VHDL- 付録

ＰＥＣＬデバイスの動作モデル

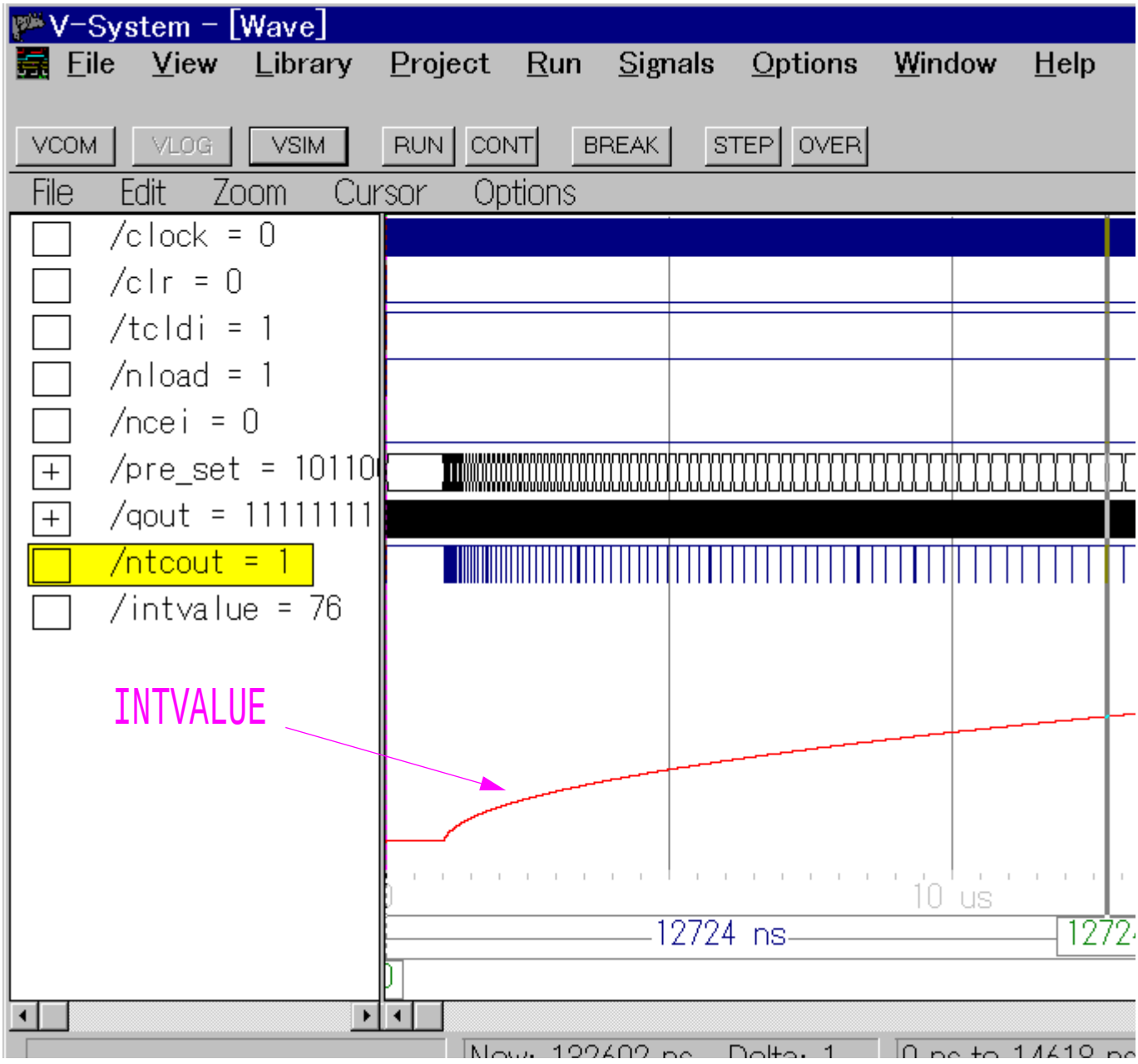
この文書の目的

既にＶＨＤＬの解説は完了しているのですが、ここでは付録として通常は余り縁のない ＰＥＣＬ デバイスについて、そのビヘイビアモデルを作成して ＭＴＩ（Ｖｓｙｓ）でのシミュレーションを行ってみます。

ＳＹ １００ Ｅ ０１６カウンタのシミュレーション結果（ＭＴＩ）

は以下のようになりますが、この結果を得る所用時間は「2 秒」でした。

ＭＴＩ でのシミュレーションでは、赤色で示すようにアナログ的な表示が可能です（この意図については後で説明します）。



付録のテーマは

普段あまり馴染みのない ＰＥＣＬ のようなデバイスを使った設計では、それらを組み合わせたシステムレベルの回路がどのように動作するか、を簡単にシミュレートできると便利なことが多いのではないでしょうか。

論理デバイスでは、アナログでの ＳＰＩＣＥ モデルなどのような共通のデータシートが提供されている訳ではありませんので、正確なタイミングモデルを作ろうなどと考える事は元々無理な事です。

ただ、論理デバイスを使う上での必要な情報は、例えば等価回路／ブロック図や真理値表などの形で与えられていますので、これを簡単に利用する事ができればシステムでの概算値での利用という目的は達成できる事になります。

しかし、余りに簡略化しすぎたモデルを作っても、そのモデルが実際に役に立つ局面は限定されてしまいますし、全ての出力信号の真理値表が常にあたえられているとは限りませんので、不足している情報はどうしてもブロック図や等価回路から補う事になります。

従って、ある程度は複雑な記述になる事を覚悟して、なるべく簡便なモデルを作成し、論理レベルでデータブックと同じ動作をする事を確かめてから、タイミングデータ `ClockTic, tpd` を付け加える事にします。

論理レベルのシミュレーションだからといって利用価値がない訳ではなく、もしこの回路に「ある特定の遅延値」を設定したら、システムとしてどのような動作が起きるかの概要を知る程度の事はできますので、今回は、ＦＰＧＡ へのフィッティング可能という事にも少し注意しながら、このような目的に使える論理シミュレーションを行います。

基になった回路は

Ｓｙｎｅｒｇｙ 社の１９９４年版データブック、Ｐ ４－５にある等価回路と、そのコントロール信号についての真理値表 Ｐ ４－６を使っています。

等価回路から明らかなように、このカウンタはマスタースレーブ形式で動作し、各カウンタ・ビットの反転条件に前段までのカウント状態と、その段の Q / $\overline{Q}$ を使って XOR 合成をきれいに処理しています。

デバイスの特徴としては、ターミナル・カウントで自動的に再ロードする機能を持っていることです。データブックから、Ｅ ０１６の等価回路は以下のように書けます。茶色で示したパッケージ「PECL_E016」を作ることで、比較的コンパクトに記述できたほうではないかと思います。

```
--=============================================================================-
Entity E016 is
     Generic(  tpd  : TIME :=1 ns  );
     PORT(     CLK,MR,TCLD,nPE,nCE : IN Bit;
               PIN  : IN BYTE;     -- original name was P
               nTC  : OUT Bit;
               Q    : OUT BYTE );
end E016;
--=============================================================================--
Architecture a of E016 is
     signal ModeIN : B3Flag;
     signal CtrMode: E016Mode;
     signal MasterFF: BYTE;
     signal SlaveFF : BYTE;
     signal nTCZ,pTCZ : Bit;
Begin
     Q         <= SlaveFF;             -- Emit output FF to pin.
     nTC       <= nTCZ;                -- Emit Terminal Count to pin.
     ModeIN    <= nCE & nPE & TCLD;    -- Get input pin, control signals.
     CtrMode   <= CtrlTruthTbl( ModeIN );
-------------------------------------------------------------------------------
GenTCZ:Process(MR,MasterFF,nTCZ,pTCZ) -- Generate nTC : a RS-FlipFlop
     variable FFlag: Bit;         -- Full-Count-Flag from "MasterFF" output.
Begin
     FFlag :='1';
     TestFULL:for I in BYTE'range loop
          FFlag := FFlag and MasterFF(I);
     end loop TestFULL;
     pTCZ<= not(nTCZ) or ( FFlag and not(MR) );
     nTCZ<= not(pTCZ) or (not(FFlag) and not(MR) ) or (MR);
end Process GenTCZ;
-------------------------------------------------------------------------------
MProc: Process(PIN,CLK,MR,CtrMode,SlaveFF,nTCZ)-- MasterFF Negative edged
Begin
     if(MR='1') then MasterFF <= (OTHERS=>'0');
     elsif(CLK'EVENT and CLK='0') then
          Case CtrMode is    -- MasterFF input multiplexer -----------------
               when Load   => MasterFF <= PIN;
               when Hold   => MasterFF <= SlaveFF;
               when Count  => MasterFF <= E016PLA( SlaveFF ) After tpd;
               when LdOnTC =>
                 if(nTCZ='0') then  MasterFF <= PIN;
                 else               MasterFF <= E016PLA( SlaveFF ) After tpd;
                 end if;
          end Case; ------------------------------------------------------
     end if;
end Process MProc;
-------------------------------------------------------------------------------
SProc: Process(MasterFF,CLK,CtrMode,MR) -- Slave FlipFlops Positive edged
Begin
     if(MR='1') then                      SlaveFF <= (OTHERS=>'0');
     elsif(CLK'EVENT and CLK='1') then    SlaveFF <= MasterFF After tpd;
     end if;
     end Process SProc;
end a; --=============================================================================--
```

特別変った事をしている訳ではありませんから、比較的簡単にＥ０１６の構造は把握できると思います。　気を付けたいのは、ＥＣＬ回路によく使われる内部コンプリメンタリ出力の読み替え部分とマスター（ MProc ）/ スレーブ（ SProc ）のＦＦがクロックエッジの負／正を使い分けているところあたりです。

見落とし易いのは、（ GenTCZ ）ブロックでターミナル・カウントの検出に使われている信号がマスター側から引き出されているところかも知れません。

この記述を行うにあたっては、２４行目で

```
CtrMode <= CtrlTruthTbl( ModeIN );
```

として、ピンからのカウンタ動作モード指定信号を受けてから４４行目で

```
Case CtrMode is  -- MasterFF input multiplexer ----------
```

と受け直している所は、元は真理値表で、これをどのように書くべきか、色々な方法があるので迷ったところなのですが、余り Std_Logic 信号を安易に使うと、こういう所が面倒になるという実例を挙げる意味もあって、Case CtrMode に渡すことにしました。

この部分で CtrMode 値の中に "X" や "U" 入ると、論理合成には使えない記述になりますし、複雑になる割に、この程度のモデル化では御利益がありません。

別途、根拠の明確なタイミングデータがあって、それを反映させる目的があるなら別ですが、なるべくこういう所は Bit_Vector で扱っておきたいと思います。

パッケージ宣言分

この記述内部でつかわれているパッケージは以下のとおりです。

```
1   --=============================================================================--
    -- A package for ECL behaviour modeling. SY100E016
    -- ------------------------------------------------------
    -- by Yoshiaki Naruse 1998 9/7 Systems Workshop Inc.
5   -- ------------------------------------------------------------------------
    library ieee;
    use ieee.Std_Logic_1164.all;
10  --=============================================================================--
    Package PECL_E016 is -- In detail,refer to SYNERGY PECL data book p4-4 ...11
    --=============================================================================--
    Type E016Mode is ( Load,Count,Hold,LdOnTC ); -- E016 operation mode.
15
    SubType BYTE is Bit_Vector(7 downto 0);
    SubType B3Flag is Bit_Vector(2 downto 0);
    function Int2BYTE(INTDATA: integer) return BYTE;
20
    function CtrlTruthTbl( ModeIN:B3Flag ) return E016Mode; -- Control In.
    function E016PLA( SlaveFF:BYTE ) return BYTE; -- Counter algorithm E016

25  end PECL_E016; --=======================================================--
    --=============================================================================--
30  Package Body PECL_E016 is
    --=============================================================================--
```

パッケージ定義部

では、宣言した関数の実体を記述しますが、見てのとおり、回路図をそのまま書き移しただけです。 E016PLA の記述では、各カウントビット毎に、下位ビットが全て「１」になっている結果を受けて反転するよう、二重のループを使っています。

```
    function Int2BYTE(INTDATA: integer ) return BYTE is
        variable retval : Bit_Vector(BYTE'LENGTH-1 downto 0);
35      variable opx : integer := INTDATA;
    Begin
        retval := (BYTE'LENGTH-1 downto 0 => '0');
        for i in 0 to BYTE'LENGTH-1 loop
            if (opx mod 2) = 1 then if (opx mod 2) = 1 then retval(i) := '1' ;
40          end if ;
            opx := opx/2 ;
        end loop ;
        return retval;
    end Int2BYTE;
45
    ---------------------------------------------------------------------------
    function E016PLA( SlaveFF:BYTE ) return BYTE is -- Counter PLA of E016
    ---------------------------------------------------------------------------
        variable RetByte: BYTE;
50      variable Temp: Bit;
    Begin
        RetByte := ( 0 => not(SlaveFF(0)) ,others => '0');
        for I in 1 to BYTE'HIGH loop     -- You know what's going on.If not,
            Temp := '1';                 -- see SYNERGY PECL data-book p4-5.
55          for J in 0 to I-1 loop       --
                Temp := Temp and SlaveFF(J);
            end loop;
            RetByte(I) := Temp xor SlaveFF(I);
        end loop;
60      return RetByte;
    end E016PLA;
    ---------------------------------------------------------------------------
    function CtrlTruthTbl( ModeIN:B3Flag ) return E016Mode is
65  ---------------------------------------------------------------------------
        variable VState : E016Mode;
    Begin
    -- For Logic synthesis, description is much simplified by not using X U values.
        Case ModeIN is
70          when "000"|"001"|"100"|"101"     => VState := Load;
            when "010"                       => VState := Count;
            when "011"                       => VState := LdOnTC;
            when "110"|"111"                 => VState := Hold;
        end Case;
75      return VState;
    end CtrlTruthTbl;       -- See p4-6 truth table

    end PECL_E016; --============================================================--
```

テストベンチ

以下のように記述しておきました。 `INTVALUE <= Pdata;` 部分は、冒頭の MTI シミュレーションで、アナログ表示を行わせている部分のテクニックです。

```
--================================================================================--
entity TestBench is  generic(ClockTic: TIME := 2 ns); end;
--================================================================================--
library ieee;
use ieee.Std_Logic_1164.all;
use work.PECL_E016.all;
--================================================================================--
Architecture a of TestBench is
     component E016
     Generic( tpd : TIME );
     PORT(      CLK,MR,TCLD,nPE,nCE : IN Bit;
                PIN : IN BYTE; -- original name was P
                nTC : OUT Bit;
                Q : OUT BYTE );
     end component;
     signal CLOCK,CLR          : Bit;
     signal TCLDI,nLOAD,nCEI   : Bit;
     signal PRE_SET,QOUT       : BYTE;
     signal nTCOUT             : Bit;
     signal INTVALUE           : integer;
Begin
CUT: E016 -- Counter Under Test ------------------------------------------
     Generic map ( tpd=> 1 ns)
     port map( CLK => CLOCK,
                MR => CLR,
                TCLD=> TCLDI,
                nPE => nLOAD,
                nCE => nCEI,
                PIN => PRE_SET,
                nTC => nTCOUT,
                Q => Qout ); -- You can add any(^^;) amount of E016 instances.--
     CLR <= '1', '0' after 10 ns;
     nLOAD <= not(CLR);
     TCLDI <= '1';
     nCEI <= '0';

CLK_GEN : Process
     Begin      CLOCK<='0'; wait for ClockTic;
                CLOCK<='1'; wait for ClockTic;
end Process CLK_GEN;

TestLoop:Process(CLOCK)
     variable Pdata: integer;
     Begin
          if(CLR='1') then Pdata := 1;
          elsif(CLOCK'EVENT and CLOCK='1') then
               if(nTCOUT ='0') then Pdata := Pdata+1;
               end if;
          end if;
          PRE_SET <= not( Int2BYTE(Pdata) );
          -- Test-Bench specific descriptions to terminate simulation -------------
          Assert ( Pdata < 256 )
          report "End of Test"
          severity FAILURE;     --- A severity is an Enum of WARNING,ERROR,FAILURE ----
          INTVALUE <= Pdata;    --- To help waveform monitoring in MTI simulator ------
end Process TestLoop;
end;
```

このソースコードはレオナルドで論理合成が可能です。（当然、実デバイスにフィットする場合には、テストベンチ部分は削除しなければなりません）

しかし、ここで設定してある ｔｐｄ などのシミュレーション用の定数が、何らかの形で実設計に反映されるような事はありませんので、誤解されないようお願いします。

コンパイル手順

このソースファイルは以下の構成をとるようにします

1. パッケージ　MikiPack.vhd
2. ＳＹ１００Ｅ０１６の設計本体　Miki.vhd

レオナルド／Ｖｓｙｓで、ここまでのコンパイルの手順は上の番号順です。

3. テスト・ベンチ　MikiTB.vhd

これは Ｖｓｙｓ でのみコンパイルする部分です。

巻末に添付してあるソースファイルは、無条件にシミュレーションが可能なように追加行を加えてあり、ここまでに引用したリスト内容と行番号に若干の差異がありますが、内容はまったく同じものです。

レオナルドでの操作

いつもの手順ですが、念のため記載しておきます

```
LEONARDO{1}: cd D:/miki
LEONARDO{2}: load_library max7
             ........
LEONARDO{3}: load_modgen max7
-- Reading module generator description from file
```

パッケージと設計本体を読込み、オプティマイズします

```
LEONARDO{4}:read -work work D:/miki/MikiPack.vhd
             ........
LEONARDO{5}: read -work work D:/miki/Miki.vhd
             ........
LEONARDO{6}: optimize .work.e016_1ns.a -target max7 -effort Standard
-chip -delay
-- Start optimization for design .work.e016_1ns.a
             est est
      Pass    LCs Delay EXPs DFFs TRIs  PIs POs    --CPU--
                                                    min:sec
             ........
      4        17     27   7    16    0   13   9      00:00
Info, Pass 1 was selected as best.
LEONARDO{7}:auto_write -format EDIF ./e016_1ns.edf
```

実フィッタの復習

この結果を受けてアルテラで７ Ｋにフィットさせてみます
maxplus2 を起動します

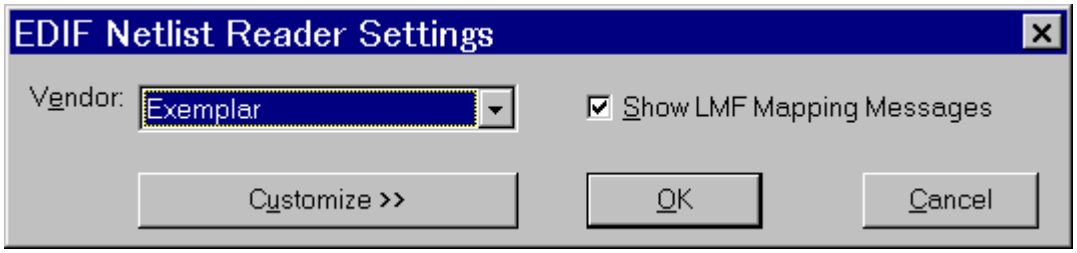
Menu/File/Open でレオナルドが生成した EDIF ファイル D:\Miki\ e016_1ns.edf をオープンします

Menu/File/Project で Set Project To Current File を選択します。

Menu/Assign で、７ Ｋの適当なデバイスを選択します。

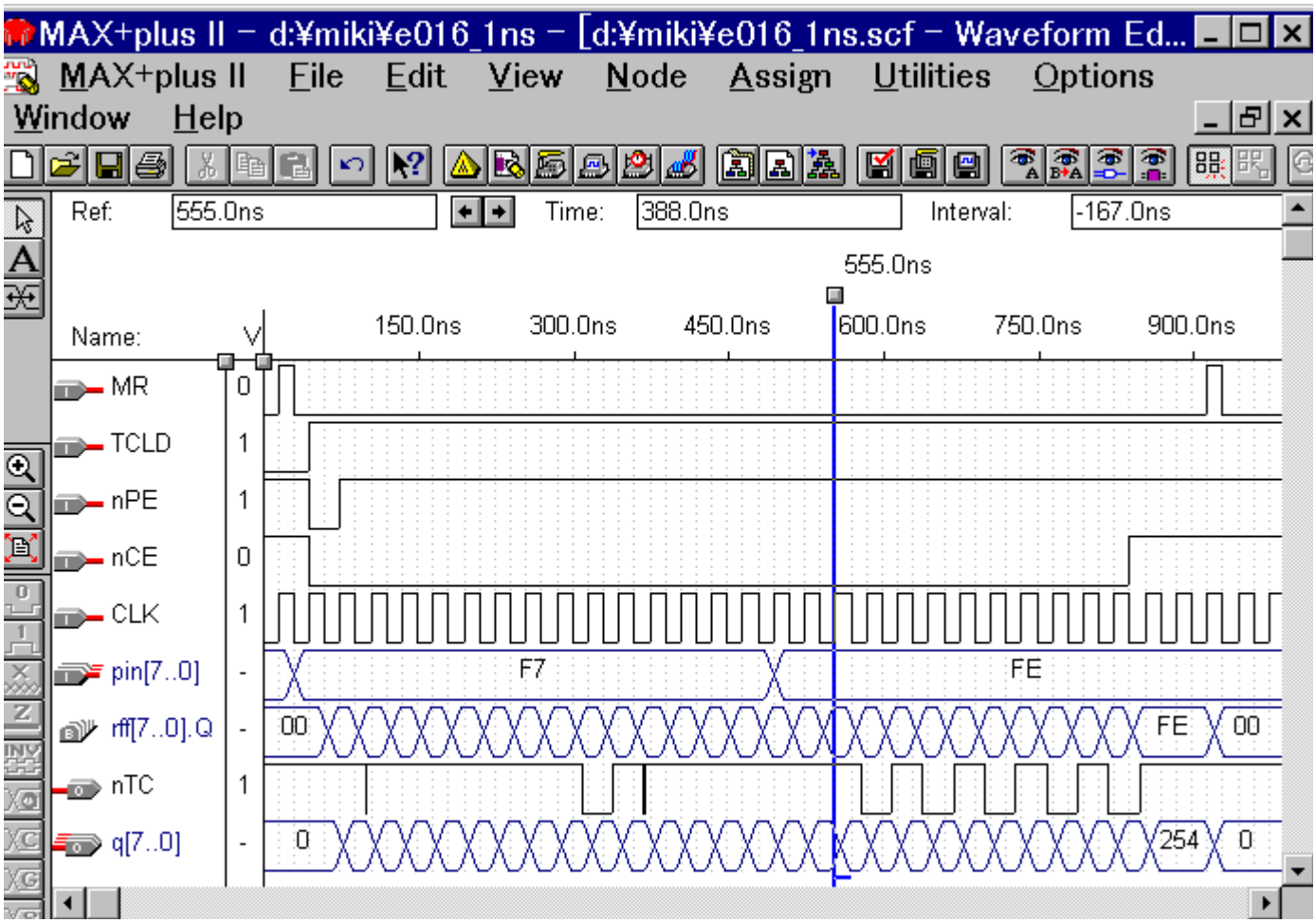
メニューバー下のアイコンでコンパイラを指定し、コンパイラメニューに切替え

Menu/InterFace/EDIF Netlist Reader Setting を選択し下図のダイアログを開き



コンパイラにスタートをかけて、フィットさせます。

アルテラフィット時のシミュレーション結果



この Miki.vhd が論理合成可能な記述になっている事は EDIF 出力が得られた事から明らかですが、基本的な回路の動作が正しいかは、このような実フィッティング結果からシミュレーションして見るのにはやや問題があります。

適切な速度でのシミュレーションを行うこと、テスト信号のセットアップ・ホールドが適正であることを必ず確認しないと、ターゲットデバイス特有の遅延から受けるタイミング制約の問題が、論理の問題と混同されることになりかねません。
当然テスト・ベンチはアルテラ用に別途回路設計を与えない限り適用不可能です。

テストベンチの復習

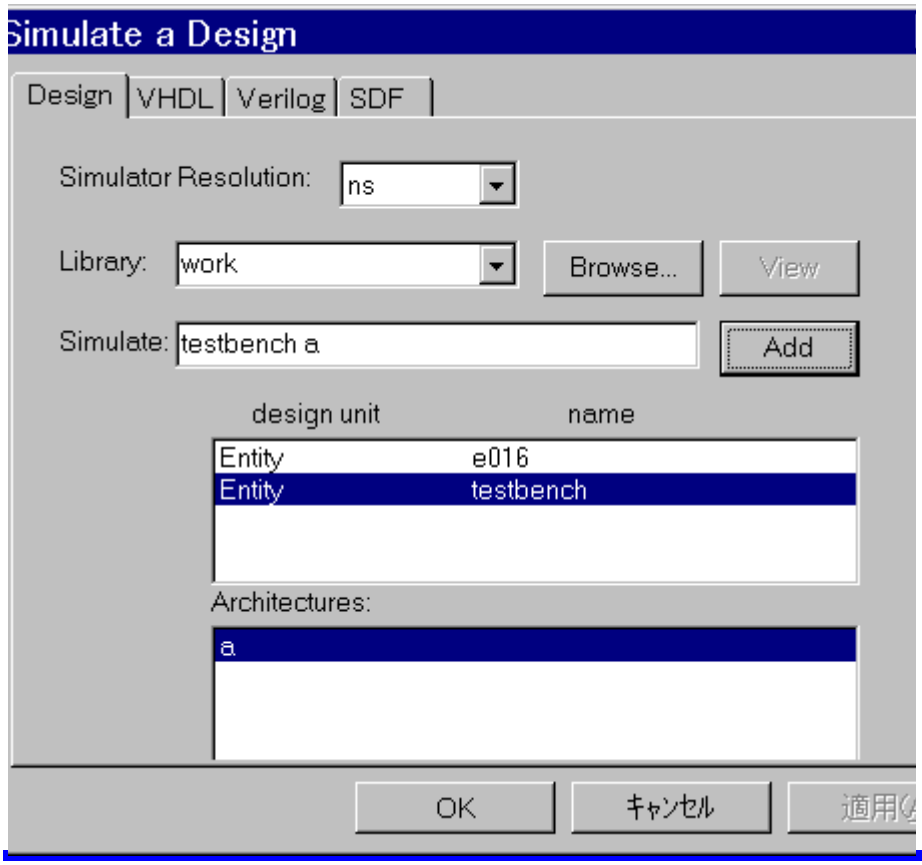
MTI / Vsystem でのテストを実行手順は以下のとおりです。
Vsys を起動し
Menue / File / Directry から、目的のディレクトリ D:\Miki を選択して、
Menue / Library / New として work をクリエートします。
Menue / Project / New として ここに VSYSTEM.INI をクリエートします。

基本的な準備はこれで出来ましたので、コンパイル作業を行います。

最初にメニュー下のアイコンから VCOM を押し

MikiPack.vhd
Miki.vhd
MikiTB.vhd の順にコンパイルを行い、DONE で終了させます。

次に Vsim を行う為、メニュー下アイコンから VSIM を選択し、



左図のダイアログを開きます。
もし、複数のモデリングを行っている場合には、シミュレートしたいコンンフィギュレーションを指定します。

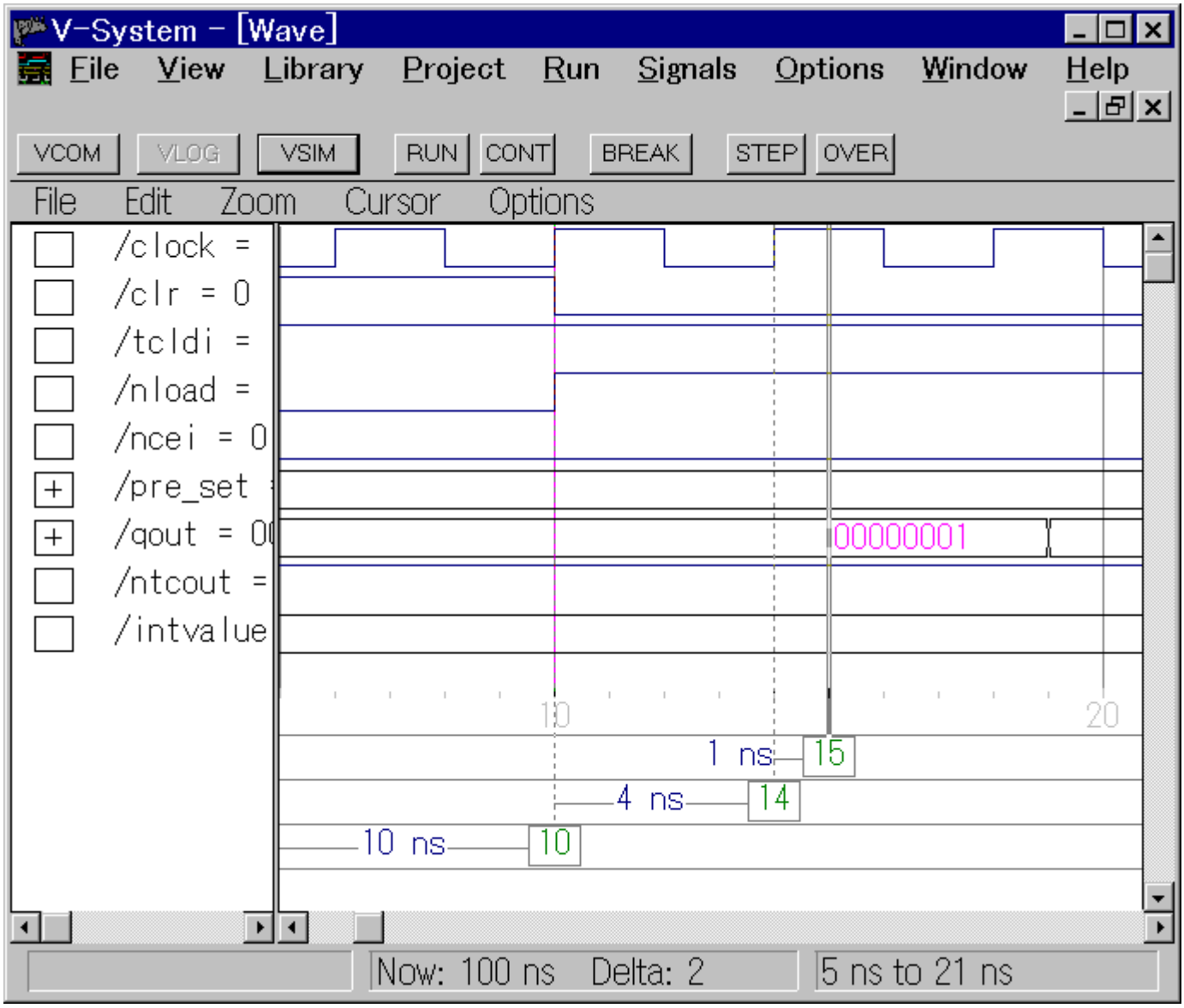
もし、設計で予め与えておいたジェネリックを操作するのであれば、この画面の ＶＨＤＬタグから、１つのジェネリック名に対して特定の値を与えることができます。
OK を押してシミュレーション内容をコンパイルします。

この段階で、すでにトランスクリプト画面はＶＳＩＭになっていますから、

view wave

wave *

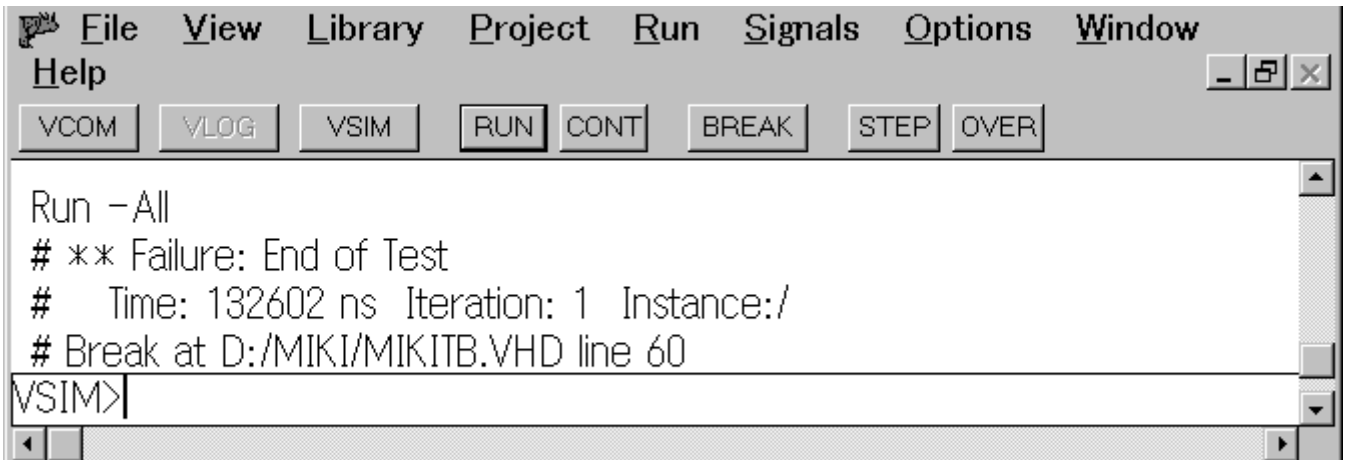
として波形画面を開き run してみれば下図のシミュレート結果が得られます。



ここでは、カーソルを与えて、予め設定しておいた tpd、ClockTic が忠実にシミュレート結果に反映されていることが確認できます。

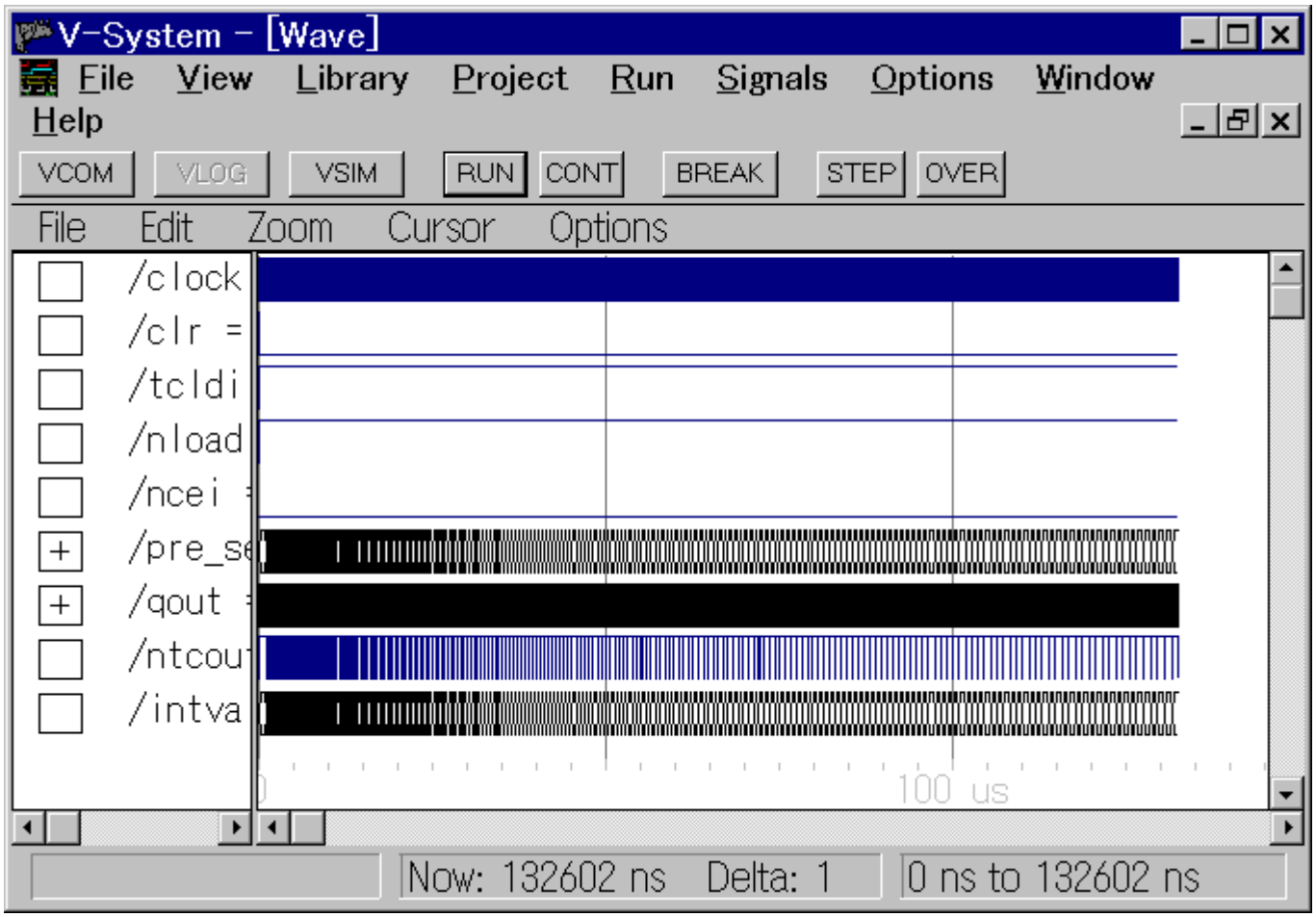
全体像を掴むために、Run -all を行ってみます。

ここで期待しているのは、仕掛けておいた打ち切り条件が正しく処理される事ですから、メニューの WINDOW で画面をトランスクリプトに戻しておきます。



上のように、テストベンチからの「End of Test 」メッセージがシビリティ・レベル Failure で検出され、シミュレーションの終了が検出されています。

波形をみてみましょう。



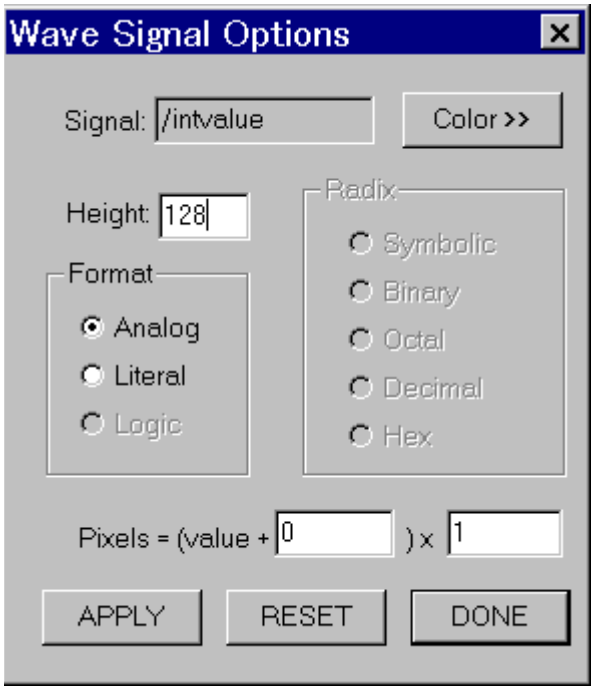
打ち切りの発生／終了時間が明確なのは良いのですが、これでは何が全体として発生しているか、すぐには把握しにくいような気がします。

コードを読んでいれば、このテスト・ベンチ MikiTB.vhd には終了処理の他に下の処理が入っていることに気がついたはずです。

INTVALUE <= Pdata; --- To help waveform monitoring in MTI simulator ------

これはビット・ベクターなどのバイナリイメージの入出力信号を意図的にインテジャーのシグナルに作っているもので、その効用は冒頭のページで表示したように、アナログ的な画面表示が行えることにあります。

波形図最下列にあるこの信号を選択し、右マウスから Options / SignalOptions を指定して下図のダイアログを開きます。



アナログ表示を選択し、表示部分の大きさをハイト指定で与えますが、必要に応じてカラー設定を行って目立たせることもできます。

この処理の結果が、冒頭ページの波形表示です。

有効な活用の例として、たとえば、「単調増加しているはずの信号」が与えた条件下で期待どおりに動作しているか、等と言うマクロな動作確認だと思います。

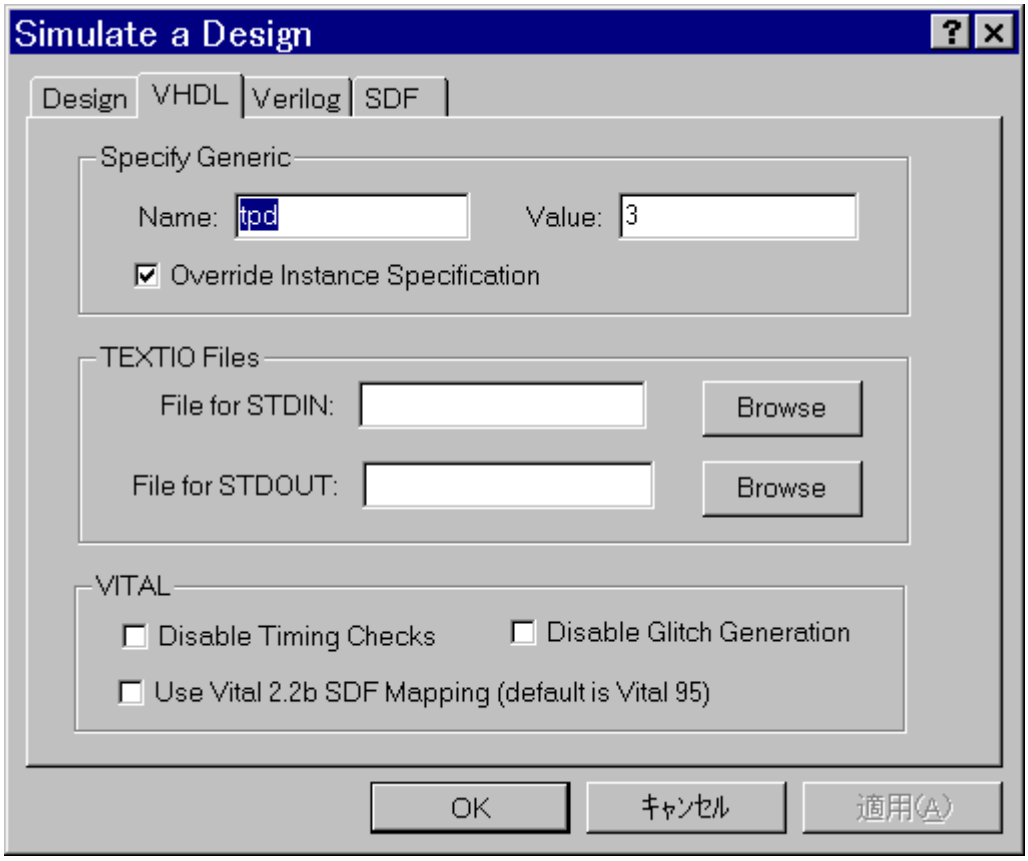
マクロに見ることで、焦点を絞ってからミクロに波形観測をする訳ですが、基本的にＶＨＤＬではＴｅｘｔＩＯの活用を心がけるようにするほうが良い結果をもたらすような気がします。

一旦 Vsys のシミュレータに設計を載せると、これまでの他の解説編で説明して来たような操作が行えます。

次ページに示すのは、デフォルト値を与えたジェネリックスに対し、Vsys のダイアログから新しいパラメータを与えてその結果を確認する手順です。

適当なＦＰＧＡデバイスに実フィットし、sdf ファイルや、アルテラの vho ファイルからバック・アノテートする手順も既に他の解説編で説明してありますので、このような動作記述モデルと複合させたり、基板の伝送遅延時間をテストベンチ上から与えて、システムの基本的な検証を行うことが（時間と体力が許せば）可能となっている事に留意して、この付録編を終わります。

この手順では、パラメタ tpd（デフォルト値 ２ ｎS）を書き換えてシミュレートさせています。



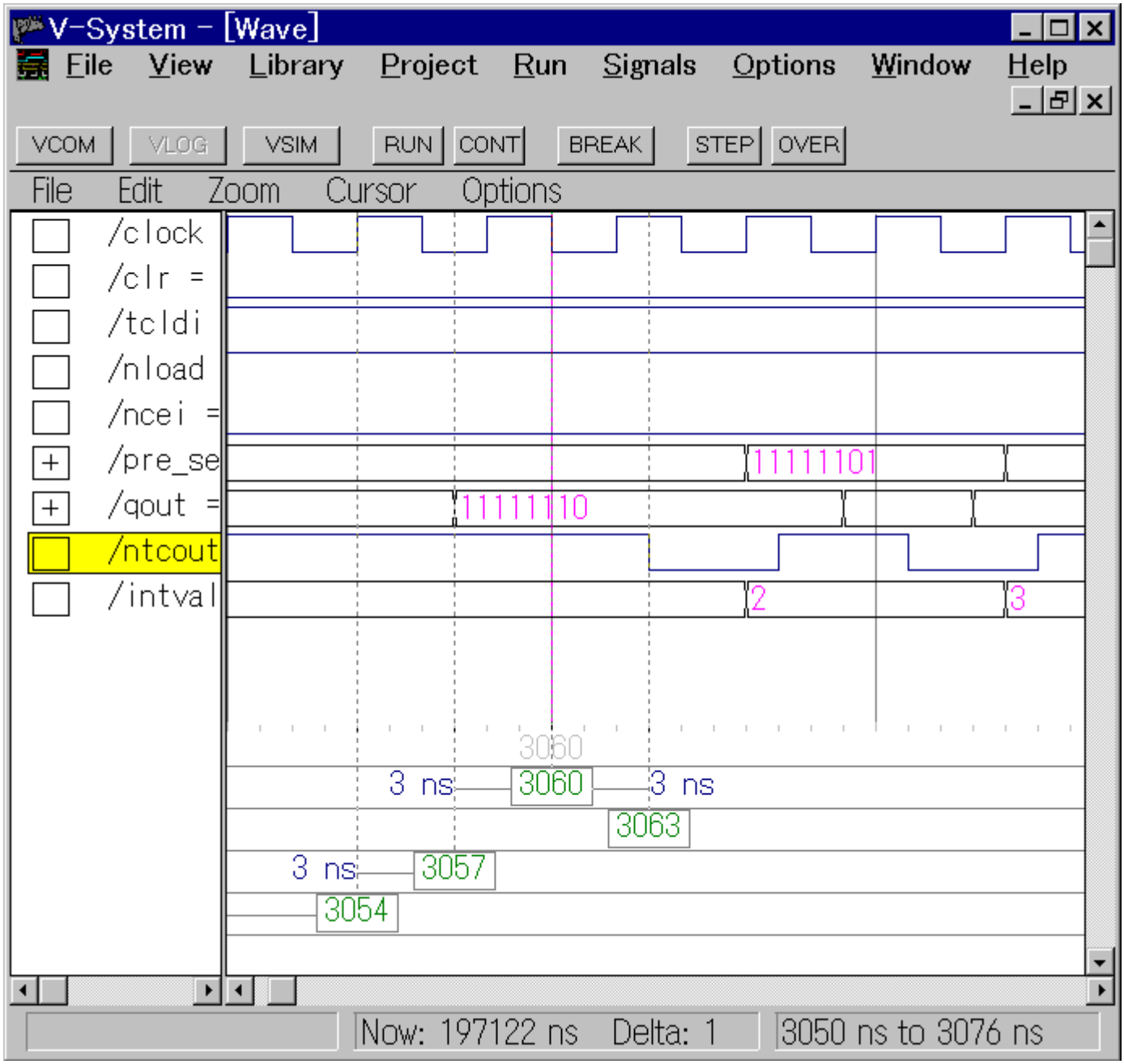
画面表示のように、TextIO ファイルの指定もここで処理します。

一般的に、バイタル / sdf ファイルには、適用できないタイミング・パラメータも混入しているものですが、この画面の VITAL 部でタイミング・チェックを禁止すれば基本的な対応が可能であることを、「ＶＨＤＬ １」編でティップスとして紹介してありますので、十分に注意して参照するようにして下さい。

次ページに示すとおり、このような簡単なモデリングでは、デバイス内のマスター／スレーブ ＦＦ が ＣＬＯＣＫ 信号に対して３ ｎＳ の伝播遅延をもつものと仮定する処理手順も簡単になります。

スタートアップ時に、ロード信号とリセット信号が全く同じタイミングで印加されている場合、このデバイスは、最初の ｎＴＣ イベントが発生するまで、フル・カウント動作を一回だけ行いますが、これはアナログ表示部分の平坦部分に注意すれば発見することが容易です。

下図のとおり、カーソルの示すタイミング値が、ダイアログから指定した３ ｎＳの遅延に変更操作できたことが確認できます。



基板のパターン引き回しによる伝播遅延の影響も、テスト・ベンチの内部で接続用シグナルを作成し、適当に遅延値を設定してシミュレート可能ですので、システムの構築時にも利用が可能です。（結構きつい作業になるかも（^^;）知れません）

MikiPack.vhd doc

```
1       --=============================================================================--
        --    A "MikiPack" package for ECL behaviour modeling. SY100E016
        --  ---------------------------------------------------------
        --        by Yoshiaki Naruse    1998 9/7  Systems Workshop Inc.
5       --  -----------------------------------------------------------------------

        library ieee;
        use ieee.Std_Logic_1164.all;

10      --=============================================================================--
        package PECL_E016 is  -- In detail,refer to SYNERGY PECL data book p4-4 ...11
        --=============================================================================--


            Type E016Mode   is ( Load,Count,Hold,LdOnTC );  -- E016 operation mode.
15
            SubType BYTE    is  Bit_Vector(7 downto 0);
            SubType B3Flag  is  Bit_Vector(2 downto 0);

            function Int2BYTE(INTDATA: integer) return BYTE;
20
            function CtrlTruthTbl( ModeIN:B3Flag ) return E016Mode; -- Control In.

            function E016PLA( SlaveFF:BYTE ) return BYTE;   -- Counter algorithm E016

25      end  PECL_E016; --=========================================================--



        --=============================================================================--
30      package body PECL_E016 is
        --=============================================================================--

        function Int2BYTE(INTDATA: integer ) return BYTE is
            variable retval : Bit_Vector(BYTE'LENGTH-1 downto 0);
35          variable opx    : integer := INTDATA;
        Begin
            retval := (BYTE'LENGTH-1 downto 0 => '0');
            for i in 0 to BYTE'LENGTH-1 loop
                if (opx mod 2) = 1 then retval(i) := '1' ;
40              end if ;
                opx := opx/2 ;
            end loop ;
            return retval;
        end Int2BYTE;
45
        -------------------------------------------------------------------------------

        function E016PLA( SlaveFF:BYTE ) return BYTE is     -- Counter PLA of E016
        -------------------------------------------------------------------------------


            variable RetByte: BYTE;
50          variable Temp: Bit;
        Begin
            RetByte := ( 0 => not(SlaveFF(0)) ,others => '0');
            for I in 1 to BYTE'HIGH loop            -- You know what's going on.If not,
                Temp := '1';                        -- see SYNERGY PECL data-book p4-5.
```

```
55              for J in 0 to I-1 loop              --
                    Temp := Temp and SlaveFF(J);
                end loop;
                RetByte(I) := Temp xor SlaveFF(I);
            end loop;
60          return RetByte;
        end E016PLA;

        ------------------------------------------------------------------------------

        function CtrlTruthTbl( ModeIN:B3Flag ) return E016Mode is
65      ------------------------------------------------------------------------------

            variable VState  : E016Mode;
        Begin
        -- For Logic synthesis, description is much simplified by not using X U values.
            Case ModeIN is
70              when "000"|"001"|"100"|"101" => VState := Load;

                when "010"                   => VState := Count;
                when "011"                   => VState := LdOnTC;
                when "110"|"111"             => VState := Hold;
            end Case;
75          return VState;
        end  CtrlTruthTbl;  -- See p4-6 truth table


        end PECL_E016; --============================================================--
80
```

Miki.vhd doc

```
1       --=============================================================================--
        --
        --  A simple but meaningfull DEMO for ECL behaviour modeling.
        --  ---------------------------------------------------------
5       --        by Yoshiaki Naruse    1998 9/7  Systems Workshop Inc.
        --
        --  -----------------------------------------------------------------------
        --   This design is Based on SYNERGY PECL data-book p4-5/4-6 SY100E016 model.
        --  and is aimed to compile into EDIF output via leonarodo 4.22.
10      --  -----------------------------------------------------------------------
        --
        --   This model does not have any simulation oriented directives and you will
        --  not encount any" Std_Logic" signals in it.  This is acctually a synthesis
        --  style modeling description.
15      --
        --   As a result, you can not expect any actual "100E016" timing data from it.
        --
        --   However, You can add some "After TIME" clauses to provide MTI simulations
        --  and you can FIT this into FPGAs to check logical behaviour of E016.
20      --  (I wish it works correctry)
        --
        --   Sometimes, in such schematic origined modeling cases, a VHDL design tends
        --  to be described in a NetList style, a hard to read complex one. (--;)
        --
25      --   In this case, I’ve tried to make it easy-reading and ’ve tried to involve
        --  some basic VHDL techniques in it also.  I hope you really enjoy it (^^)/
        --
        --
        --=============================================================================--
30      library ieee;
        use ieee.Std_Logic_1164.all;
        --=============================================================================--
        use work.PECL_E016.all;
        --=============================================================================--
35      Entity  E016 is
            Generic(tpd :TIME := 1 ns );
            PORT(   CLK     : IN Bit;
                    MR      : IN Bit;
                    TCLD    : IN Bit;
40                  nPE     : IN Bit;
                    nCE     : IN Bit;
                    PIN     : IN BYTE;      -- original name was P
                    nTC     : OUT Bit;
                    Q       : OUT BYTE  );
45      end E016;
        --=============================================================================--



        --=============================================================================--
50      Architecture a of E016 is
            signal  ModeIN      : B3Flag;
            signal  CtrMode     : E016Mode;
            signal  MasterFF    : BYTE;
            signal  SlaveFF     : BYTE;
55          signal  nTCZ,pTCZ   : Bit;

        Begin
```

```
        Q           <= SlaveFF;         -- Emit output FF to pin.
        nTC         <= nTCZ;            -- Emit Terminal Count to pin.
60
        ModeIN  <= nCE & nPE & TCLD;    -- Get input pin, control signals.
        CtrMode <= CtrlTruthTbl( ModeIN );

        ------------------------------------------------------------------------
65      GenTCZ:Process(MR,MasterFF,nTCZ,pTCZ)   -- Generate nTC : a RS-FlipFlop
        ------------------------------------------------------------------------
            variable FFlag: Bit;    -- Full-Count-Flag from "MasterFF" output.
        Begin
            FFlag := '1';
70          TestFULL:for I in BYTE'range loop
                 FFlag := FFlag and MasterFF(I);
            end loop TestFULL;
            -- Caution : This AND-GATE is an AS-IS description of PECL schematic.
            --   When inpremenring to FPGAs, it might cause a problem because of
75          --  prop-delays on each bits of MasterFF(i) are not equal eachother.
            --   As a R/S-type flip-flop state is decided by its last S/R inputs
            --  combinations, these outputs of the FF will have glitched instants
            --  whenever a MasterFF changes its value.
            --   For this application, it is safe to use this AND-GATE to the RS-
80          --  Flip-Flop input, but is not allways safe, you have to notice it.
            --

            pTCZ<= not(nTCZ) or (   FFlag   and not(MR) );
            nTCZ<= not(pTCZ) or (not(FFlag) and not(MR) ) or (MR); -- A "MUST" gate
85
        end Process GenTCZ;

        ------------------------------------------------------------------------
        MProc: Process(PIN,CLK,MR,CtrMode,SlaveFF,nTCZ) -- Master FlipFlops
90      ------------------------------------------------------------------------
        -- MasterFF is Negative edged .
        ------------------------------------------------------------------------

        Begin
            if(MR='1') then         MasterFF <= (OTHERS=>'0');
95          elsif(CLK'EVENT and CLK='0') then
                Case CtrMode is     -- MasterFF input multiplexer -----------------
                    when Load   =>  MasterFF <= PIN After tpd;
                    when Hold   =>  MasterFF <= SlaveFF After tpd;
                    when Count  =>  MasterFF <= E016PLA( SlaveFF ) After tpd;
100                 when LdOnTC =>
                        if(nTCZ='0') then MasterFF <= PIN After tpd;
                        else        MasterFF <= E016PLA( SlaveFF ) After tpd;
                        end if;
                end Case;   ---------------------------------------------------
105
            end if;
        end Process MProc;

        ------------------------------------------------------------------------

110     SProc: Process(MasterFF,CLK,CtrMode,MR) -- Slave FlipFlops
        ------------------------------------------------------------------------
        -- SlaveFF is Positive edged .
```

```
          ------------------------------------------------------------------------------

          Begin
115           if(MR='1') then                    SlaveFF <= (OTHERS=>'0');
              elsif(CLK'EVENT and CLK='1') then  SlaveFF <= MasterFF After tpd;
              end if;
          end Process SProc;

120   end a;  --=============================================================--
```

Test Bench

```
--================================================================================--
entity TestBench is  Generic(ClockTic : TIME:=2 ns );   end;
--================================================================================--

library ieee;
use ieee.Std_Logic_1164.all;
use work.PECL_E016.all;

--================================================================================--
Architecture a of TestBench is
component E016
    Generic( tpd : TIME  );
    PORT(   CLK,MR,TCLD,nPE,nCE     : IN Bit;
            PIN                     : IN BYTE;      -- original name was P
            nTC                     : OUT Bit;
            Q                       : OUT BYTE  );
end component;
    signal CLOCK,CLR        : Bit;
    signal TCLDI,nLOAD,nCEI : Bit;
    signal PRE_SET,QOUT     : BYTE;
    signal nTCOUT           : Bit;
    signal INTVALUE         : integer;

Begin

CUT: E016   -- Counter Under Test ------------------------------------------
 Generic map ( tpd  => 1 ns)
 port map(  CLK => CLOCK,
            MR  => CLR,
            TCLD=> TCLDI,
            nPE => nLOAD,
            nCE => nCEI,
            PIN => PRE_SET,
            nTC => nTCOUT,
            Q   => Qout );  -- You can add any(^^;) amount of E016 instances.--

CLR     <= '1', '0' after 10 ns;
nLOAD   <= not(CLR);
TCLDI   <= '1';
nCEI    <= '0';

CLK_GEN : Process
    Begin   CLOCK<='0'; wait for ClockTic;
            CLOCK<='1'; wait for ClockTic;
    end Process CLK_GEN;

TestLoop:Process(CLOCK)
    variable Pdata: integer;
    Begin
        if(CLR='1') then    Pdata := 1;
        elsif(CLOCK'EVENT and CLOCK='1') then
            if(nTCOUT ='0') then Pdata := Pdata+1;
            end if;
        end if;
        PRE_SET <= not( Int2BYTE(Pdata) );
```

Test Bench

```
       -- Test-Bench specific descriptions to terminate simulation -------------------

           Assert ( Pdata < 256 )
           report "End of Test"
           severity FAILURE;   --- A severity is an Enum of WARNING,ERROR,FAILURE ----

           INTVALUE <= Pdata;  --- To help waveform monitoring in MTI simulator ------

       end Process TestLoop;

       end;
```