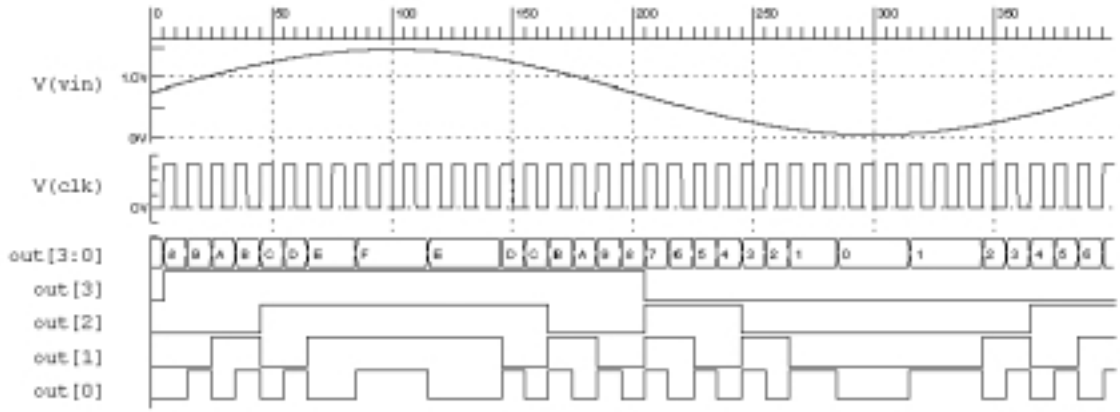
〔リスト11.6〕並列比較型4ビットA-D変換の記述例

```
module flash_ad4(adout, ain, clk);
  output[3:0] adout;
  input ain, clk;
  electrical[3:0] adout;
  electrical ain, clk;

  parameter real
    vdd = 3.3,       // 電源電圧
    vth = 1.65,      // スレッショルド電圧
    logic1 = vdd,    // 論理値1の出力電圧
    logic0 = 0.0,    // 論理値0の出力電圧
    vstep = 0.1,     // 最小ステップ電圧
    trise = 1.0n,    // 立ち上がり遷移時間
    tfall = 1.0n;    // 立ち下がり遷移時間

  real d[0:3];        // 出力ベクタ
  real state[1:16];   // 比較電圧
  integer i, j;
  real set_bit, rem;

  analog begin
    @(initial_step) begin   // 比較基準電圧の生成
       state[1] = vstep/2;
       for (i = 2; i <= 16; i = i + 1)
         state[i] = state[i-1] + vstep;
    end

    @(cross(V(clk) - vth, 1))  begin
      for (i = 1; i < 16; i = i + 1) begin
        // 入力電圧と基準電圧の比較
        if ((state[i]<=V(ain) ) && (V(ain)<state[i+1])) begin
           // ディジタル信号への変換
          rem = i;
          for (j = 3; j >= 0; j = j - 1) begin
            set_bit = rem/pow(2,j);
            if (set_bit >= 1) begin
              d[j] = vdd;
              rem = rem - pow(2,j);
            end else
              d[j] = 0.0;
           end
         end
       end

      // アンダフロー & オーバフロー
      if (V(in) < state[1])
        for (j=0; j <= 3; j = j + 1)
```

```
                d[j] = 0.0;
            else if(V(in) > state[16])
              for (j=0; j <= 3; j = j + 1)
                d[j] = vdd;
          end

          V(adout[0]) <+ transition(d[0], 0.0, trise, tfall);
          V(adout[1]) <+ transition(d[1], 0.0, trise, tfall);
          V(adout[2]) <+ transition(d[2], 0.0, trise, tfall);
          V(adout[3]) <+ transition(d[3], 0.0, trise, tfall);
        end

      endmodule
```

〔図11.7〕並列型A-Dコンバータのシミュレーション結果



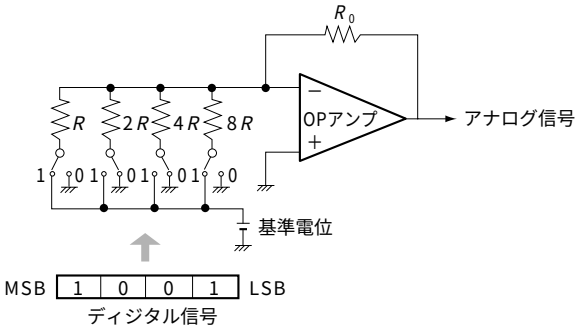
◆D-A コンバータ

D-A コンバータは，ディジタル信号をアナログ信号に変換します．ディジタル信号では，データは2進数の形で扱われています．そのため，4ビットの最上位ビット(MSB)の値は10進数では2^3を意味し，2ビット目は2^2，さらに3ビット目は2^1，最下位ビット(LSB)は2^0を意味します．このように2進数は，それぞれのビットに応じて2のベキ乗の重みが付いています．これを利用したのが**図11.8**のような重み付き回路網を利用したD-A コンバータです．ディジタル信号のそれぞれのビットに応じてスイッチがオン/オフし，それに応じた電圧(電流)を切り替えることで，最終的なアナログ信号が各ビットの電圧の総和として出力されます．

D-A コンバータの記述では，各ビットのベキ乗を計算し，その総和をアナログ信号として出力します．D-A コンバータの記述は，すでに前記A-D コンバータの記述内に含まれていますので，ここでは，ディジタル信号の入力ベクタ幅をパラメータ化したD-A コンバータの記述例を**リスト11.7**に示します．パラメータの上書きによって，この記述は入力ベクタ幅が可変な汎用的なモデルとして使用できます．

D-A コンバータには，R-2Rラダー回路を利用したD-A コンバータや，乗算型D-A コンバータなどがありますが，機能自体は変わらないので，基本的な記述を応用するだけですみます．

〔図11.8〕
重み付け回路網による
D-Aコンバータ



〔リスト11.7〕ベクタ・ポートを用いたD-Aコンバータの記述例

```
// パラメータ化したD-Aコンバータ・モデル
module dac(vin, vout);
  input[size-1:0] vin;
  output vout;
  electrical[size-1:0] vin;
  electrical vout;

  parameter real
    size = 2 from (1:inf),      // ベクタ幅
    vhigh = 5.0,                // 出力最大電圧
    vth = 2.5,                  // スレッショルド電圧
    trise = 0 from [0:inf);     // 立ち上がり遷移時間
    tfall = 0 from [0:inf);     // 立ち下がり遷移時間

  real code;
  integer i, pow2[size:0];

  analog begin
    @(initial_step)
      for (i = 0; i <= size; i = i + 1)
        pow2[i] = pow(2,i);   // 2のべき乗値の生成

    // 入力ベクタのビット比較
    code = 0;
    for (i = 0 ; i < size; i = i + 1)
      code = code + (V(in[i]) > vth) ? pow2[i] : 0;

    V(out) <+ V(vhigh)*transition(code/pow2[size],0,trise,tfall);
  end
endmodule
```

11.5 PLL(Phase Locked Loop)

PLLは，基準周波数源から所望の出力周波数を得るための回路です．**図11.9**のようにPLLを構成する主要なブロックは，位相比較器(Phase Frequency Detector)，ローパス・フィルタ，電圧制御発振器VCO(Voltage Controlled Oscillator)，分周期(Divider)です．

位相比較器は，基準信号と比較信号との位相差を検出し，位相差に応じた誤差信号を出力します．ローパスフィルタは，位相比較器からの誤差信号を平滑化，VCOの制御信号を決定し，PLLの応答を左右する重要な働きをします．VCOは制御信号によって出力周波数が変化する発振器です．VCOは通常，遅延素子をループさせた回路として構成します．また，分周期は，VCOからの発振周波数をN分の1に分周させ，基準信号と比較する比較信号を生成します．

PLLからの発振周波数f_{out}は，基準周波数をf_{ref}，分周数をNとすると以下の関係が得られます．

$$f_{out} = f_{ref} \times N$$

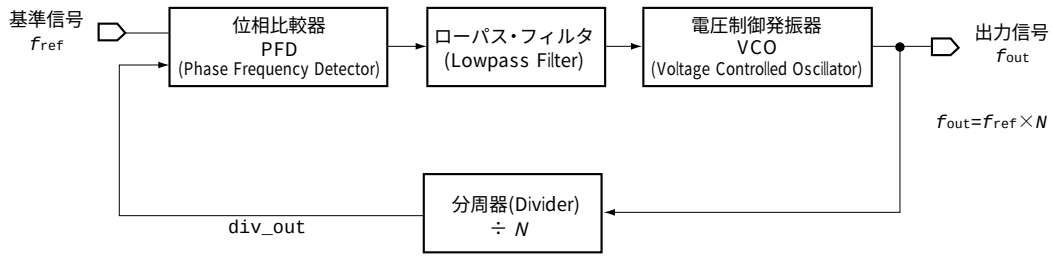
分周数が可変可能なプログラマブル分周器を用いることで，所望の出力周波数を得ることができます．

◆PLL記述

PLL全体を一つの記述モデルとして作成することは可能ですが，PLLを構成するフィルタやVCOの特性を検証することが重要な場合には，ブロックごとのモデル化が適しています．

リスト11.8にPLLのトップ記述を示します．最上位モジュール`pll`の入力信号は，基準信号`ref`，出力信号は`fout`です．`fout`は分周器の入力にも接続されていますので，ここでは`inout`と定義する必要があります．モジュール`pll`は構造記述で，位相比較器`pfd`，`vco`，`divider`，`filter`の四つのモジュールを呼び出してインスタンス化し，ネットによって相互に接続します．構造記述では，それぞれのインスタンスがVerilog-DまたはVerilog-AMSで記述されているのかはわかりません．実際にインスタンス化するモジュールは，シミュレーションする際にどのモジュール記述を読み込むかによって決まります．シミュレータは，インスタンスと同じ名前をもつモジュールを検索し，それを読み込みます．

〔図11.9〕PLLの構成図



〔リスト11.8〕PLL最上位モジュール

```
module pll(ref, fout);
  input ref;
  inout fout;
  electrical ref, fout;
  parameter integer divide_num = 4;

  pfd xpfd(pfdout, div_out, ref);
  filter xfilter(pfdout);
  vco xvco(fout, pfdout);
  divider #(.count(divide_num)) xdivide(div_out, fout);
endmodule
```

〔リスト11.9〕位相比較器PFDの記述例

```
module pfd(out, fin, refclk);
  inout out, fin, refclk;
  electrical out, fin, refclk;

  parameter real
    igain = 15u,  // PFDゲイン
    rout = 2e6,   // 出力インピーダンス
    vdd = 5.0,    // 電源電圧
    ttol = 1p,    // 0交差誤差
    td = 0,       // 出力遅延時間
    tt=2p;        // 出力立ち上がり/立ち下がり時間

  parameter integer dir = 1;  // 1(立ち上がり),-1( 立ち下がり), 0(両エッジ)
  integer state;  // ステート

  analog begin
    // リファレンス・エッジの検出 : state = -1 (upstate)
    @(cross((V(refclk) - vdd/2), dir, 1p)) begin
      state = state - 1;
    end

    // VCOエッジの検出 :  state = 1 (dnstate)
    @(cross((V(fin) - vdd/2), dir, 1p)) begin
      state = state + 1;
    end

    if (state > 1) state = 1;
    if (state < -1) state = -1;

    I(out) <+ transition(state * igain, 0, tt, tt);
    I(out) <+ V(out)/rout * abs(state);
  end
endmodule
```