振り分けるかは、**BoolCtl** 型の入力で決まります。多くの場合、**Compare** モジュールによる比較結果を与えます。**Router** モジュールとの間に **BoolCtl** マクロなどの中継モジュールを置くこともあります。

Router を使えば、ストラクチャーによる処理内容を実行時に切り替えることができます。

分岐した信号経路は、通常、**Merge** その他のモジュールで1本の経路にマージすることになります。**Router** モジュールの出力と、このモジュールに入力する前の(分岐していない)信号とをマージすることもあります。必ずこう接続しなければならない、という制約はありませんが、処理性能には注意してください。

B.5. ラッチ

ラッチは、イベントが適切でないタイミングで送られるのを防ぐために、頻繁に使われる技法です。例えばオーディオ信号の処理中は、制御信号イベントが送られてきても、その時点で処理パラメーターを計算し直すとは具合の悪いことが起こります。

ラッチの代わりに、**Expert Macros > Modulation** メニュー以下に並んでいる変調マクロを使う方法もあります。これはよく使われる算術演算モジュールとラッチを組み合わせたものです。

B.6. クロック信号

これはイベント源のひとつで、所定の時間間隔でイベントが発生します。発振器、フィルターなど、さまざまなモジュールの動作に欠かせま

せん。こういったモジュールでは、表面上はクロック信号に接続しなくてもよいようになっていますが、実際にはコア・ストラクチャー内で、標準のクロック源につながっています。これをサンプル・レート・クロック (SR.C) といい、処理対象であるオーディオ信号のサンプル・レートに同期しています。

なお、イベント型コア・セルの場合、サンプル・レート・クロックを接続することは可能ですが、クロック信号を処理できません。したがって、発振器、フィルタなどのモジュールをイベント型で実装することはできません。

付録 C. コア・マクロ用のポート

C.1. In



コア・マクロ外からイベントを受け取り、そのまま内部に転送します。マクロ内部と外部を中継する役割があります。

ポートに何も接続されていない場合のデフォルト値を設定できます。

C.2. Out



処理結果のイベントを受け取り、そのままコア・マクロ外に出力します。

C.3. Latch (入力)



オブジェクト・バス接続を、マクロの外部から内部に向けて中継します。なお、ポートに何も接続されていない場合のデフォルト値を設定できます。

C.4. Latch (出力)



オブジェクト・バス接続を、マクロの内部から外部に向けて中継します。

C.5. Bool C (入力)



BoolCtl 接続を、マクロの外部から内部に向けて中継します。なお、ポートに何も接続されていない場合のデフォルト値を設定できます。

C.6. Bool C (出力)



BoolCtl 接続を、マクロの内部から外部に向けて中継します。

付録 D. コア・セル用のポート

D.1. In (オーディオ・モード)



モジュール外部からコア・セルに対してオーディオ信号を供給するインターフェイスです。サンプル・レートに従って (SR.C に同期して)、一定時間間隔でイベントを送ります。

初期設定イベントを送出する機能があります。値はコア・セルの外部から設定します。

D.2. Out (オーディオ・モード)



コア・セル内の処理の結果求められた値を、モジュール外部に送出するためのインターフェイスです。直近に求められた値を保持しており、外部からは任意のタイミングで読み取れます。

D.3. In (イベント・モード)



外部から供給された基本レベルのイベントを、コア・レベルのイベントに変換して内部に渡すインターフェイスです。

外部から初期設定イベントが与えられると、それに応じてコア・セル内部に**初期設定イベント**を供給する機能があります。

D.4. Out (イベント・モード)



コア・セル内の処理の結果生成されたコア・イベントを、基本レベルのイベントに変換し、外部に送出するためのインターフェイスです。複

数の出力ポートが同時にコア・イベントを受け取った場合は、並び順に従って上から順に基本レベルのイベントを送出します。

付録 E. 組み込みバス

E.1. **SR.C** (Sample-Rate Clock)

サンプル・レートに応じて一定時間間隔のクロック・イベントを送ります。

初期設定の際は常に**初期設定イベント**を送出します。

E.2. **SR.R** (Sample-Rate Rate)

サンプル・レート (Hz 単位) を、値が変わる都度、イベントの形で供給します。

初期設定の際は常に、初期状態のサンプル・レートを値とする**初期設定イベント**を送出します。

付録 F. 組み込みモジュール

F.1. Const



定数値の信号を生成します。モジュール上にはその値が表示されています。

初期設定の際に、所定の定数値の**初期設定イベント**を送出します。それ以外にこのモジュールがイベントを送出することはありません。

プロパティ:

Value 送出される定数値

F.2. Math > +



2つの入力信号の和を出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.3. Math > -



2つの入力信号の差を出力します (上側の入力値から下側の入力値を引いた値)。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.4. Math > *



2つの入力信号の積を出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.5. Math > /



2つの入力信号の商を出力します (上側の入力値を下側の入力値で除した値)。整数値モードであれば剰余は切り捨てます。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.6. Math > |x|



入力信号の絶対値を出力します。入力値が変化する都度、イベントを送出します。

F.7. Math > -x



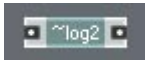
入力信号の符号を反転した値を出力します。入力値が変化する都度、イベントを送出します。

F.8. Math > DN Cancel



入力信号として非正規数が与えられた場合に、これを正規数に変換します。現在の実装では、微小定数を加算するという方法で実現しています。対象は浮動小数点数値のみであり、イベントが入力される都度、出力にイベントが発生します。

F.9. Math > ~log

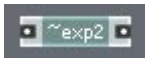


入力値の対数を近似計算します。イベントが入力される都度、出力にイベントが発生します。

プロパティ：

Base	対数の底
Precision	精度 (これを高くすると CPU に対する負荷も増します)

F.10. Math > ~exp



入力値の指数 (冪) を近似計算します。イベントが入力される都度、出力にイベントが発生します。

プロパティ：

Base	冪乗の底
Precision	精度 (これを高くすると CPU に対する負荷も増します)

F.11. Bit > Bit AND



入力信号の、ビットごとの論理積を出力します。対象は整数値のみであり、一方の入力値が変化するか、または両方の入力値が同時に変化する都度、イベントを送出します。

F.12. Bit > Bit OR



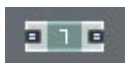
入力信号の、ビットごとの論理和を出力します。対象は整数値のみであり、一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.13. Bit > Bit XOR



入力信号の、ビットごとの排他的論理和を出力します。対象は整数値のみであり、一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.14. Bit > Bit NOT



入力信号の、ビットごとの否定を出力します。対象は整数値のみであり、入力値が変化する都度、イベントを送出します。

F.15. Bit > Bit <<



上側の入力に与えられた信号を左に論理シフトします。シフト幅 N は下側の入力で指定しますが、 $N < 0$ または $N > 31$ の場合の結果は不定なので、 $0 \leq N \leq 31$ の範囲で指定してください。対象は整数値のみであり、一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.16. Bit > Bit >>



上側の入力に与えられた信号を右に論理シフトします。シフト幅 N は下側の入力で指定しますが、 $N < 0$ または $N > 31$ の場合の結果は不定なので、 $0 \leq N \leq 31$ の範囲で指定してください。対象は整数値のみであり、一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

F.17. Flow > Router



制御信号 (上側) の値に応じて、信号入力 (下側) を上下いずれかの出力に振り分けます。すなわち、制御信号が真であれば出力 1 (上側)、偽であれば出力 0 (下側) になります。信号入力を与えられる都度、いずれかの出力に 1 つだけイベントを送出します。

F.18. Flow > Compare



2 つの入力値を比較した結果に応じて BoolCtl 信号を出力します。上側の入力比較演算子の左辺、下側の入力比較演算子の右辺に相当します。したがって図の場合、上側の入力値が下側よりも大きいときに真になります。

プロパティ:

Criterion	比較演算の種類
-----------	---------

F.19. Flow > Compare Sign



2 つの入力値の符号を比較した結果に応じて BoolCtl 信号を出力します。上側の入力比較演算子の左辺、下側の入力比較演算子の右辺に相当します。したがって図の場合、上側の入力値の符号が下側よりも大きいときに真になります。

符号の比較は次のように定義されています。

「+」は「+」と等しい
「-」は「-」と等しい
「+」は「-」よりも大きい

0 の符号は未定義なので、ある値と 0 とを比較した場合の結果は不定です。

プロパティー:

Criterion 比較演算の種類

F.20. Flow > ES Ctl



入力ポートにイベントが与えられているかどうかを、BoolCtl 信号の形で出力します。イベントが存在すれば真になります。

F.21. Flow > ~BoolCtl



BoolCtl 信号の否定を出力します。

F.22. Flow > Merge

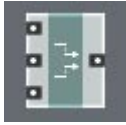


いずれかの入力にイベントが与えられた、あるいはいくつかの入力に同時にイベントが与えられた都度、それに応じたイベントを出力します。前者の場合は入力されたイベントの値をそのまま出力しますが、後者の場合は、イベントが与えられた中で一番下側のものを選択します。例えば上から数えて 2 番目と 3 番目の入力に、同時にイベントが与えられた場合は、3 番目の入力と同じ値のイベントを出力します。

プロパティー:

Input Count 入力ポートの個数

F.23. Flow > EvtMerge



機能は **Merge** モジュールと同様ですが、イベントの値は無視するため、出力イベントの値も不定です。クロックなどの信号源として使うことを意図したものです。値はいずれにしても使わないので、Float モードでのみ動作するようになっています。

プロパティ:

Input Count 入力ポートの個数

F.24. Memory > Read



オブジェクト・バス接続によって共有されるメモリー領域から値を読み込みます。クロック入力 (上側) に与えられるイベントに同期して値を読み込み、上側の出力から送出します。下側の入出力ポートは、オブジェクト・バス接続のマスター / スレーブです。

F.25. Memory > Write



オブジェクト・バス接続によって共有されるメモリー領域に値を書き出します。クロック入力 (上側) に与えられるイベントに同期して値を書き出します。下側の入出力ポートは、オブジェクト・バス接続のマスター / スレーブです。

F.26. Memory > R/W Order



このモジュール自身は何もしませんが、オブジェクト・バス接続されたモジュール間の処理順序を変える効果があります。下側の入出力ポートはオブジェクト・バス接続のマスター / スレイブで、信号をそのまま通します。上側の入力「サイド・チェーン (側鎖)」接続というもので、ここにつながっているモジュールよりも、スレイブ出力以降のモジュールが、必ず論理的にあとで処理されることになります。

サイド・チェーンには **Latch OBC** 型のモジュールしか接続できません。一方、マスター / スレイブ・ポートには、プロパティの設定により、**Latch OBC** 型、**Array OBC** 型のいずれかのモジュールを接続できます。いずれにしても、接続できるのは、型と精度が一致する信号に限ります。したがって例えば、サイド・チェーンに整数値の **Read** モジュール、マスター / スレイブに浮動小数点数値のモジュールをつなぐ、といったことはできません。

プロパティ:

Connection Type	マスター / スレイブ・ポートに接続するモジュールの種類 (Latch OBC または Array OBC)
------------------------	--

F.27. Memory > Array

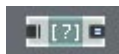


メモリー・オブジェクトのアレイを定義します。このモジュール自身は何の処理もしません。アレイに対する処理は、出力ポート (OBC スレイブ接続) に他のモジュールを接続して行います。

プロパティ:

Size	アレイの要素数
-------------	---------

F.28. Memory > Size []



入力ポートに接続されたアレイの要素数 (整数値) を出力します。

初期設定の際に、要素数を値とする**初期設定イベント**を送出します。

これ以外にこのモジュールがイベントを送出することはありません。

F.29. Memory > Index



アレイのある要素にアクセスするために使います。アレイ・モジュールと **Latch OBC** 型で接続しておき、インデックス番号を与えて、アレイの要素を取り出します。インデックス (上側の入力) は整数値で、0 から始まる番号です。下側の入力はおブジェクト・バス接続のマスター側で、ここにアレイ・モジュールを接続します。出力は、インデックスで指定された要素に対する、**Latch OBC** 型の接続です。各要素の型と精度は当然、オブジェクト・バス接続の入力と出力で一致しており、プロパティーとして設定するようになっています。

F.30. Memory > Table



読み込み専用のアレイ (表) を定義します。このモジュール自身は何の処理もしません。表に対する処理は、出力ポート (OBC スレイブ接続) に他のモジュールを接続して行います。

プロパティー：



表の各要素の値を設定



FP Precision 出力する値の精度

F.31. Macro



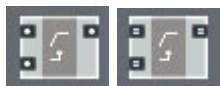
まとまった機能を実装したストラクチャーをひとつの単位として扱えるようにするための「入れ物」です。入出力ポートの個数は設計時に変更可能で、内部ストラクチャーによって決まります。

プロパティ：

- | | |
|----------------------|--|
| FP Precision | 出力する値の精度 |
| Look | 表示形式。ラベルとポート名を表示する Large と、表示を省略する Small から選択 |
| Pin Alignment | 外側から見たとき、ポートの並びを揃える基準 |
| Solid | コア・エンジンの処理方法を切り替え。オフにすると、帰還を解消するなどの処理で、マクロ外部と内部に境界がないものとして扱います。特に必要な場合を除き、オンのままにしておいてください。 |
| Icon |  ボタンを押すと、マクロを表示する際のアイコンを設定できます。  ボタンを押すとクリアされます。 |

付録 G. 高度なマクロ

G.1. Clipping > Clip Max / IClip Max



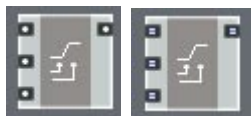
上側の入力に与えられた値が、下側の入力に与えられた最大値を上回らないよう制限します。なお、最大値を変更してもイベントは発生しません。

G.2. Clipping > Clip Min / IClip Min



上側の入力に与えられた値が、下側の入力に与えられた最小値を下回らないよう制限します。なお、最小値を変更してもイベントは発生しません。

G.3. Clipping > Clip MinMax / IClipMinMax



上側の入力に与えられた値が、中央の入力に与えられた最小値を下回らず、かつ下側の入力に与えられた最大値を上回らないよう制限します。なお、最小 / 最大値を変更してもイベントは発生しません。

G.4. Math > 1 div x



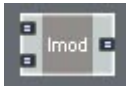
入力値の逆数を出力します。

G.5. Math > 1 wrap



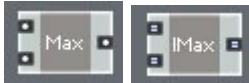
入力値を $-0.5 \sim +0.5$ の範囲に「折りたたむ」演算をします (周期は 1)。
{ (入力値 + 0.5) の小数部分 } - 0.5 という計算です。

G.6. Math > lmod



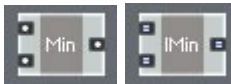
上側の入力値を下側の入力値で割った剰余を出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

G.7. Math > Max / lMax



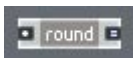
各入力の最大値を出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

G.8. Math > Min / lMin



各入力の最小値を出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

G.9. Math > round



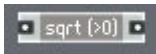
入力値に最も近い整数値を出力します。なお、隣り合う整数のちょうど中央に当たる、例えば 1.5 を入力した場合の結果は、1 になるか 2 になるか不定です。

G.10. Math > sign +/-



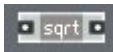
入力値の符号に従い、正ならば 1、負ならば -1 を出力します。0 の場合はイベントが送出されません。

G.11. Math > sqrt (>0)



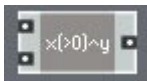
入力値の平方根を近似計算します。正の値を与えた場合のみ動作します。

G.12. Math > sqrt



入力値の平方根を近似計算します。0 を与えた場合も動作します。

G.13. Math > x(>0)^y



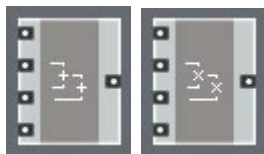
x の y 乗を近似計算します。x は正であるとしします。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。

G.14. Math > x^2 / x^3 / x^4



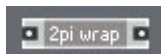
x の 2 乗 / 3 乗 / 4 乗を出力します。

G.15. Math > Chain Add / Chain Mult



上側の入力から順に足し合わせた / 掛け合わせた結果を出力します。
いずれかの入力値が変化する都度、イベントを送出します。

G.16. Math > Trig-Hyp > 2 pi wrap



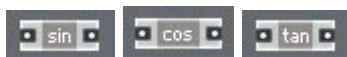
入力値を $-\pi \sim +\pi$ の範囲に「折りたたむ」演算をします (周期は 2π)。 π を加えて 2π を法とする剰余を取り、その結果から π を引く、という計算です。

G.17. Math > Trig-Hyp > arcsin / arccos / arctan



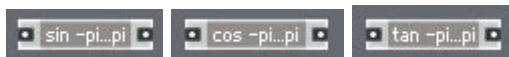
逆正弦 / 逆余弦 / 逆正接を近似計算します。

G.18. Math > Trig-Hyp > sin / cos / tan



正弦 / 余弦 / 正接を近似計算します。

G.19. Math > Trig-Hyp > sin -pi..pi / cos -pi..pi / tan -pi..pi



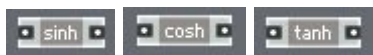
正弦 / 余弦 / 正接を近似計算します。但し入力値は $-\pi \sim \pi$ の範囲であるとしています。

G.20. Math > Trig-Hyp > tan -pi4..pi4



正接を近似計算します。但し入力値は $-\pi/4 \sim \pi/4$ の範囲であるとしています。

G.21. Math > Trig-Hyp > sinh / cosh / tanh



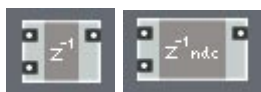
双曲正弦 / 双曲余弦 / 双曲正接を近似計算します。

G.22. Memory > Latch / lLatch



上側の入力に与えられた信号をラッチに保存しておき、クロック信号 (下側の入力) に同期して出力します。両方のイベントが同時に届いた場合は、入力信号がそのまま出力されます。

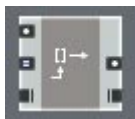
G.23. Memory > z^-1 / z^-1 ndc



クロック信号 (下側の入力) に同期して、上側の入力に与えられた 1 つ前の値を (遅延させて) 出力します。クロック信号を与えなければ、サンプル・レートに合わせた標準のオーディオ・クロック (SR.C) を使うので、1 サンプル分の遅延を与えることになります。

どちらのモジュールも帰還を解消するために使えますが、非正規数の解消を行うかどうかという違いがあります。**z^-1 ndc** を使うのは、非正規数が決して現れないと分かっている場合だけにしてください。

G.24. Memory > Read []



アレイのある要素を読み取り、クロック・イベントに同期して出力します。上側の入力にクロック、中央の入力にインデックスを与えます。一番下はオブジェクト・バス接続の入力で、ここにアレイを接続します。

オブジェクト・バス接続 (OBC) の出力は、一連のアレイ操作モジュールを OBC で接続し、所定の順序で処理されるようにするためのものです。

G.25. Memory > Write []



アレイのある要素に対し、値を書き出します。上側の入力に値、中央の入力にインデックスを与えます。値が入力されることにより、書き出し操作のトリガーがかかります。一番下はオブジェクト・バス接続の入力で、ここにアレイを接続します。

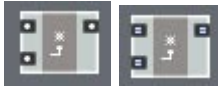
オブジェクト・バス接続 (OBC) の出力は、一連のアレイ操作モジュールを OBC で接続し、所定の順序で処理されるようにするためのものです。

G.26. Modulation > $x + a$ / Integer > $lx + a$



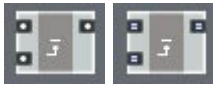
信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値を加算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

G.27. Modulation > $x * a$ / Integer > $lx * a$



信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値を乗算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

G.28. Modulation > $x - a$ / Integer > $lx - a$



信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値を減算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

G.29. Modulation > $a - x$ / Integer > $la - x$



信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値から減算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

G.30. Modulation > x / a



信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値で除算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

G.31. Modulation > a / x



信号入力 (上側の入力) に応じて、パラメーター (下側の入力) 値を除算して出力します。パラメーター値が変わった時点では、イベントは生成されません。

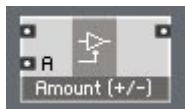
G.32. Modulation > xa + y



信号入力 (上側の入力) に応じて、利得パラメーター (中央の入力) を掛け、もうひとつの信号入力 (下側の入力) を加算して出力します。一方の入力値が変化する、または両方の入力値が同時に変化する都度、イベントを送出します。しかし利得が変わった時点では、イベントは生成されません。

付録 H. 標準マクロ

H.1. Audio Mix-Amp > Amount



オーディオ信号の振幅を線形に増幅 (定数倍) します。反転も可能です。

$A = 0$	無音 (振幅 0)
$A = 1$	元の信号のまま
$A = -1$	振幅を反転

主な用途：オーディオ帰還信号の制御

H.2. Audio Mix-Amp > Amp Mod



オーディオ信号の振幅を、AM 入力の指定に従って線形に変化させます。

$AM = 1$	振幅を 2 倍
$AM = 0$	元の信号のまま
$AM = -1$	無音 (振幅 0)

主な用途：トレモロ、振幅変調

H.3. Audio Mix-Amp > Audio Mix



2 つのオーディオ信号をミックスします。

H.4. Audio Mix-Amp > Audio Relay



x 入力に応じて、2つのオーディオ信号のうち一方を出力します。x が正ならば信号 1、負または 0 ならば信号 0 を出力します。

H.5. Audio Mix-Amp > Chain (+/- amount)

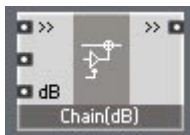


A 入力に従って入力信号の振幅を線形に変化させ、チェーン・オーディオ信号 (>>) とミックスして出力します。

- | | |
|--------|-----------|
| A = 0 | 無音 (振幅 0) |
| A = 1 | 元の信号のまま |
| A = -1 | 振幅を反転 |

主な用途: オーディオ・ミキシング・チェーン、帰還させる信号量の制御

H.6. Audio Mix-Amp > Chain (dB)



dB 入力に従って入力信号の振幅を調整し、チェーン・オーディオ信号 (>>) とミックスして出力します。

主な用途: オーディオ・ミキシング・チェーン

H.7. Audio Mix-Amp > Gain (dB)

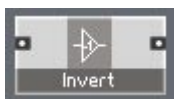


dB 入力に従って入力信号の振幅を調整します。

+6dB	振幅を 2 倍
0dB	元の信号のまま
-6dB	振幅を半減

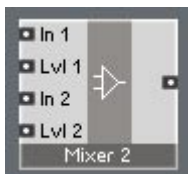
主な用途：音量制御 (dB 単位)

H.8. Audio Mix-Amp > Invert



オーディオ信号の極性を反転します。

H.9. Audio Mix-Amp > Mixer 2/3/4



入力オーディオ信号 (In 1/2/3/4) を、Lvl 1/2/3/4 の指定 (dB 単位) に従って減衰させ、その結果をミックスして出力します。

H.10. Audio Mix-Amp > Pan



入力信号を左右に振り分けてパンを実現します。特性曲線として放物線を使います

-1	最も左
0	中央
1	最も右

H.11. Audio Mix-Amp > Ring-Amp Mod



搬送信号（上側の入力）を、Mod 入力のオーディオ信号で変調します。変調の種類は、R/A 入力により、リング変調から振幅変調まで連続的に変化させることができます。

R/A = 0 リング変調

R/A = 1 振幅変調

（正常に振幅変調するためには、変調信号の振幅が 1 を超えないようにする必要があります）

H.12. Audio Mix-Amp > Stereo Amp



単声のオーディオ信号を dB 入力の指定に従って増幅し、Pan 入力で指定された方向から聴こえるようパンを調整します。パンは次のように指定します。

-1	最も左
0	中央
1	最も右

H.13. Audio Mix-Amp > Stereo Mixer 2/3/4



入力オーディオ信号 (In 1/2/3/4) を、Lvl 1/2/3/4 の指定 (dB 単位) に従って減衰させ、Pan 1/2/3/4 の指定に従ってパンを調整した結果をミックスして出力します。パンは次のように指定します。

-1	最も左
0	中央
1	最も右

H.14. Audio Mix-Amp > VCA



オーディオ信号を増幅します。振幅は A 入力に従って線形に変化します。

A = 0	無音 (振幅 0)
A = 1	元の信号のまま

主な用途 : A 入力に振幅エンベロープを接続

注 : 振幅を反転したい場合は **Audio Amount** モジュールを使ってください。

H.15. Audio Mix-Amp > XFade (lin)



オーディオ信号を線形にクロスフェードします。

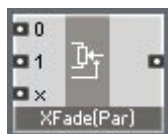
$x = 0$ 入力 0 の信号のみ出力

$x = 0.5$ 2 つの信号を半分ずつミックスして出力

$x = 1$ 入力 1 の信号のみ出力

注：線形よりも放物型のクロスフェードの方が自然に聞こえます。

H.16. Audio Mix-Amp > XFade (par)



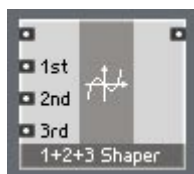
オーディオ信号を放物型でクロスフェードします。線形のクロスフェードよりも自然に聞こえるという特徴があります。

$x = 0$ 入力 0 の信号のみ出力

$x = 0.5$ 2 つの信号を半分ずつミックスして出力

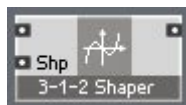
$x = 1$ 入力 1 の信号のみ出力

H.17. Audio Shaper > 1+2+3 Shaper



オーディオ信号の波形を調整するシェイパーで、2 次 / 3 次の歪み成分の量を調整できます。1st 入力、元の信号をどの位の割合で出力するか、を表します (1= そのまま出力、0= まったく出力しない)。2nd/ 3rd 入力、2 次 / 3 次の歪み成分を加える量を表します。

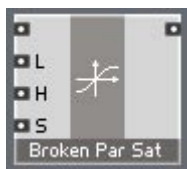
H.18. Audio Shaper > 3-1-2 Shaper



オーディオ信号の波形を調整するシェイパーで、2 次 /3 次の歪み成分の量を調整できます。歪み成分の加え方と量は Shp 入力で設定します。

Shp = 0	変形しない
Shp > 0	3 次成分による変形
Shp < 0	2 次成分による変形

H.19. Audio Shaper > Broken Par Sat



擬似放物型サチュレーターです。レベル 0 付近では線形になります。

L 入力 は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。

H 入力 (0 ~ 1) の値を大きくするほど、レベル 0 近傍の、特性曲線が線形である範囲が広がります。

S 入力 (-1 ~ 1) は特性曲線の対称性をあらわします。0 であれば完全に対称になります。

H.20. Audio Shaper > Hyperbol Sat



単純な双曲型サチュレーターです。L 入力 は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。もっとも、双曲型の場合、実際に完全飽和に達することはありません。

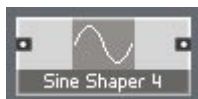
H.21. Audio Shaper > Parabol Sat



単純な放物型サチュレーターです。L 入力は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。

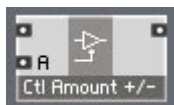
注: 完全に飽和するのは、入力レベルが 2L に等しいときです。

H.22. Audio Shaper > Sine Shaper 4/8



4 次 / 8 次の正弦曲線型シェイパーです。8 次の方が正弦曲線への近似の度合いが高いのですが、CPU に対する負荷も高くなります。

H.23. Control > Ctl Amount



制御信号の振幅を線形に増幅 (定数倍) します。反転も可能です。

A = 0	振幅 0
A = 1	元の信号のまま
A = -1	信号を反転

主な用途: 変調強度の制御

H.24. Control > Ctl Amp Mod



制御信号の振幅を、AM 入力の指定に従って線形に変化させます。

AM = 1	振幅を 2 倍
AM = 0	元の信号のまま
AM = -1	振幅 0

H.25. Control > Ctl Bi2Uni



両極の信号 (-1 ~ 1) を単極の信号 (0 ~ 1) に変換します。a 入力の変換の度合いを表します (デフォルト値は 1)。0 ならばまったく変換せず、1 ならば完全に変換します。

主な用途 : LFO による変調信号出力に直接接続して変調の極性を調整

H.26. Control > Ctl Chain



A 入力に従って制御信号の振幅を線形に変化させ、チェーン制御信号 (>>) とミックスして出力します。

A = 0	振幅 0
A = 1	元の信号のまま
A = -1	振幅を反転

主な用途 : ミキシング・チェーンの制御

H.27. Control > Ctl Invert



制御信号の極性を反転します。

H.28. Control > Ctl Mix



2つの制御信号をミックスします。

H.29. Control > Ctl Mixer 2



2つの制御信号 (In 1/2) にそれぞれ利得係数 (A 1/2) を適用したのち、ミックスして出力します。

$A = 0$	信号なし
$A = 1$	元の信号のまま
$A = -1$	反転

H.30. Control > Ctl Pan



制御信号を左右に振り分けてパンを実現します。特性曲線として放物線を使います

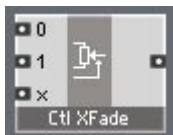
$Pos = -1$	最も左
$Pos = 0$	中央
$Pos = 1$	最も右

H.31. Control > Ctl Relay



x 入力に応じて、2つの制御信号のうち一方を出力します。x が正ならば信号 1、負または 0 ならば信号 0 を出力します。

H.32. Control > Ctl XFade



制御信号を線形にクロスフェードします。

$x = 0$	入力 0 の信号のみ出力
$x = 0.5$	2 つの信号を半分ずつミックスして出力
$x = 1$	入力 1 の信号のみ出力

H.33. Control > Par Ctl Shaper



制御信号の振幅を調整するシェイパーで、その特性曲線として、正負によって反転した放物線を使います。入力値の範囲は $-1 \sim 0 \sim 1$ 、これに応じて出力値も $-1 \sim 0 \sim 1$ の範囲になります。曲線を「曲げる (ベンド)」度合いは b 入力で設定しますが、この範囲も $-1 \sim 0 \sim 1$ です。

$b = 0$	ベンドなし (線形)
$b = -1$	X 軸側に最大限ベンド
$b = 1$	Y 軸側に最大限ベンド

入力値の範囲が $0 \sim 1$ であってもこのモジュールを適用できます。この場合、特性曲線のうち半分だけが使われることになります。

主な用途：ペロシティーその他の制御信号の調整

H.34. Convert > dB2AF



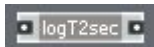
dB 単位 (対数目盛) の制御信号を、振幅の利得係数 (真数目盛) に変換します。0dB \rightarrow 1.0、-6dB0.5 などとなります。

H.35. Convert > dP2FF



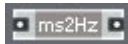
半音単位で音程を表す制御信号を、周波数比を表す値に変換します。
12 半音→ 2、-12 半音→ -2 となります。

H.36. Convert > logT2sec



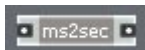
基本レベルでエンベロープ曲線を扱う際に使う、対数目盛の時間を、
真数目盛(秒)の時間に変換します。0→0.001 秒、60→1 秒となります。

H.37. Convert > ms2Hz



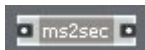
ミリ秒単位の時間を、これを周期とする周波数 (Hz 単位) に変換しま
す。100 ミリ秒→ 10Hz となります。

H.38. Convert > ms2sec



ミリ秒単位の時間を秒単位に変換します。500 ミリ秒→ 0.5 秒とな
ります。

H.39. Convert > P2F



MIDI ノート番号 (ピッチ) を周波数に変換します。69 番→ 440Hz
となります。

H.40. Convert > sec2Hz



秒単位の時間を、これを周期とする周波数 (Hz 単位) に変換します。0.1 秒 → 10Hz となります。

H.41. Delay > 2/4 Tap Delay 4p



2/4 タップのディレイで、4 点で補間を行います。T1 ~ T4 入力、各タップの遅延時間 (秒単位) を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒分に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

H.42. Delay > Delay 1p/2p/4p



1 点 (補間なし) / 2 点の補間 / 4 点の補間を行うディレイです。T 入力、遅延時間 (秒単位) を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒分に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

遅延時間を変調したい場合は、補間ありのモジュールを使ってください。逆に遅延時間が一定であれば、補間なしのモジュールが向いています。

H.43. Delay > Diff Delay 1p/2p/4p



1点(補間なし)/2点の補間/4点の補間を行うディフュージョン・ディレイです。T 入力は遅延時間(秒単位)、Dffs 入力はディフュージョン係数を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒分に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

H.44. Envelope > ADSR



ADSR エンベロープを生成します

- | | |
|-------|--|
| A、D、R | アタック時間、ディケイ時間、リリース時間を秒単位で指定します。 |
| S | サステイン・レベルを表します。範囲は 0 ~ 1 で、1 ならばピーク・レベルと等しくなります。 |
| G | ゲート入力。正值のイベントがあるとエンベロープの先頭から再開され、0 または負値のイベントがあると出力が中断されます。 |
| GS | ゲートの感度。0 にすると、エンベロープのピーク・レベルの振幅が常に 1 になります。1 にすると、ピーク・レベルは正のゲート・レベルに等しくなります。 |
| RM | 再トリガー・モード。アナログ / デジタルの別、および再トリガー / レガートの別を切り替えます。デジタル・モードならばエンベロープ先頭のレベルは常に 0 ですが、アナログ・モードにすると現在のレベルからエンベロープが始まるようになります。また、再トリガー・モードならば、正值のゲート・イベントがあるとエンベロープ先頭から再開されますが、レガート・モードにすると、0 また |

は負値から正值にゲート信号が切り替わった場合に限って再開されるようになります。次のいずれかの値を指定してください。

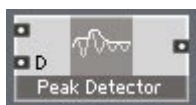
RM = 0	アナログ、再トリガー (デフォルト)
RM = 1	アナログ、レガート
RM = 2	デジタル、再トリガー
RM = 3	デジタル、レガート

H.45. Envelope > Env Follower



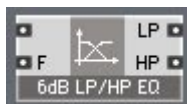
入力オーディオ信号のエンベロープに追従する制御信号を出力します。A 入力、D 入力は、アタック時間およびデケイ時間 (秒単位) を表します。

H.46. Envelope > Peak Detector



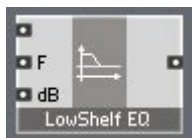
入力オーディオ信号の、直近のピーク値を、制御信号として出力します。D 入力は出力レベルのデケイ時間 (秒単位) を表します。

H.47. EQ > 6dB LP/HP EQ



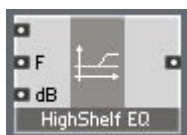
1 ポール (6dB/oct.) のロー / ハイ・パス・イコライザーです。F 入力はカットオフ周波数 (Hz 単位) を表します。

H.48. EQ > 6dB LowShelf EQ



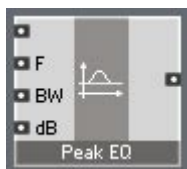
1 ポールのロー・シェルビング・イコライザーです。dB 入力値は低周波成分を増幅する度合いを表します (負値ならば減衰)。また、F 入力値は特性曲線が変化する領域の中心周波数 (Hz 単位) を表します。

H.49. EQ > 6dB HighShelf EQ



1 ポールのハイ・シェルビング・イコライザーです。dB 入力値は高周波成分を増幅する度合いを表します (負値ならば減衰)。また、F 入力値は特性曲線が変化する領域の中心周波数 (Hz 単位) を表します。

H.50. EQ > Peak EQ



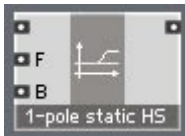
2 ポールのピーク / ノッチ・イコライザーです。F 入力値は中心周波数 (Hz 単位)、BW 入力値は帯域幅 (オクターブ単位)、dB 入力値はピーク部分の高さを表し、これが負であればノッチ・イコライザーということになります。

H.51. EQ > Static Filter > 1-pole static HP



1 ポールのスタティック型ハイ・パス・フィルターです。F 入力はカットオフ周波数 (Hz 単位) を表します。

H.52. EQ > Static Filter > 1-pole static HS



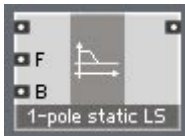
1 ポールのスタティック型ハイ・シェルピング・フィルターです。F 入力はカットオフ周波数 (Hz 単位)、B 入力は高周波成分の増幅率 (dB 単位) を表します。

H.53. EQ > Static Filter > 1-pole static LP



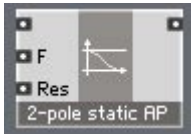
1 ポールのスタティック型ロー・パス・フィルターです。F 入力はカットオフ周波数 (Hz 単位) を表します。

H.54. EQ > Static Filter > 1-pole static LS



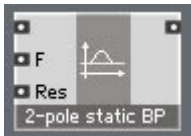
1 ポールのスタティック型ロー・シェルピング・フィルターです。F 入力はカットオフ周波数 (Hz 単位)、B 入力は低周波成分の増幅率 (dB 単位) を表します。

H.55. EQ > Static Filter > 2-pole static AP



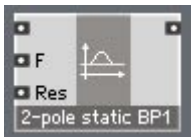
2 ポールのスタティック型オール・パス・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

H.56. EQ > Static Filter > 2-pole static BP



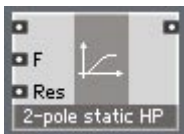
2 ポールのスタティック型バンド・パス・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

H.57. EQ > Static Filter > 2-pole static BP1



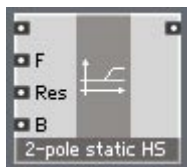
2 ポールのスタティック型バンド・パス・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。カットオフ周波数に一致する信号に対する増幅度は、レゾナンスに関係なく常に 1 となっています。

H.58. EQ > Static Filter > 2-pole static HP



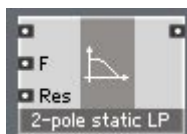
2 ポールのスタティック型ハイ・パス・フィルタです。F 入力のカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

H.59. EQ > Static Filter > 2-pole static HS



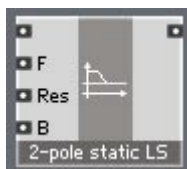
2 ポールのスタティック型ハイ・シェルフ・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1)、B 入力は高周波成分の増幅率 (dB 単位) を表します。

H.60. EQ > Static Filter > 2-pole static LP



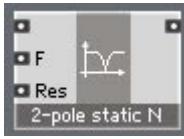
2 ポールのスタティック型ロー・パス・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

H.61. EQ > Static Filter > 2-pole static LS



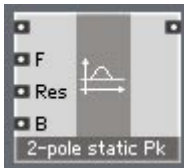
2 ポールのスタティック型ロー・シェルフ・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1)、B 入力は低周波成分の増幅率 (dB 単位) を表します。

H.62. EQ > Static Filter > 2-pole static N



2 ポールのスタティック型ノッチ・フィルターです。F 入力 is カットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1) を表します。

H.63. EQ > Static Filter > 2-pole static Pk



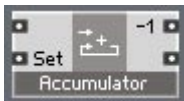
2 ポールのスタティック型ピーク・フィルターです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1)、B 入力は中心周波数成分の増幅率 (dB 単位) を表します。

H.64. EQ > Static Filter > Integrator



入力オーディオ信号を、矩形近似して加算する方法で積分します。Rst 入力にイベントを与えると、出力値が指定した値に初期化されます。

H.65. Event Processing > Accumulator



上側入力に与えられた値を累積します。Set 入力は累積値を初期化するために使います。下側の出力から累積値、上側の -1 出力からは 1 回前までの累積値を送出します。

H.66. Event Processing > Clk Div



クロック信号を分周します。分周率 N は下側入力で指定します。上側入力に届いたクロック・イベントは、 N 回おきに出力されます。

H.67. Event Processing > Clk Gen



クロック・イベントを、指定された周波数 (Hz 単位) で生成します。オーディオ型コア・セルでしか動作しません。

H.68. Event Processing > Clk Rate



入力クロック・イベントの周波数 (Hz 単位) と周期 (秒単位) を測定し、F 出力および T 出力から送じます。オーディオ型コア・セルでしか動作しません。

初期状態では、周期は 0、周波数は非常に大きな値になっています。クロック・イベントが少なくとも 2 回与えられなければ、意味のある値は得られません。

H.69. Event Processing > Counter



上側入力に与えられたイベントの個数を数えます。Set 入力は個数を初期化するために使います。下側の出力から現在までの個数、上側の -1 出力からは 1 回前までの個数を送出します。

H.70. Event Processing > Ctl2Gate



上側入力に与えられた制御信号（またはオーディオ信号）を、ゲート信号に変換します。ゲート信号の振幅は下側入力で指定します。制御信号が負から正に変化するとゲートが開き、正から負に変化するとゲートが閉まります。

H.71. Event Processing > Dup Flt / IDup Flt



値が重複するイベントを廃棄し、前回と値が異なるイベントのみ通します。

H.72. Event Processing > Impulse



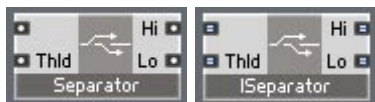
入力イベントに応じて、振幅 1、長さ 1 サンプル分のインパルスを生成します。オーディオ型コア・セルでしか動作しません。

H.73. Event Processing > Random



入力クロックに応じて、ランダムな値 (-1 ~ 1) のイベントを生成します。Seed 入力にイベントを与えると、乱数を生成する「種」として使われます。

H.74. Event Processing > Separator / ISeparator



上側入力に与えられたイベントを、Thld 値よりも大きいかなにかによって、Hi 出力と Lo 出力に振り分けます。

H.75. Event Processing > Thld Crossing



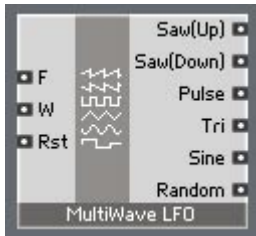
上側入力に与えられた信号が、Thld 値を下から上に横切ると、Up 出力にイベントを送出します。逆に上から下に横切った場合は Dn 出力に送ります。

H.76. Event Processing > Value / IValue



上側入力に与えられたイベントの値を、その時点の下側入力の値に置き換えて出力します。

H.77. LFO > MultiWave LFO



位相の揃った、各種の波形の低周波信号を生成します。F 入力 は 周波数 (Hz 単位)、W 入力 は パルス幅 (-1 ～ 0 ～ 1、パルス波のみに適用) を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.78. LFO > Par LFO



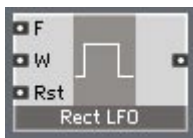
放物波の低周波制御信号を生成します。F 入力 は 周波数 (Hz 単位) を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.79. LFO > Random LFO



振幅がランダムに決まる低周波ステップ (サンプル & ホールド) 制御信号を生成します。F 入力 は 周波数 (Hz 単位) を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.80. LFO > Rect LFO



矩形波の低周波制御信号を生成します。F 入力 は 周波数 (Hz 単位)、W 入力 は パルス幅 (-1 ～ 0 ～ 1) を表します。Rst 入力 に イベント を 与え る と、その 値 (0 ～ 1) に 対 応 す る 位 相 か ら 発 振 が 再 開 さ れ ま す。

H.81. LFO > Saw(down) LFO



鋸 波 (下 向 き) の 低 周 波 制 御 信 号 を 生 成 し ま す。F 入 力 は 周 波 数 (Hz 単 位) を 表 し ま す。Rst 入 力 に イ ベ ン ト を 与 え る と、そ の 値 (0 ～ 1) に 対 応 す る 位 相 か ら 発 振 が 再 開 さ れ ま す。

H.82. LFO > Saw(up) LFO



鋸 波 (上 向 き) の 低 周 波 制 御 信 号 を 生 成 し ま す。F 入 力 は 周 波 数 (Hz 単 位) を 表 し ま す。Rst 入 力 に イ ベ ン ト を 与 え る と、そ の 値 (0 ～ 1) に 対 応 す る 位 相 か ら 発 振 が 再 開 さ れ ま す。

H.83. LFO > Sine LFO



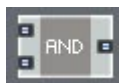
正弦波の低周波制御信号を生成します。F 入力(周波数 (Hz 単位))を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.84. LFO > Tri LFO



三角波の低周波制御信号を生成します。F 入力(周波数 (Hz 単位))を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.85. Logic > AND



2つの論理信号の論理積、すなわち、両方とも1であれば1、それ以外は0を出力します。入力値が0でも1でもない場合の結果は不定です。

H.86. Logic > Flip Flop



値が0と1の論理信号を、クロック入力ごとに交互に出力します。

H.87. Logic > Gate2L



ゲート信号を論理信号に変換します。ゲートが開いていれば1、閉まっていれば0を出力します。

H.88. Logic > GT / IGT



2つの浮動小数点数値 / 整数値を比較し、上側の方が大きければ 1、そうでなければ 0 を出力します。

H.89. Logic > EQ



2つの整数値を比較し、等しければ 1、そうでなければ 0 を出力します。

H.90. Logic > GE



2つの整数値を比較し、上側の方が大きいとか等しければ 1、そうでなければ 0 を出力します。

H.91. Logic > L2Clock



論理信号をクロック信号に変換します。入力信号が 0 から 1 に変化した時点でクロック・イベントを出力します。入力値が 0 でも 1 でもない場合の結果は不定です。

H.92. Logic > L2Gate



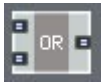
論理信号をゲート信号に変換します。入力信号が 0 から 1 に変化した時点でゲートを開き、1 から 0 に変化した時点で閉じます。ゲートが開いているときの信号レベルは、下側の入力で設定できます (デフォルト値は 1)。入力値が 0 でも 1 でもない場合の結果は不定です。

H.93. Logic > NOT



論理信号の否定、すなわち、1 ならば 0、0 ならば 1 を出力します。入力値が 0 でも 1 でもない場合の結果は不定です。

H.94. Logic > OR



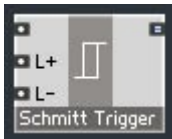
2 つの論理信号の論理和、すなわち、少なくとも一方が 1 であれば 1、それ以外は 0 を出力します。入力値が 0 でも 1 でもない場合の結果は不定です。

H.95. Logic > XOR



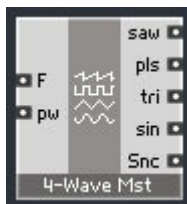
2 つの論理信号の排他的論理和、すなわち、いずれか一方が 1 であれば 1、それ以外は 0 を出力します。入力値が 0 でも 1 でもない場合の結果は不定です。

H.96. Logic > Schmitt Trigger



入力値が L+(デフォルト値は 0.67) を上回れば 1、L-(デフォルト値は 0.33) を下回れば 0 に値を切り替えて出力します。

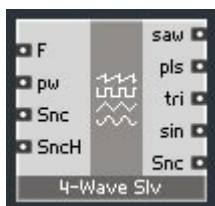
H.97. Oscillators > 4-Wave Mst



位相の揃った、4種類のオーディオ信号を生成します。F 入力 は 周波数 (Hz 単位)、pw 入力 は パルス幅 (-1 ～ 0 ～ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することも可能です。また、**4-Wave Slv** モジュール用の同期出力もあります。

H.98. Oscillators > 4-Wave Slv



位相の揃った、4種類のオーディオ信号を生成します。F 入力 は 周波数 (Hz 単位)、pw 入力 は パルス幅 (-1 ～ 0 ～ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することも可能です。また、他の **4-Wave Mst/Slv** モジュールと同期できます。SncH 入力 は 同期の度合いを表し、まったく同期しない 0 から完全に同期する 1 までの間で連続的に変化させることができます。さらに、他の **4-Wave Slv** モジュール用の同期出力もあります。

H.99. Oscillators > Binary Noise



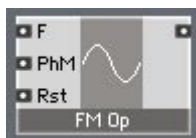
1 と -1 の 2 値をランダムに取る白色雑音を生成します。Seed 入力にイベントを与えると、乱数を生成する「種」として使われます。

H.100. Oscillators > Digital Noise



-1 ～ 1 の範囲の値をランダムに取る白色雑音を生成します。Seed 入力にイベントを与えると、乱数を生成する「種」として使われます。

H.101. Oscillators > FM Op



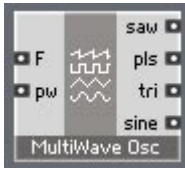
昔ながらの FM 音源です。正弦波を出力しますが、その周波数 (Hz 単位) は F 入力で指定します。これを PhM 入力 (ラジアン単位) で位相変調できます。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

H.102. Oscillators > Formant Osc



F 入力で指定された周波数 (Hz 単位) の信号をもとに、Fmt 入力で指定されたフォルマント周波数 (Hz 単位) の領域を強めた信号を生成します。

H.103. Oscillators > MultiWave Osc



位相の揃った、4種類のオーディオ信号を生成します。F 入力 は 周波数 (Hz 単位)、pw 入力 は パルス幅 (-1 ～ 0 ～ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することはできません。

H.104. Oscillators > Par Osc



放物波のオーディオ信号を生成します。F 入力 は 周波数 (Hz 単位) を表します。

H.105. Oscillators > Quad Osc



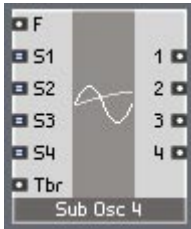
位相が 90°ずれた正弦波の組を生成します。F 入力 は 周波数 (Hz 単位) を表します。

H.106. Oscillators > Sin Osc



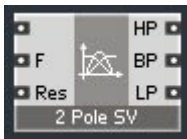
正弦波を生成します。F 入力 は 周波数 (Hz 単位) を表します。

H.107. Oscillators > Sub Osc 4



基本となる信号をもとに、位相の揃った4種類の分周信号を生成します。F 入力 は基底周波数 (Hz 単位) を表します。分周数は S1/S2/S3/S4 入力 で指定します (1 ~ 120)。Tbr 入力 は、出力波形に高周波成分が含まれる比率を表します (0 ~ 1)。

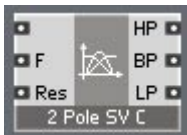
H.108. VCF > 2 Pole SV



2 ポールの状態可変型 (SV、State-Variable) フィルターです。F 入力 はカットオフ周波数 (Hz 単位)、Res 入力 はレゾナンス (0 ~ 0.98) を表します。

HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

H.109. VCF > 2 Pole SV C

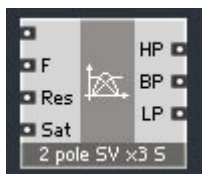


2 ポールの状態可変型 (SV、State-Variable) フィルターで、補償 (C、Compensation) の機能があるため、カットオフ周波数としてかなり高い値を指定できます。F 入力 はカットオフ周波数 (Hz 単位)、Res 入力 はレゾナンス (0 ~ 0.98) を表します。レゾナンス値として負の値を指

定すると、カットオフ周波数付近をあまり強調しないよう、特性曲線の傾斜が緩やかになります。

HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

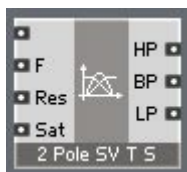
H.110. VCF > 2 Pole SV (x3) S



2 ポールの状態可変型 (SV、State-Variable) フィルターで、オーバー・サンプリング (x3 型) とサチュレーション (S) の機能があります。F 入力のカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1)、Sat 入力はサチュレーション・レベル (通常は 8 ~ 32 程度) を表します。

HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

H.111. VCF > 2 Pole SV T (S)



2 ポールの状態可変型 (SV、State-Variable) フィルターで、表にもとづく補償機能 (T、Table compensation) と、サチュレーション (S) の機能があります。カットオフ周波数が高い場合の性能に優れていますが、**2 Pole SV C** モジュールとは若干異なります。F 入力のカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1)、Sat 入力はサチュレーション・レベル (通常は 8 ~ 32 程度) を表します。

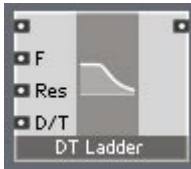
HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

H.112. VCF > Diode Ladder



ダイオード・ラダー・フィルタを線形にエミュレートしたものです。F 入力のカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 0.98) を表します。

H.113. VCF > D/T Ladder



ラダー・フィルタを線形にエミュレートしたもので、ダイオード型からトランジスタ型まで、連続的に変化させることができます。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 0.98) を表します。D/T 入力 (0 ～ 1) は、0 ならばダイオード型、1 ならばトランジスタ型を表します。

H.114. VCF > Ladder x3



サチュレーション機能付きのトランジスタ型ラダー・フィルタ (x3 回のオーバー・サンプル) をエミュレートしたものです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1)、Sat 入力はサチュレーション・レベル (通常は 8 ～ 32 程度) を表します。

出力 1 ～ 4 は、ラダーの対応するタップから出力される信号です。一般に言うラダー・フィルターの出力信号は、4 番出力から得られます。

付録I. コア・セル・ライブラリー

I.1. Audio Shaper > 3-1-2 Shaper



オーディオ信号の波形を調整するシェイパーで、2 次 / 3 次の歪み成分の量を調整できます。歪み成分の加え方と量は Shp 入力で設定します。

Shp = 0	変形しない
Shp > 0	3 次成分による変形
Shp < 0	2 次成分による変形

I.2. Audio Shaper > Broken Par Sat



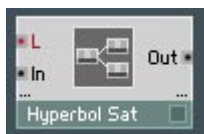
擬似放物型サチュレーターです。レベル 0 付近では線形になります。

L 入力は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。

H 入力 (0 ~ 1) の値を大きくするほど、レベル 0 近傍の、特性曲線が線形である範囲が広がります。

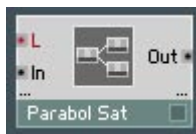
S 入力 (-1 ~ 1) は特性曲線の対称性をあらわします。0 であれば完全に対称になります。

I.3. Audio Shaper > Hyperbol Sat



単純な双曲型サチュレーターです。L 入力は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。もっとも、双曲型の場合、実際に完全飽和に達することはありません。

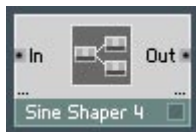
l.4. Audio Shaper > Parabol Sat



単純な放物型サチュレーターです。L 入力値は信号が完全に飽和したときの出力レベルを表します (デフォルト値は 1)。

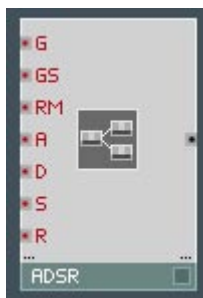
注: 完全に飽和するのは、入力レベルが 2L に等しいときです。

l.5. Audio Shaper > Sine Shaper 4/8



4 次 / 8 次の正弦曲線型シェイパーです。8 次の方が正弦曲線への近似の度合いが高いのですが、CPU に対する負荷も高くなります。

l.6. Control > ADSR



ADSR エンベロープを生成します

A、D、R アタック時間、ディケイ時間、リリース時間を秒単位で指定します。

S	サスティン・レベルを表します。範囲は 0 ～ 1 で、1 ならばピーク・レベルと等しくなります。
G	ゲート入力。正值のイベントがあるとエンベロープの先頭から再開され、0 または負値のイベントがあると出力が中断されます。
GS	ゲートの感度。0 にすると、エンベロープのピーク・レベルの振幅が常に 1 になります。1 にすると、ピーク・レベルは正のゲート・レベルに等しくなります。
RM	再トリガー・モード。アナログ / デジタルの別、および再トリガー / レガートの別を切り替えます。デジタル・モードならばエンベロープ先頭のレベルは常に 0 ですが、アナログ・モードにすると現在のレベルからエンベロープが始まるようになります。また、再トリガー・モードならば、正值のゲート・イベントがあるとエンベロープ先頭から再開されますが、レガート・モードにすると、0 または負値から正值にゲート信号が切り替わった場合に限って再開されるようになります。次のいずれかの値を指定してください。
RM = 0	アナログ、再トリガー (デフォルト)
RM = 1	アナログ、レガート
RM = 2	デジタル、再トリガー
RM = 3	デジタル、レガート

1.7. Control > Env Follower



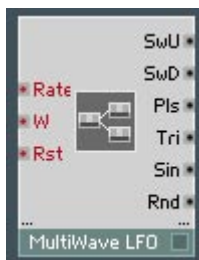
入力オーディオ信号のエンベロープに追従する制御信号を出力します。A 入力、D 入力は、アタック時間およびデケイ時間 (秒単位) を表します。

1.8. Control > Flip Flop



値が 0 と 1 の論理信号を、トリガー入力ごとに交互に出力します。

1.9. Control > MultiWave LFO



位相の揃った、各種の波形の低周波信号を生成します。Rate 入力は周波数 (Hz 単位)、W 入力はパルス幅 (-1 ~ 0 ~ 1、パルス波のみに適用) を表します。Rst 入力にイベントを与えると、その値 (0 ~ 1) に対応する位相から発振が再開されます。

1.10. Control > Par Ctl Shaper



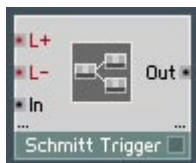
制御信号の振幅を調整するシェイパーで、その特性曲線として、正負によって反転した放物線を使います。入力値の範囲は -1 ~ 0 ~ 1、これに応じて出力値も -1 ~ 0 ~ 1 の範囲になります。曲線を「曲げる (ベンド)」度合いは b 入力で設定しますが、この範囲も -1 ~ 0 ~ 1 です。

b = 0	ベンドなし (線形)
b = -1	X 軸側に最大限ベンド
b = 1	Y 軸側に最大限ベンド

入力値の範囲が 0 ～ 1 であってもこのモジュールを適用できます。この場合、特性曲線のうち半分だけが使われることになります。

主な用途：ペロシティーその他の制御信号の調整

1.11. Control > Schmitt Trigger



入力値が L+ (デフォルト値は 0.67) を上回れば 1、L- (デフォルト値は 0.33) を下回れば 0 に値を切り替えて出力します。

1.12. Control > Sine LFO



鋸波 (上向き) の低周波制御信号を生成します。Rate 入力 は周波数 (Hz 単位) を表します。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

1.13. Delay > 2/4 Tap Delay 4p



2/4 タップのディレイで、4 点で補間を行います。T1 ～ T4 入力 は、各タップの遅延時間 (ミリ秒単位) を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

I.14. Delay > Delay 4p



4 点の補間を行うディレイです。T 入力は遅延時間 (ミリ秒単位) を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

I.15. Delay > Diff Delay 4p



4 点の補間を行うディフュージョン・ディレイです。T 入力は遅延時間 (ミリ秒単位)、Dffs 入力はディフュージョン係数を表します。

遅延時間の最大値は 44100 サンプル分で、サンプル・レートが 44.1kHz であれば 1 秒に相当します。時間の調整には、ディレイ・マクロ内のアレイ長を変更してください。

I.16. EQ > 6dB LP/HP EQ



1 ポール (6dB/oct.) のロー / ハイ・パス・イコライザーです。F 入力はカットオフ周波数 (Hz 単位) を表します。

I.17. EQ > HighShelf EQ



1 ポールのハイ・シェルフ・イコライザーです。dB 入力が高周波成分を増幅する度合いを表します (負値ならば減衰)。また、F 入力は特性曲線が切り替わる中心周波数 (Hz 単位) を表します。

I.18. EQ > LowShelf EQ



1 ポールのロー・シェルフ・イコライザーです。dB 入力は低周波成分を増幅する度合いを表します (負値ならば減衰)。また、F 入力は特性曲線が切り替わる中心周波数 (Hz 単位) を表します。

I.19. EQ > Peak EQ



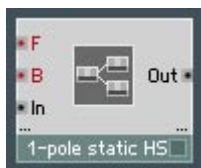
2 ポールのピーク / ノッチ・イコライザーです。F 入力は中心周波数 (Hz 単位)、BW 入力は帯域幅 (オクターブ単位)、dB 入力はピーク部分の高さを表し、これが負であればノッチ・イコライザーということになります。

I.20. EQ > Static Filter > 1-pole static HP



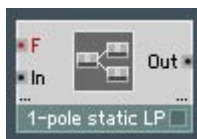
1 ポールのスタティック型ハイ・パス・フィルタです。F 入力のカットオフ周波数 (Hz 単位) を表します。

I.21. EQ > Static Filter > 1-pole static HS



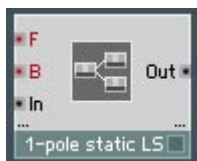
1 ポールのスタティック型ハイ・シェルフ・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、B 入力は高周波成分の増幅率 (dB 単位) を表します。

I.22. EQ > Static Filter > 1-pole static LP



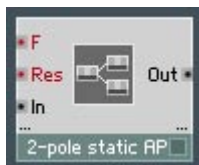
1 ポールのスタティック型ロー・パス・フィルタです。F 入力はカットオフ周波数 (Hz 単位) を表します。

I.23. EQ > Static Filter > 1-pole static LS



1 ポールのスタティック型ロー・シェルビング・フィルタです。F 入力カットオフ周波数 (Hz 単位)、B 入力は低周波成分の増幅率 (dB 単位) を表します。

1.24. EQ > Static Filter > 2-pole static AP



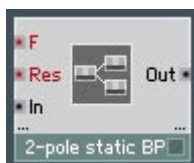
2 ポールのスタティック型オール・パス・フィルタです。F 入力カットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

1.25. EQ > Static Filter > 2-pole static BP



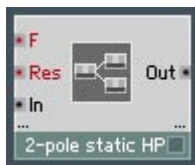
2 ポールのスタティック型バンド・パス・フィルタです。F 入力カットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

1.26. EQ > Static Filter > 2-pole static BP1



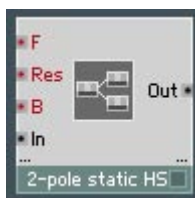
2 ポールのスタティック型バンド・パス・フィルタです。F 入力カットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。カットオフ周波数に一致する信号に対する増幅度は、レゾナンスに関係なく常に 1 となっています。

1.27. EQ > Static Filter > 2-pole static HP



2 ポールのスタティック型ハイ・パス・フィルターです。F 入力のカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

1.28. EQ > Static Filter > 2-pole static HS



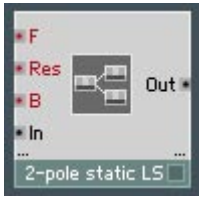
2 ポールのスタティック型ハイ・シェルフ・フィルターです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1)、B 入力は高周波成分の増幅率 (dB 単位) を表します。

1.29. EQ > Static Filter > 2-pole static LP



2 ポールのスタティック型ロー・パス・フィルターです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ~ 1) を表します。

l.30. EQ > Static Filter > 2-pole static LS



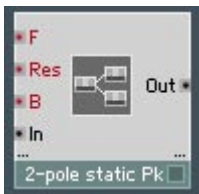
2 ポールのスタティック型ロー・シェルビング・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1)、B 入力は低周波成分の増幅率 (dB 単位) を表します。

l.31. EQ > Static Filter > 2-pole static N



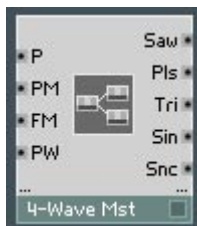
2 ポールのスタティック型ノッチ・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1) を表します。

l.32. EQ > Static Filter > 2-pole static Pk



2 ポールのスタティック型ピーク・フィルタです。F 入力はカットオフ周波数 (Hz 単位)、Res 入力はレゾナンス (0 ～ 1)、B 入力は中心周波数成分の増幅率 (dB 単位) を表します。

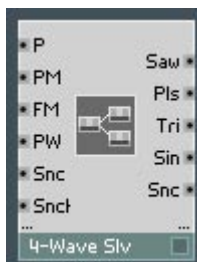
I.33. Oscillator > 4-Wave Mst



位相の揃った、4種類のオーディオ信号を生成します。P 入力が発振信号のピッチで、MIDI ノート番号で指定します。これを PM 入力 (半音単位)、FM 入力 (Hz 単位) で指定される幅だけ変調できます。PW 入力はパルス幅 (-1 ~ 0 ~ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することも可能です。また、**4-Wave Slv** モジュール用の同期出力もあります。

I.34. Oscillator > 4-Wave Slv



位相の揃った、4種類のオーディオ信号を生成します。P 入力が発振信号のピッチで、MIDI ノート番号で指定します。これを PM 入力 (半音単位)、FM 入力 (Hz 単位) で指定される幅だけ変調できます。PW 入力はパルス幅 (-1 ~ 0 ~ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することも可能です。また、他の **4-Wave Mst/Slv** モジュールと同期できます。SncH 入力は同期の度合いを表し、まったく同期しない 0 から完全に同期する 1 までの間で連続的に変化させることができます。さらに、他の **4-Wave Slv** モジュール用の同期出力もあります。

I.35. Oscillator > Digital Noise



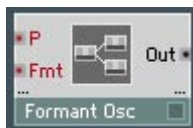
-1 ～ 1 の範囲の値をランダムに取る白色雑音を生成します。Seed 入力にイベントを与えると、乱数を生成する「種」として使われます。

I.36. Oscillator > FM Op



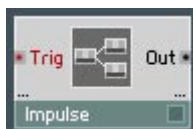
昔ながらの FM 音源です。正弦波を出力しますが、その発振信号のピッチ (MIDI ノート番号) は P 入力で指定します。これを PhM 入力 (ラジアン単位) で位相変調できます。Rst 入力にイベントを与えると、その値 (0 ～ 1) に対応する位相から発振が再開されます。

I.37. Oscillator > Formant Osc



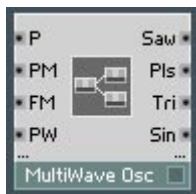
P 入力で指定されたピッチ (MIDI ノート番号) の信号をもとに、Fmt 入力で指定されたフォルマント周波数 (Hz 単位) の領域を強めた信号を生成します。

I.38. Oscillator > Impulse



入力イベントに応じて、振幅 1、長さ 1 サンプル分のインパルスを生成します。

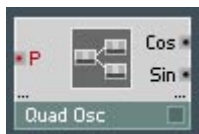
l.39. Oscillator > MultiWave Osc



位相の揃った、4 種類のオーディオ信号を生成します。P 入力が発振信号のピッチ (MIDI ノート番号) を表します。これを PM 入力 (半音単位)、FM 入力 (Hz 単位) で指定される幅だけ変調できます。PW 入力はパルス幅 (-1 ~ 0 ~ 1、パルス波のみに適用) を表します。

周波数として負の値を指定することはできません。

l.40. Oscillator > Quad Osc



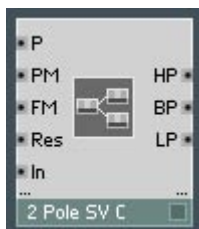
位相が 90°ずれた正弦波の組を生成します。P 入力が発振信号のピッチ (MIDI ノート番号) を表します。

l.41. Oscillator > Sub Osc



基本となる信号をもとに、位相の揃った4種類の分周信号を生成します。P 入力 は 基底ピッチ (MIDI ノート番号) を表します。分周数は S1/S2/S3/S4 入力 で 指定 します (1 ~ 120)。Tbr 入力 は、出力 波形 に 高周波成分が 含まれる 比率を表 します (0 ~ 1)。

I.42. VCF > 2 Pole SV C



2 ポールの状態可変型 (SV、State-Variable) フィルターで、補償 (C、Compensation) の機能があるため、カットオフ周波数としてかなり高い値を指定できます。P 入力 は カットオフ周波数 (MIDI ノート番号) を表し、これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。Res 入力 は レゾナンス (0 ~ 0.98) を表します。レゾナンス値として負の値を指定すると、カットオフ周波数付近をあまり強調しないよう、特性曲線の傾斜が緩やかになります。

HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

I.43. VCF > 2 Pole SV T

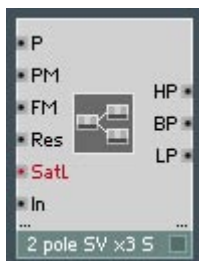


2 ポールの状態可変型 (SV、State-Variable) フィルターで、表にもとづく補償機能 (T、Table compensation) があります。カットオフ周波数が高い場合の性能に優れていますが、**2 Pole SV C** モジュールとは若干異なります。P 入力 は カットオフ周波数 (MIDI ノート番号) を表し、

これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。
Res 入力はレゾナンス (0 ~ 1) を表します。

HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

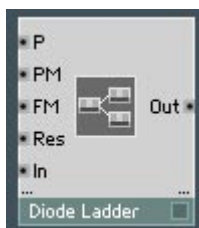
I.44. VCF > 2 Pole SV x3 S



2 ポールの状態可変型 (SV、State-Variable) フィルターで、オーバー・サンプリング (x3 型) とサチュレーション (S) の機能があります。P 入力はカットオフ周波数 (MIDI ノート番号) を表し、これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。Res 入力はレゾナンス (0 ~ 1)、SatL 入力はサチュレーション・レベル (通常は 8 ~ 32 程度) を表します。

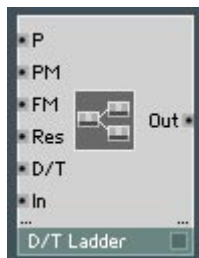
HP/BP/LP の各出力に、ハイ・パス / バンド・パス / ロー・パスの処理を施した信号がそれぞれ出力されます。

I.45. VCF > Diode Ladder



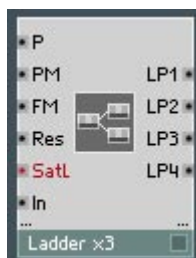
ダイオード・ラダー・フィルターを線形にエミュレートしたものです。P 入力はカットオフ周波数 (MIDI ノート番号) を表し、これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。Res 入力はレゾナンス (0 ~ 0.98) を表します。

I.46. VCF > D/T Ladder



ラダー・フィルターを線形にエミュレートしたもので、ダイオード型からトランジスター型まで、連続的に変化させることができます。P 入力はカットオフ周波数 (MIDI ノート番号) を表し、これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。Res 入力はレゾナンス (0 ~ 0.98) を表します。D/T 入力 (0 ~ 1) は、0 ならばダイオード型、1 ならばトランジスター型を表します。

I.47. VCF > Ladder x3



サチュレーション機能付きのトランジスター型ラダー・フィルター (x3 回のオーバー・サンプル) をエミュレートしたものです。P 入力はカットオフ周波数 (MIDI ノート番号) を表し、これを PM 入力 (半音単位)、FM 入力 (Hz 単位) に従って変調できます。Res 入力はレゾナンス (0 ~ 1)、SatL 入力はサチュレーション・レベル (通常は 1 ~ 32 程度) を表します。

出力 1 ~ 4 は、ラダーの対応するタップから出力される信号です。一般に言うラダー・フィルターの出力信号は、4 番出力から得られます。

索引

記号

* 163
+ 163
- 163
-x 164
/ 164
|x| 164
~BoolCtl 168
~exp 165
~log 165

数字

1+2+3 Shaper 186
1-pole static HP 197, 223
1-pole static HS 197, 223
1-pole static LP 197, 223
1-pole static LS 197, 223
1 div x 173
1 wrap 174
2-pole static AP 198, 224
2-pole static BP 198, 224
2-pole static BP1 198, 224
2-pole static HP 198, 225
2-pole static HS 199, 225
2-pole static LP 199, 225
2-pole static LS 199, 226
2-pole static N 200, 226
2-pole static Pk 200, 226
2/4 Tap Delay 4p 193, 220
2 pi wrap 176
2 Pole SV 212
2 Pole SV C 21, 29, 31, 212, 213,

230, 231

2 Pole SV T 230
2 Pole SV T (S) 213
2 Pole SV (x3) S 213
2 Pole SV x3 S 231
3-1-2 Shaper 186, 216
4-Wave Mst 209, 227
4-Wave Mst/Slv 209, 227
4-Wave Slv 209, 227
6dB HighShelf EQ 196
6dB LowShelf EQ 196
6dB LP/HP EQ 195, 221

アルファベット

A

a-x 87
a - x 179
Accumulator 200
Add 26, 30
ADSR 49, 194, 217
A/E 30
Always Active 66
Amount 35, 36, 37, 43, 181
Amp (dB) 35
Ampl 90
Amp Mod 181
AND 53, 206
arccos 176
arcsin 176
arctan 176
Array 125, 126, 127, 128, 130, 170
Array OBC 125, 126, 170
Audio Amount 185
Audio Core Cell 134

Audio Mix 37, 181

Audio Relay 182

a / x 180

B

Base 165

Binary Noise 209

Bit AND 165

Bit OR 165

Bit XOR 166

Bool C 39, 158, 159

BoolCtl 108, 156

Broken Par Sat 187, 216

Built-In Module 19, 62, 152, 153

C

Chain 37

Chain (+/- amount) 182

Chain Add 176

Chain(amount) 37

Chain (dB) 182

Chain Mult 176

Clip Max 173

Clip Min 173

Clip MinMax 173

Clipping 111

Clk Div 201

Clk Gen 201

Clk Rate 201

Clock 91

Compact Board 10

Compare 108, 156, 167

Compare Sign 120, 167

CONNECTION TYPE 130, 131

Connection Type 170

Connect to New QuickBus 28, 154

Connect to New QuickConst 41, 154

Connect to QuickBus 154

Connect to SR.C 91

Connect to SR.R 91

Const 63, 73, 74, 79, 80, 85, 87, 97,
163

Core Cell 2, 4, 152

Core Cells 3

Core Macros 83

cos 176

cos -pi..pi 176

cosh 177

Counter 201

CRITERION 108

Criterion 167, 168

Ctl2Gate 54, 202

Ctl Amount 43, 44, 188

Ctl Amp Mod 188

Ctl Bi2Uni 189

Ctl Chain 44, 47, 189

Ctl Invert 189

Ctl Mix 26, 43, 44, 189

Ctl Mixer 2 190

Ctl Pan 190

Ctl Relay 190

Ctl XFade 191

Cutoff 107

D

dB2AF 191

Debug Mode 81, 82

Delay 141

Delay 1p/2p/4p 193

Delay 4p 33, 38, 40, 44, 45, 221

Delete Module 21
Denormal Cancel 101, 102, 103, 149
Detect 122, 123
Diff Delay 1p/2p/4p 193
Diff Delay 4p 221
Digital Noise 210, 228
Diode Ladder 20, 214, 231
Distance 139, 140
DN Cancel 101, 164
dP2FF 192
D/T Ladder 214, 232
Dup Flt 202

E

Echo 45
Env Follower 195, 218
EQ 207
ES Ctl 122, 148, 168
Event 11
Event Accum 75, 80
Event Counter 123
Event Processing 54
EvtMerge 169
Expert Macro 19, 152
Expon.(F) 25
Ext 90

F

Flip Flop 206, 219
Float 114
Float モード 117
Flow 78, 108
FM Op 210, 228
Formant Osc 210, 228
FP PRECISION 114, 145

FP Precision 40, 171, 172

G

Gain (dB) 183
Gate2L 53, 206
Gate LFO 53
GE 207
GT 206

H

HighShelf EQ 15, 222
Hyperbol Sat 187, 216

I

Ia - x 179
IClip Max 173
IClip Min 173
IClipMinMax 139, 173
Icon 172
IDup Flt 202
IGT 206
ILatch 119, 151, 177
IMax 174
IMin 174
Imod 174
Impulse 202, 228
In 23, 29, 39, 158, 160
Index 126, 127, 129, 171
INF 103
Input Count 168, 169
Integer 116, 119
Integrator 200
Int モード 117
Invert 183
ISeparator 203

IValue 203

Ix * a 179

Ix + a 178

Ix - a 179

L

L2Clock 207

L2Gate 53, 207

LABEL 23, 27, 98

Ladder x3 214, 232

Large 172

Latch 39, 68, 69, 70, 78, 79, 111,
119, 129, 130, 158, 177

Latch OBC 127, 170, 171

Load... 2, 3, 152

Load Module... 152

Logic 53

Logic AND 33

logT2sec 50, 192

Look 172

LowShelf EQ 222

M

Macro 38, 172

Math 63

Max 174

Memory 75, 78, 93, 142

Merge 77, 78, 79, 80, 106, 110, 111,
112, 122, 148, 156, 168, 169

Meter 64, 123

Min 174

Mixer - Simple - Mono 6

Mixer 2/3/4 183

Mod 47

Modulation 85, 156

ms2Hz 192

ms2sec 34, 192

Multi 2-pole FM 30

MultiWave LFO 204, 219

MultiWave Osc 6, 8, 11, 15, 211, 229

N

NaN 103

New 22, 153

New Audio 16, 18, 152

New Event 16, 152

NOT 208

O

OBC 69

OR 208

Out 22, 39, 158, 160

Owner Properties 26, 39, 115, 152,
153

P

P2F 25, 47, 112, 192

Pan 183

Parabol Sat 40, 187, 217

Par Ctl Shaper 16, 191, 219

Par LFO 42, 204

Par Osc 211

Peak Detector 195

Peak EQ 196, 222

Phase 145

Pin Alignment 172

Pitch 124

PlayPos 140

Port Alignment 136

Precision 165

Properties 11, 23, 26, 29, 41, 45, 63,
80, 115, 142, 145, 152, 153

Q

QNaN 103

Quad Osc 211, 229

QuickBus 28

QuickConst 41

R

Random 202

Random LFO 204

Rate 91

Read 69, 70, 71, 72, 73, 74, 75, 76,
92, 97, 112, 118, 127, 129, 130,
156, 169, 170

RecordPos 138, 140

Rect LFO 32, 54, 205

Ring-Amp Mod 184

round 174

Router 108, 109, 110, 111, 122, 125,
148, 149, 156, 167

R/W Order 130, 131, 170

S

Save As... 83, 153

Save Core Cell As... 3, 152

Saw(down) LFO 205

Saw(up) LFO 205

Schmitt Trigger 208, 220

SE-IV Chorus 7

sec2Hz 192

Sel 135

Separator 203

Show Properties 45

sign +/- 175

SIGNAL MODE 11

SIGNAL TYPE 117, 126

Sign Comparison 122

Simple Scope 89

sin 176

sin -pi..pi 176

Sine LFO 205, 220

Sine Shaper 4/8 188, 217

sinh 177

Sin Osc 31, 211

SIZE 45, 126

Size 170

Small 172

Solid 40, 97, 98, 99, 172

sqrt 175

SR.C 162

SR.R 162

Standard Macro 19, 21, 152

Stereo Amp 184

Stereo Mixer 2/3/4 185

Sub Osc 229

Sub Osc 4 212

T

Table 142, 171

tan 176

tan -pi4..pi4 177

tan -pi..pi 176

tanh 177

Tape Delay 40

Thld Crossing 203

Thru 95, 98, 99

Time 142

Tri LFO 206

U

User Macro 84

User macro 152

V

VALUE 41

Value 54, 65, 163, 203

VCA 185

VCF 20, 21

W

Wrap 140

Write 69, 70, 71, 72, 74, 75, 76, 78,
92, 97, 112, 121, 122, 127, 129,
130, 131, 156, 169

X

$x * a$ 179

$x + a$ 178

$x - a$ 179

x / a 179

$xa + y$ 180

XFade (lin) 185

XFade (par) 34, 186

$x \text{ mul } a$ 87

XOR 208

Z

Z^{-1} 71

Z(帰還の表示) 92

かな

い

イベント型 15

イベント 55, 155

イベント型コア・セル 61

イベント出力 62

イベント信号 48

イベント入力 62

イベント累算器 75

お

オーディオ / イベント・ポート 10

オーディオ型 15, 88

オーディオ・クロック 106

オーディオ出力 88

オーディオ信号 30, 88

オーディオ入力 88

オブジェクト・バス接続 69, 116, 156

き

帰還 91, 95, 119

く

クロック信号 68, 156

け

経路制御 (イベント) 108

こ

コア信号 55

コア・セル 1, 2, 88, 152

コア・マクロ 38, 39, 83

コア・モジュールの追加 19

コア・レベルのストラクチャー 8

さ

作成 (コア・セル) 16

作成 (ポート) 22

サンプリング周波数 90

サンプル・レート 55, 69, 90

サンプル・レート・クロック 92

し

条件処理 108

初期設定 72, 155

初期設定イベント 73, 79, 85, 89, 104,
122

処理順序 (コア・モジュール) 60, 70

信号 30, 155

す

スレイブ 69

せ

制御信号 30

整数値 116, 150

整数値モード 117

そ

送出順序 (イベント) 62

て

デバッグ・モード 81

と

同時イベント 58

な

名前変更 (コア・セル) 26

名前変更 (コア・マクロ) 39

に

入力 (デフォルト信号) 41

ひ

非正規数 99

ふ

フィードバック 91

符号比較 120

浮動小数点数値 114, 150

浮動小数点数値モード 117

へ

編集 (コア・セル) 8

変調マクロ 85, 89, 147

変調マクロ (整数値用) 119

ほ

ポート 8

ポート (コア・セル) 22

ま

マージ 76

マスター 69

め

メモリー書き出しモジュール 69

メモリー読み込みモジュール 69

ゆ

ユーザー・ライブラリー 3, 83

ら

ラッチ 68, 106, 147, 156

る

累算器 75

ろ

論理信号 52

