「ハードウエア設計コンテスト」課題発表（信号発生システム部門）

カオス・ジェネレータをつくろう

基本原理と実現へのヒント

長嶋洋一

先月号で発表した「ハードウェア設計コンテスト」の課題の一つとして，いま話題の「カオス」を取り上げる．物理現象を含む世の中のいろいろな現象は，ある(物理)法則のとおりに確定して起こるわけではないことは，昔から知られている(異常気象，株価の暴落，大事故の連続性，…)．これは，従来，雑音や要素項目の複雑さによる誤差などに起因するとみられ，これらを確率的に論じることにとどまっていた．これらの不確定性を数学的なモデリングで説明しようというのが，カオス理論である．ここでいうカオスとは「まったくの混沌」ではなく「何か秩序ある混沌」であり，「なんらかの周期性をもつ混沌」なのである．ここでは，カオスの紹介を簡単にしたあと，「カオス・ジェネレータ」開発のヒントを示している．（編集部）

1 カオスとは

▶キーワードとしてのカオス

家電製品のセールス・ポイントとして，そのときどきの新しいテクノロジが商品名に冠されることは多い．たとえば，「エアコン」「洗濯機」「掃除機」「炊飯器」などに「AI」「ニューロ」「ファジイ」といった枕言葉が付された製品が，ほとんどすべての電気メーカで一斉に発売され，やがてはブームの一巡とともに消え去って行く．そして今，これらに続くキーワードとして「カオス」がぼちぼち登場してきた．そこで，「ハードウェア設計コンテスト」の課題の一つとして，この「カオス」を身近に体験できるようなチップの設計にチャレンジしていただくことを提案する．

ここではまず，カオス(Chaos，発音は「ケイオス」，邦訳では「混沌」)とはどのような概念であるのかを，ごく簡単に解説する．研究者の世界での「カオス情報処理」については，まだ未知の領域も広がっている現在進行中の分野であるが，いろいろな書籍(後述)での入門的な解説も数多く入手できるので，数学的に深入りせずに，ここではごく一部についての紹介にとどめる．

▶自然界での状態記述

カオスは，「自然界の状態の記述」に用いる．あまりにも抽象的で漠然とした言い方に戸惑われるかもしれないが，逆に言えば，カオスとは特別な場面に現れるものではなく，自然界(ここでは時間とともに変化する「量」「状態」)のどのような局面にも登場することを強調したいのである．

カオスの解説書にあるように，いろいろな科学者が調べてみると，カオス現象というのは別に物理的・数学的なものではなく，たとえば，

- 神経細胞のパルスのゆらぎ
- 補食関係にある動物の個体数の変化
- 経済市況データの時間的変化

などの現象にもみごとに発見されている．そして，このカオスは，「新パラダイム」としていろいろな応用が注目・期待されているのである．

2 状態の記述

さて，そこでこの「状態」を $X(t)$ としよう．ここでの $X(t)$ とはもちろん，時間 t に関連して変化する量 X というだけのことで，これが物理的な量として，

- 物体の運動を表す位置座標
- 回路のある部分を流れる電流
- 物体のある部分の温度や圧力

などに自由に適応できるのである．つまりカオスを検討する対象が物理量であれば，応用範囲はこれまでの物理学のすべての物理量でありうるのである．

また，より抽象化された数学的なアプローチでは，$X(t)$ という「量」がまったく自由に規定できるのも明らかである．

▶時間変化を微分方程式で表す

数学的な立場からは，時間 t という量を特別扱いする必要はなく，多くのパラメータのたまたま一種でしかないが，ここではカオス現象をイメージしやすいように，時間 t を特別に扱い，他の物理量の「時間的な変化」としてカオスを考えることにする．

物理量 $X(t)$ が時間 t とともに変化すれば，これは微分方程式による表現，

$$\frac{dX(t)}{dt}=f(X,t)$$

によって扱うのは多くの人が知っていることだろう．

たとえば，位置 $x(t)$ の時間微分が時間 t に比例しているだけの場合，比例定数を g とすると，微分方程式は，

$$\frac{dx}{dt}=gt$$

となり，この式を変形して両辺を積分するという定石的な計算から，

$$x(t)=\frac{1}{2}gt^2+C \qquad (C \text{ は定数})$$

という，よく知られた「自由落下運動の公式」が得られる．

この例では，$X(t)$ の微分表現の右辺が時間 t だけであったために，計算はかなり簡単なものでよかったが，一般には右辺に $X(t)$ そのものも関係していることが普通である．

たとえば，質量 m の物体の位置 $x(t)$ に比例定数 k で比例する「復元力」が働く1次元の運動方程式は，

$$m\frac{d^2x}{dt^2}=-kx$$

と表され，これを解くと，二つの定数 m と k と初期状態によって，

(1) 一定周期・一定振幅で無限に振動しつづける
(2) 減衰振動でしだいに(指数関数的に)振幅がゼロに向かう
(3) 振動せずに急速に減衰してゼロに向かう

という，三つの状態のいずれかとなる．

これは，CR 時定数回路による電気信号の振動とまったく同じものである．この場合，減衰していれば時間 t が十分経過したときの状態は「ゼロ」であり，振動している場合でも，時間 t を与えればその時点での位置 x は「厳密に」求めることができることを確認しておこう．というのは，後述するとおりカオスは，これと対照的に，ある時間を与えても状態を厳密に求めることができない(一種のランダム状態)のである．

3 カオスの記述

▶数式表現の準備

さて，カオス現象の数式表現に向けて，もう少しスッキリとさせていくことにする．

まずは「正規化」である．いろいろな物理量や概念を同じように扱うために，ここでは「量」$X(t)$ を，0から1までの範囲に正規化する．これはたんにデータの範囲を共通の物差しで表現するための処理である．たとえば+5Vの単一電源の回路であれば，扱う「電圧」を+5Vで割っておく，といった程度のものである．

また，たとえば無限に続く空間を扱うのはかなり面倒であるが，もともとカオス現象は「発散」してしまう場合には起きず，基本的に「ある種の繰り返し」の現象に発生する．そこで，繰り返しの最大振幅(あるいは収束する極限値)を0と1にするように「正規化」することで，いろいろなカオス現象のアナロジ化にも役立つ．

▶促進因子と抑制因子をもつ微分方程式

カオス現象は，概念的には「促進因子」と「抑制因子」の相反するように見える二つが関係しているような系に発生する．これをもっとも単純に表現した微分方程式は，

$$\frac{dX}{dt}=kX(1-X)$$

となる．つまり，ある瞬間の「状態 X の変化のしかた」は，その時間の X 自身に関する成分と，$(1-X)$，つまり全体から X を除いた成分との積に比例する，というものである[注]．

これは，とくに作為的な関係というわけではない．たとえば「生物の増殖」を考えると，

(1) 現在，存在している生物(親)が多ければ，次世代の生物(子)が多く発生する
(2) 環境によって生存できる総数は限りがあり，リミットから現在の総数を引いた量「生存しやすさ」が大きいと，次世代の生物が多く発生する

という相反する二つの側面があり，これはまさに上にあげた方程式そのものである．このような促進因子・抑制因子がともに関係する現象としては，この「人口の推移」だけでなく，

●植物の年輪に見られる成長変化の特性

注：ここでは省略するが，興味のある読者は上の微分方程式をいろいろな条件で解いてみてもいいだろう．じつは，この式の k の値によってカオス状態が得られる(次節を参照)．

●快適なエアコンの温度制御・管理
●興奮と鎮静の両方のホルモンからなる人体の生理的機構
●「売り」「買い」が交錯する経済市況
など，自然科学だけでなく社会科学においても数多く指摘されており，現実に株価取り引きのプログラムにカオス理論がすでに適用されているのである．

▶離散化と量子化 ── 漸化式による記述

さて，ここでアナログ量である微分方程式を離れて，もっと簡単にカオスを体験するためにディジタルの世界に移行することにする．数学的にはまったくこれまでの議論を外れることなく，さきの微分方程式は，

$$X(n)=f(X(n-1))$$

という漸化式に置き換えることができる．これは，「状態」あるいは「量」を連続した時間 t から「n 番目のもの」と離散化して規定しただけのことである．考えてみれば，微分方程式で連続して表してきた量といっても，
●前世代のある生物の個体数 → 次世代の個体数
●昨日の株価 → 今日の株価
●地球上のある地点の昨日の気温 → 今日の気温
といった，実際にはもともと離散している時間軸をもつものも非常に多いのである．

このように数式表現を漸化式にすると，技術的には非常に扱いやすくなる．ディジタル信号処理でおなじみのように，たとえばコンピュータ・プログラムとしては，微分方程式を近似式で処理するよりも，アルゴリズムとして離散的な状態を次々に「ステップ」として計算するほうが簡単である．また，ハードウェア化した場合にも，ディジタル同期システムのシステム・クロックごとに，次々とデータ演算を進めるだけでよい．

ソフトウェアでいえば，漸化式をあらわす関数 f(x) を定義しておけば，初期値 data1 に対して，

```
data2        = f(data1)
data3        = f(data2)
data4        = f(data3)
  ⋮            ⋮
data(n)      = f(data(n-1))
data(n+1)    = f(data(n))
  ⋮            ⋮
```

と，いくらでも単純に計算を続ければよい．円周率の計算のように桁があふれることもないので，家庭のパソコンで夜中に勝手に走らせておけば，いくらでも望むだけの回数のシミュレーションを個人で実験することが可能なのである．

▶データの精度について

なお，アナログでないカオスの計算において重要なことに「データの精度」の問題がある．後述するように，8 ビットや 16 ビット程度の量子化精度では，カオス現象にともなうフラクタルな構造(あとでじっさいのフラクタル構造を示す)の演算において，無視できない誤差が発生する．

このために，たとえばパソコンでシミュレーションする場合であれば，言語処理系ごとに最大精度のデータ型を利用することが大切である．ハードウェア化する場合も同様で，誤差成分を補間するなど数値精度を上げる工夫が効果を上げることになる．

▶多次元化

本稿では簡単のために 1 次元の表現に限定したが，たとえば変数を X(n) と Y(n) の二つとして，漸化式を，

```
X(n) = f(X(n-1), Y(n-1))
Y(n) = g(X(n-1), Y(n-1))
```

と連立させたような系に，簡単に拡張できる．これは，2 次元平面に対応するデータを表示して視覚的に現象をとらえられるために，多くのカオス現象でいろいろな漸化式の美しい「図形」が報告されている．

4 カオスを体験する

▶カオス体験ソフト

それでは，ここでもっとも簡単な実例によって，実際に「カオス」を体験してみよう．

リスト 1 (pp.175-176) に示した C 言語プログラムは，筆者が日曜大工的に製作した，メイの漸化式を使用した「カオス体験ソフト」である．同じような処理を繰り返したり，禁断の goto 文があったりと，まったく美しくないプログラムではあるが，PC-9801 とマウス (CONFIG.SYS に DEVICE＝MOUSE.SYS の 1 行を追加登録しておくこと)によって，任意の場所をズーミングしながら，カオス状態への移行と，その周辺のフラクタル構造を自由に体験できるものである[注]．

▶メイの漸化式

ここで使われている漸化式(Logistic Function)は，

$$X(n) = \mu \cdot X(n-1) \cdot (1-X(n-1))$$

というもので，「メイの漸化式」と呼ばれている．漸化式の右辺に，パラメータとして μ という値を掛けた

注：このソース・プログラムは MS-C ver 5.1 でコンパイル可能．また NIFTY-Serve の人工知能フォーラム(FAI)のデータ・ライブラリには，Sysop 氏が美しく書き直してくれた改訂版がアップされている．

だけである．もちろん，この式はカオス状態を示すもっとも基本的で単純なモデルを示すもので，実際に提案されている漸化式はいろいろある．

このプログラム(MAY_ZOOM.EXE)を実行するとまず画面に描かれるのは，

●横方向：μ の値(左端が 1.0，右端が 4.0)

●縦方向：$X(n)$ の値(下端がゼロ，上端が 1.0)

での，この漸化式の振る舞いである．

具体的には，横軸のそれぞれの μ の値に対して，

(1) 初期値として，$X(1)=0.5$ をとる

(2) そこから 100 回，漸化式にしたがって，つぎつぎと $X(n)$ を求める

(3) 101 回目から 300 回目までの 200 ポイントについて，$X(n)$ の値をプロットする

という処理を行っている．

▶メイの漸化式の振る舞い

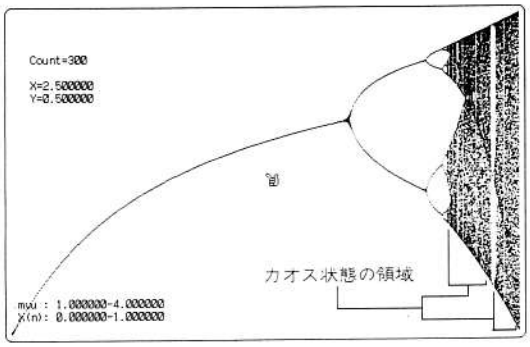
実際のプロットの様子は，つぎのようになる(図 1)．

(1) まず，画面左下からなめらかに上昇する曲線が，$\mu=3.0$ まで，たった 1 本の曲線としてつながっている．これは，200 回のプロットがすべて同じ値となっている，すなわち漸化式による計算結果が一定の値に収束していることを意味している．

(2) そして $\mu=3.0$ を越えると，不思議なことに曲線が 2 本に分岐する．これは，200 回のプロットにおいて，上の曲線と下の曲線の値の 2 値に収束して交互にプロットしている(振動している)もので，μ の値の増加とともに，しだいにその間隔が増大していく．

(3) さらに μ の値が増加すると，2 本の曲線はそれぞれが 2 本ずつ，つまり 4 本に分岐する．そしてしだいに分岐が繰り返され，$\mu=3.6$ を越えたところで，画面が塗りつぶされたような領域になる．ここが「カオス」状態の領域である．つまり，プロットするごとに $X(n)$ の値は毎回異なったものとなるために，収束して 1 本の曲線とならないのである．

〔図 1〕メイの漸化式の振る舞い(MAY_ZOOM の実行)



(4) そこから $\mu=4.0$ までずっとカオスであるかというと，そうではない．ところどころに，「窓」と呼ばれるいくつかの収束値だけの領域があり，よく見るとこの「窓」はかなりたくさんあることがわかる．「窓」の周辺の振る舞いは，現在も研究されているところである．

▶ズーミングによる探索

このソフト(MAY_ZOOM.EXE)は，マウス対応である．画面に表示されているうち，ある部分を画面全体に拡大して再描画するための方法は，

(1) **ズーミングしたい領域の「左下」**にマウス・カーソルをもっていく

(2) マウスの左ボタンをクリック(縦/横に線が出る)

(3) **ズーミングしたい領域の「右上」**にマウス・カーソルをもっていく

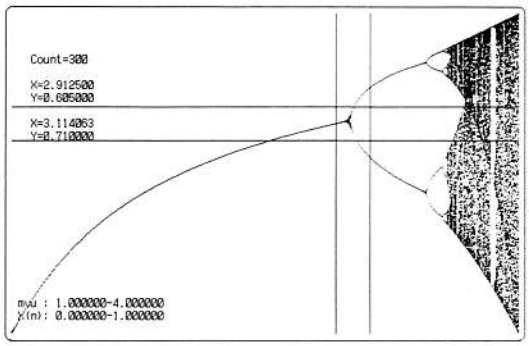
(4) マウスの左ボタンをクリック(縦/横に線が出る)

(5) 拡大したときのプロット数を多くしたければ右クリック→数字変更)

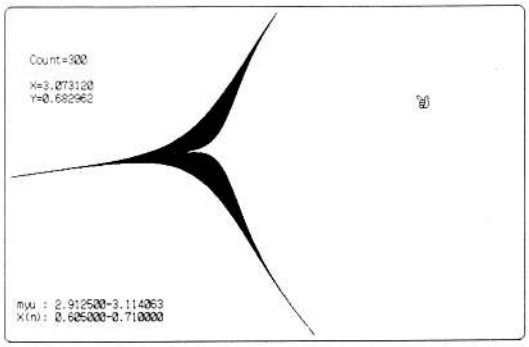
(6) マウスの左ボタンをクリック

というもので，すぐに拡大画面が描画される．コプロセッサがあれば数秒である．また，領域全体を最初の画面に戻すためには，1 回目のクリック・ポイントよ

〔図 2〕ズーミングする



(a) ズームする位置を決める



(b) 拡大表示

りも左下にカーソルを置いてクリックする．プログラム終了はESCキーである．

この方法によって，たとえば**図1**の1本の曲線が2本に分岐する部分を拡大してみよう．なめらかな曲線が組み合わさっていることがわかる(**図2**)．また，別の場所のもっと細かい分岐の部分を拡大してみると，「同じような構造」が繰り返されていることがわかる．これが話題の「**フラクタル構造**」(拡大しても拡大しても同じ構造が繰り返されている［入れ子構造］)である．「カオス」の振る舞いの基本には，この「フラクタル」がある．この両者は別のものではなく，同じ概念にもとづく，ちょっとだけ違った現象なのである．

ズーミングの対象を，全体画面で顕著に見える，大きな「窓」の中に変えて，またあちこちを探索してみよう．カオス・ゾーンから1本の細い「ひげ」が「窓」に出てくる部分も，拡大してみると非常に美しい．また，「窓」の中の分岐はまたまたフラクタルで，他の部分とまったく同様になっている(**図3**)．

ズーミングのエリアとして，縦方向はほとんど画面全体のまま，横方向だけ非常に狭く指定して(大きく拡大して)みるのも面白い．それまでカオス・ゾーンとなっていると思っていた領域にも，また細い「窓」があることもわかる．いずれにしても，いろいろな場所に，いたるところに「フラクタル構造」を確認できる．

▶カオスの世界の広がり

NIFTY-ServeのAIフォーラム(FAI)には，このような単純な1次元カオスだけでなく，2次元表示オール・カラーのいろいろなソフトがアップされている．2次元カオスの美しい世界に興味のある読者は，ぜひこちらも体験してほしい．

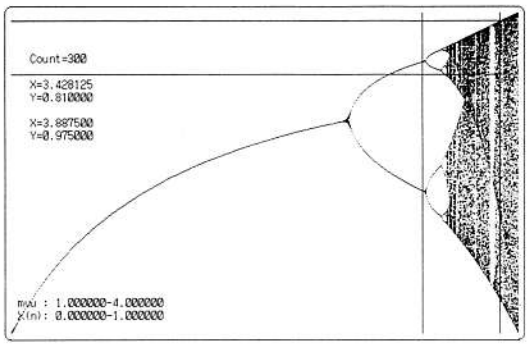
また，上のMAY_ZOOM.EXEでは，心臓部となっている漸化式は，

```
y_pt＝myu＊y_pt＊(1.0－y_pt);
```

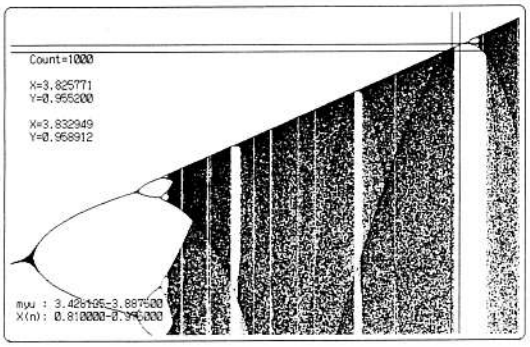
という，たった1行だけである．この漸化式をいろいろに変えてみると，カオスの状態はまた変化する．また，初期値やプロット条件によっても大きく変わる要因がある．筆者は整式だけでなく，sin関数や指数関数なども試してみた．ここから先は，まったく実験する者の自由である．まだ誰も発見していない面白い現象が隠れているかもしれないのである．

このように，カオスのシミュレーションでは，高性能パソコンやコプロセッサがなくても，それなりに「面白い何か」を発掘できる可能性がある．

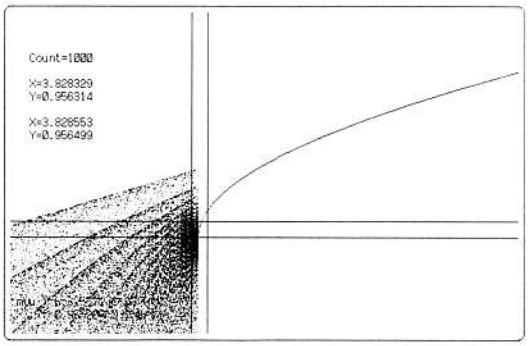
〔図3〕ズーミングの例



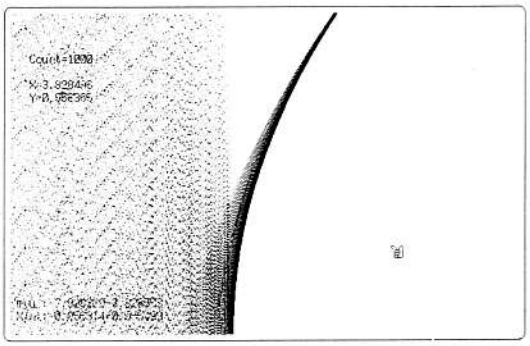
(a) 別の場所をズーミングすると



(b) フラクタル構造が見える．さらにズーミングすると



(c) カオス状態が大きく変化する部分をさらにズーミングすると



(d) たつまきのようになっている

5 カオス利用の一例

▶音楽へのカオス応用

つぎに「カオス利用」の一例として，筆者の創作・研究での利用について紹介する．一言でいえば，「カオスを利用した作曲」である．まずその原理を紹介する．

「メイの漸化式」の1次元カオス動作を解析してみると，あるパラメータ領域を拡大した中に，より大きなスケールの構造と類似した構造を発見できる．これはフラクタル構造の本質である「自己相似性」であり，よく知られたマンデルブロ集合やジュリア集合と同様に，拡大しても拡大しても同じ構造を繰り返す．

ところで音楽においても，これと同様の階層性と自己相似性は指摘されており，たとえばバルトークのように，自己相似性を数学的指導原理として，フィボナッチ数列と黄金分割を基本に作曲した作曲家もいて，フラクタル構造の数学的性質は作曲の一つのテーマとなっている．単純な乱数による「サイコロ音楽」や単調な繰り返しによる「ミニマル・ミュージック」に決定的に欠けている，音楽の全体的構成や階層構造を実現するための一つの可能性としてフラクタルやカオスがComputer Musicの分野で注目される理由の一つは，この自己相似性である．

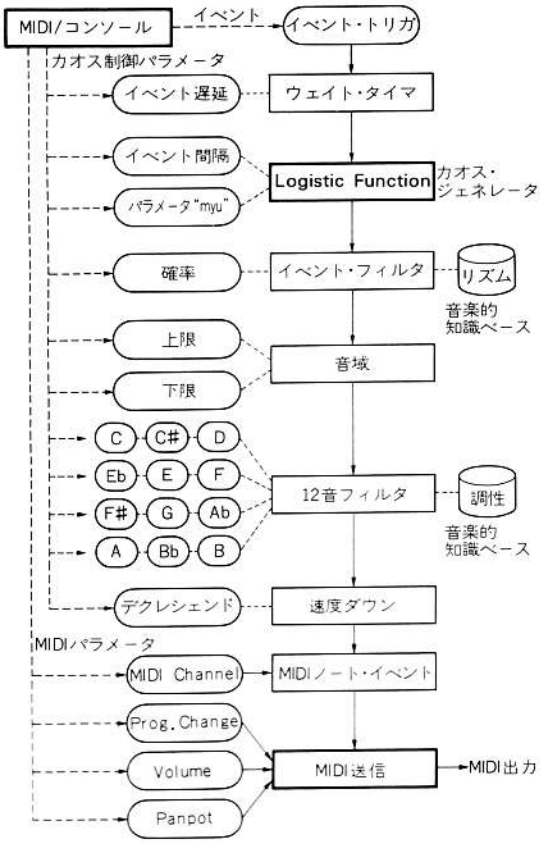
▶ X(n)をMIDIのノート・ナンバに割り当てる

そこで，もっとも単純な例として，この$X(n)$の値をMIDIのノート・ナンバ(音程を示す番号)に割り当てると，周期的に変化する一種のフレーズに対応させることができる．この場合，2周期であればトレモロ的なメロディ，4周期であれば装飾音的なメロディになり，さらに6周期や9周期の聴取実験によると，変拍子系のビートやシャッフルのビートにも移行しうるのである．また，カオス出力に対する音程を与えるための12音生成確率のフィルタを調整することで，古典的和声の枠組みにしたがった「分散和音」とすることもできる．

さらにカオス・パラメータを微小に変化させていった場合，同じ2周期でも二つの収束点の値がしだいに変化するために，あるメロディからの「変奏」のように知覚され，非常に長い周期をもったメロディを高速で演奏させることで，一種のジャズの「アドリブ」的なフレーズとも知覚できる．

これらは，従来の単純な乱数生成系では見られない特徴であり，音楽の基本的要素である「時間的繰り返し」という特性にカオス動作の周期性を対応させ，さらにパラメータを変化させて周期状態を変化させる可能性を示している．

〔図4〕製作したカオス・ジェネレータのブロック図



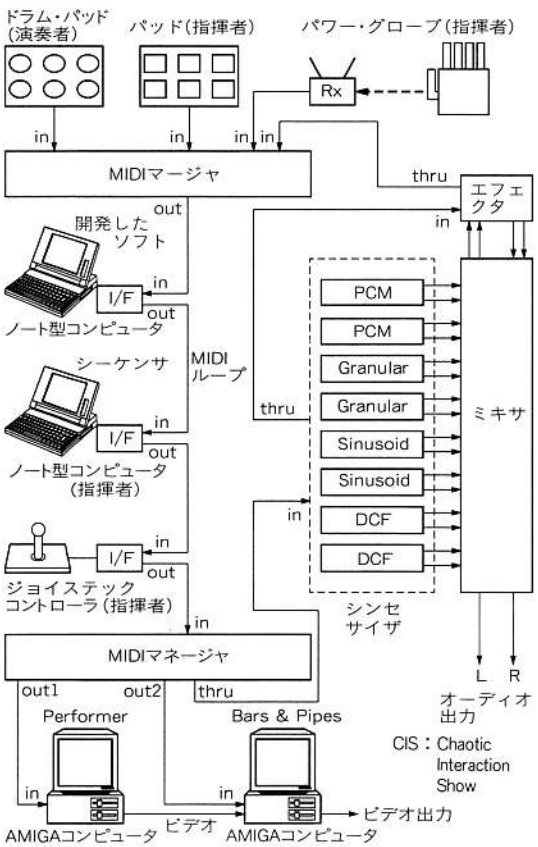
▶音楽用カオス・ジェネレータ

このカオス系による「自動作曲」の具体的な実現として，ある作品のためにリアルタイムにカオス状態を変化させる音楽生成ソフトを製作した(このソフトウェア開発は，筆者にとって「作曲の一部」である．)．図4のブロック図は，このソフトウェアの一部(同時に動作している独立8系統のうちの一つ)である．

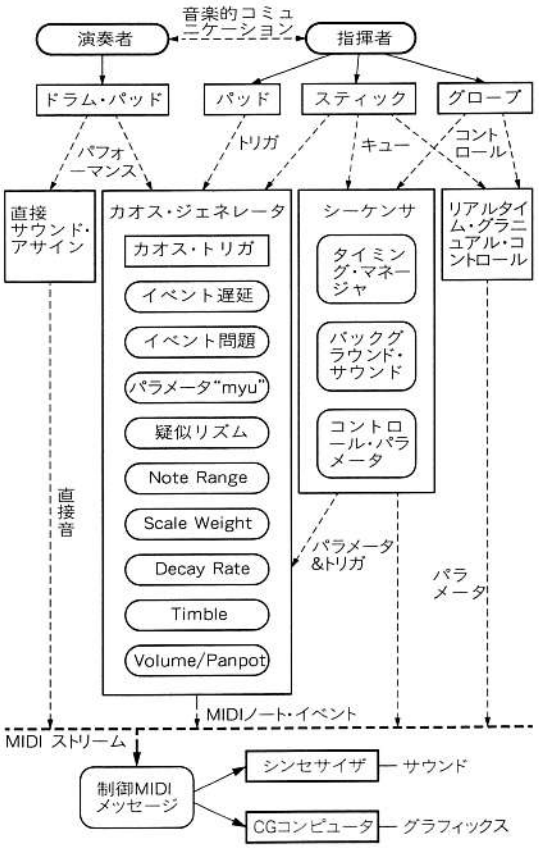
すべてのパラメータとトリガは，パソコンのキーボード・マウスとMIDIからコントロールされ，音楽の中ではシーケンス・データと演奏者からのセンサによって与えられる．発音イベントは遅延パラメータを経て，リアルタイムに演算されているLogistic Functionのブロックでカオス状態ないし多分岐振動の倍精度実数データを供給する．

このデータは，疑似リズムを生成するイベント・フィルタによって「発音」「休符」の選択を受け，最高音・最低音の音域指定パラメータと乗算される．そしてさらに，12音それぞれに別個に指定される「音程出現確率」データの重み付け乗算を受けて，実際に対応するMIDIノート・ナンバが得られる．さらに「デクレッシェンド」パラメータによる減衰データをMIDIベロシ

〔図5〕CISのシステム・ブロック図



〔図6〕CIS制御メッセージのフロー



ティとして与えて，最終的なMIDI音源群へのノート・イベントとして出力とされる．

音楽の中では，それぞれのパラメータは刻々と変化し，また演奏者からのトリガはCGモニタを見ながらの即興に任されているために，演奏時に初めてその作品の音楽要素が決定(作曲)される．

▶具体的なシステム構成

図5はシステム・ブロック図である．音楽と映像に関するすべてのリアルタイム情報は，MIDI信号として与えられ，システム内には特別に定義されたプロトコルのMIDIが一種のLANとして配置された．

ステージ上には，MIDIドラム・パッドを演奏するパーカッション奏者と指揮者が立ち，指揮者はMIDIコントロール・パッド，ジョイスティック・コントローラ，ワイヤレス・パワー・グローブによって，特別に定義されたMIDIコントロール情報を送る．両者の足元にはビデオ・モニタが置かれ，ステージ後方の大スクリーンにプロジェクタで投射されたCGと同じ映像を見ながら，さらに音響を聴きながら，即興によって「演奏」を行った．

2台のノート・パソコンのうちの1台は，作品の一部であるオリジナルのソフトが走り，もう1台は通常のMIDIシーケンサで，BGM部分の演奏情報とシステム・パラメータが記述されている．BGM部分の情報はごくわずかであり，大部分の音楽要素(音列・リズムなど)はステージ上でリアルタイムに生成(作曲)された．音源群は通常のPCM音源・DCF音源に加えて，オリジナル音源として，Granular Synthesis音源とSinusoid音源がそれぞれ2台ずつ使用された．

CG系も同じようにMIDI情報によってコントロールされ，2台のAMIGAコンピュータが2種類のソフトを使用した．前者はパーカッション奏者のトリガによって生成される一連のアニメーションの描画に使用され，後者は背景映像として刻々と変化する画面に使用した(CGの製作とCGシステムは共同作業を行ったCGアーティスト：由良泰人による)．

図6は，この作品のコントロール・メッセージを示したフロー図である．パーカッション奏者の演奏するドラム・パッドに内蔵されている音源は使用せず，すべての情報はMIDI化されてシステムに取り込まれ，パーカッション音としてアサインされる場合でも，オ

リジナル・ソフトによって刻々と音色の割り当てを変更した.

6 カオスの応用分野

さて，いろいろとカオスについて紹介してみたが，カオスの応用については，前述の「音楽生成」のような「生成系」だけでなく，シミュレーションや分析の分野でもいろいろに紹介されている．また，ノイズ中の信号のパターン認識への応用とか，人工知能システムでの「高次記憶」「推論機構」「論理の飛躍」への応用なども検討されている．

筆者の印象では，「カオス」は「ニューロ」や「ファジイ」よりもかなりコンパクトなシステムで容易に実現できるために，アマチュアにとっては敷居が低い，それでいて可能性を秘めた面白い分野であると思う．今回のチップ製作企画とは離れても，いろいろと挑戦できる世界なのである．

7 チップ化の視点と条件

▶ゲート規模の条件

後述するように，今回の「ハードウェア設計コンテスト」の課題部門では，原則としてFPGAで実現を考える場合，5000ゲート・レベル(使用可能レベルで2500ゲート)という条件が最大のボトルネックとなる．実効2500ゲートというのは，思ったほど大きな規模ではない．そこで，ヒントをまず述べておこう．

たとえば，CPU周辺のいろいろな機能を盛り込んだ「便利LSI」は500ゲートもかからずに実現できるが，カオス生成では高精度な演算能力が必要になる．アルゴリズムの自由度もほしい，などと希望を盛り込むと，あっという間に10000～20000ゲートのシステム規模になりそうである．したがって今回のゲート数の条件は，

「いかにコンパクトに実現するか」

のテクニックが問われているものと言える．

(1)「カオス生成システムの中核LSI」構想

豊富にゲートが余っているわけではないので，せっかくのゲートを「ホストCPUインターフェース回路」などの周辺サービス機能に割かれるのはもったいない．つまり，この1チップだけで「システム全体」をすべて実現する，という考えでなく，「システムの中核はこのチップで行う」という割り切り方である．

たとえば，コスト的に安価な「メモリ」「出力ラッチ」「クロック回路」など，場合によっては「乗算器」までをLSIチップ外に出してしまって，このチップを中核に置いた1枚のコンパクトなボード全体として，高い能力のカオス生成システムを実現するのだ，という方針である．

(2)「並列処理対応」構想

1チップのゲート数が限られている反面，このチップはFPGAやゲートアレイとして多量に安価に提供されることを前提として設計される．つまり，システムとしてこのチップを多数使用して，「並列処理システム」によって高い性能を実現できるように設計されていれば，問題点はクリアされるのである．

並列・分散処理については，「時間的な分割」「処理の分割」「ビット幅の分割」などいろいろなアプローチがある．しかし一方で，全体として同期・協調するためのアルゴリズムやメカニズムが必要になる．

(3) 外部メモリの活用

この規模では，オンチップのメモリはほとんど無意味の発想である．したがって，必要に応じて外部に置かれたメモリをいかに活用するか，が一つのテクニックとなる．メモリはRAMに限らず，処理に必要なデータをあらかじめ用意したROMを配置して，「テーブル参照」テクニックでゲート数の削減や演算処理の高速化をはかるのは定石となるかもしれない．

(4) 時分割多重化

アナログ的な「カオスIC」は，すでにJRCなどから出ている．せっかくディジタル化するのであれば，時分割多重化のテクニックによって，あまり変わらないゲート数で多チャネルのカオス信号を同時に生成することが「付加価値」となりうる．うまくシステム設計することで，ほとんど回路規模を増大させずに，同時に数チャネルのカオス出力を生成することもできる．

(5) DSP化の構想

チップ上に漸化式やアルゴリズムを固定してしまう代わりに，最近の汎用DSPのように，内部にマイクロプログラム・メモリ(RAM)とアドレス・シーケンサをもって，マイクロプログラムを外部からロードして，自在に演算処理を決定できるようにする方法．

この場合，たとえばパソコン側でマイクロプログラムの設計支援とかコンパイラを作ることにもなるが，それは「可能性」をもたせておけば，具体的に製作しなくてもかまわないであろう．要は，カオスに興味のあるソフト屋を「その気」にさせるハードウェアが提供されることにあるのだから．

●カオスを理解するための参考文献

1)合原一幸，『カオス　まったく新しい創造の波』，㈱講談社，1993

2) 合原一幸，『カオス ── カオス理論の基礎と応用』，サイエンス社㈱

3) R.L. Devaney(後藤 訳)，『カオス力学系入門(第2版)』，共立出版㈱

ながしま・よういち　ASL長嶋技術士事務所

〔リスト1〕
メイの漸化式

```
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <dos.h>
#include <math.h>

static    union REGS inregs,outregs;
static    struct SREGS segregs;
int       moux,mouy,right,left,old_l,old_r,old_x,old_y;

mouse(flag) int flag; {
        if(flag==0){inregs.x.ax=2;} else if(flag==1){inregs.x.ax=1;}
        int86(51,&inregs,&outregs); }
getmouse(right,left,x,y) int *right,*left,*x,*y; {
        inregs.x.ax=3; int86(51,&inregs,&outregs); *right=outregs.x.bx;
        *left=outregs.x.ax; *x=outregs.x.cx; *y=outregs.x.dx; }
setmouse(x,y) int x,y; { inregs.x.ax=4; inregs.x.cx=x; inregs.x.dx=y;
        int86(51,&inregs,&outregs); }
m_style(x,y,buf) int x,y; char *buf; { inregs.x.ax=9; inregs.x.bx=x;
        inregs.x.cx=y; segread(&segregs); segregs.es=segregs.ds;
        inregs.x.dx=(int)buf; int86x(51,&inregs,&outregs,&segregs); }
m_view(xmin,ymin,xmax,ymax) int xmin,ymin,xmax,ymax; {
        inregs.x.ax=16; inregs.x.cx=xmin; inregs.x.dx=xmax;
        int86(51,&inregs,&outregs); inregs.x.ax=17; inregs.x.cx=ymin;
        inregs.x.dx=ymax; int86(51,&inregs,&outregs); }
m_color(color) int color; {
        inregs.x.ax=18; inregs.x.bx=color; int86(51,&inregs,&outregs); }
symbol(x,y,color,str,skip) int x,y,color,skip; unsigned char str[80]; {
        int i=0; _setcolor(color);
        while(str[i]!=0){ _gputchar(x,y,str[i++],_GXOR); x += skip; } }
static int style[]={
        0,0,0,0,0,0,0,0,0x0070,0x0e88,0x1184,0x2142,0x2221,0x9210,
        0xe107,0x1108,0x1108,0xe107,0x1108,0x1108,0xe107,0x4108,
        0x4208,0xfc07,0,0,0,0,0,0,0,0 };

main(){
        int d,i,y,st,counts;
        double myu,y_pt,y_y,mx,my;
        double start_x,finish_x,start_y,finish_y;
        double start_xx,finish_xx,start_yy,finish_yy;
        inregs.x.ax=0; int86(51,&inregs,&outregs);
        if(outregs.x.ax==0){printf("¥nPlease install mouse.sys !¥n");exit(0);}
        m_style(0,8,style); m_color(1); m_view(0,0,639,399);
        _setvideomode(_98RESS16COLOR); _displaycursor(_GCURSOROFF);
        _main_loop_init:
        counts=300;

start_x=1.0; finish_x=4.0;
start_y=0.0; finish_y=1.0;
_main_loop_top:
_clearscreen(_GCLEARSCREEN);
setmouse(320,200); mouse(1);
_setcolor(7);
_settextposition(22,1); printf("myu : %f−%f",start_x,finish_x);
_settextposition(23,1); printf("X(n): %f−%f",start_y,finish_y);
st=0;
while(1){
        if(kbhit()!=0){ d=getch(); if(d==0x1b) goto _deguchi; }
        y_pt=0.5;
        myu=start_x+(finish_x−start_x)*(double)st/640.0;
        for(i=0;i<counts;i++){
                if(i<100){ y_pt=myu*y_pt*(1.0−y_pt); }
                else{
                        y_pt=myu*y_pt*(1.0−y_pt);
                        if((y_pt<finish_y)&&(y_pt>start_y)){
                                y_y=(y_pt−start_y)/(finish_y−start_y);
                                y=(int)(399.999*(1.0−y_y));
                                _setpixel(st,y);
```

本プログラムのソースと実行形式ファイルをディスク頒布します(MIDI音源出力機能を付加している)．頒布サービスについてはp.217をご参照ください．

```
                        }
                    }
                }
                st++; if(st==640) break;
        }
        getmouse(&old_r,&old_l,&old_x,&old_y);
        while(1){
                if(kbhit()!=0){ d=getch(); if(d==0x1b) goto _deguchi; }
                getmouse(&right,&left,&moux,&mouy);
                mx=start_x+(finish_x-start_x)*(double)moux/640.0;
                my=finish_y-(finish_y-start_y)*(double)mouy/400.0;
                _settextposition(3,3); printf("Count=%d  ",counts);
                _settextposition(5,3); printf("X=%f",mx);
                _settextposition(6,3); printf("Y=%f",my);
                if((old_l==0)&&(left!=0)){ break; }
                if((old_r==0)&&(right!=0)){
                        counts=counts+100;
                        if(counts>9999) counts=300;
                }
                old_l=left; old_r=right; old_x=moux; old_y=mouy;
        }
        old_l=left; old_r=right; old_x=moux; old_y=mouy;
        _moveto(0,mouy); _lineto(639,mouy);
        _moveto(moux,0); _lineto(moux,399);
        start_xx=mx; start_yy=my;
        while(1){
                if(kbhit()!=0){ d=getch(); if(d==0x1b) goto _deguchi; }
                getmouse(&right,&left,&moux,&mouy);
                mx=start_x+(finish_x-start_x)*(double)moux/640.0;
                my=finish_y-(finish_y-start_y)*(double)mouy/400.0;
                _settextposition(3,3); printf("Count=%d  ",counts);
                _settextposition(8,3); printf("X=%f",mx);
                _settextposition(9,3); printf("Y=%f",my);
                if((old_l==0)&&(left!=0)){ break; }
                if((old_r==0)&&(right!=0)){
                        counts=counts+100;
                        if(counts>9999) counts=300;
                }
                old_l=left; old_r=right; old_x=moux; old_y=mouy;
        }
        old_l=left; old_r=right; old_x=moux; old_y=mouy;
        _moveto(0,mouy); _lineto(639,mouy);
        _moveto(moux,0); _lineto(moux,399);
        finish_xx=mx; finish_yy=my;
        mouse(0);
        while(1){
                if(kbhit()!=0){ d=getch(); if(d==0x1b) goto _deguchi; }
                getmouse(&right,&left,&moux,&mouy);
                _settextposition(3,3); printf("Count=%d  ",counts);
                if((old_l==0)&&(left!=0)){ break; }
                if((old_r==0)&&(right!=0)){
                        counts=counts+100;
                        if(counts>9999) counts=300;
                }
                old_l=left; old_r=right; old_x=moux; old_y=mouy;
        }
        if(finish_xx<=start_xx) goto _main_loop_init;
        else if(finish_yy>start_yy){
                start_x=start_xx; start_y=start_yy;
                finish_x=finish_xx; finish_y=finish_yy;
        }
        goto _main_loop_top;
        _deguchi:
        _clearscreen(_GCLEARSCREEN); _displaycursor(_GCURSORON);
        _setvideomode (_DEFAULTMODE);
}
```

「設計コンテスト」課題規定

信号発生システム課題部門「カオス・ジェネレータ」

■目的

● カオス・ジェネレータ・システムを構築する.

■システム化にあたって

● その応用目的は明確にすることが望ましい.

● システム構成は,「組み込み用スタンドアロン・システム」か「パソコン接続のカオス・ジェネレータ・システム」を推奨する.

■カオス・ジェネレータ ASIC 開発の推奨

● システムの全体構成は自由であるが, コンテストの性格上, 中心にカオス・ジェネレータ部をチップまたは回路レベルで実現することが望ましい.

● カオス・ジェネレータ部は, 5000～10000 ゲート規模の FPGA に収まることを推奨する(推奨チップ型番は近月号で発表予定).

■第1次レポートについて（締切：5月10日）

以下の項目についてレポートすること.

● システム全体仕様の概要

● システム構成図

● カオス生成方法(微分方程式, 漸化式, アルゴリズム)

● カオス・チップ仕様（目標演算処理能力, 出力形態, 外部インターフェースなど)