VHDL-１２（前）

最終回（前半） 簡単に確認できる事

この文書の目的

言語を使った設計について「色々な問題があり実用にならない」というような話を聞くことが希にありますが、この最終回でその実態を少し見てみます。

はじめに、合成回路の実現速度について、符号付加算回路の場合は RTL 記述でどの程度の加算回路が自動生成されるか試し、デバイスの性能をもっとも効果的に使用すると言われている回路図形式での専用マクロを使った例との比較をします。

例題では最初に、符号付 8 ビット整数レジスタを加算し、符号付 8 ビット整数レジスタで出力する回路を、次に同じ回路を２４ビットへ拡張して結果をみます。

フレックスやＬＣＡのようデバイスと、プロダクト・ターム形式のデバイスであるＣＰＬＤの場合、について、同じコンパイラ／フィッタでの比較を行う為に、アルテラの EPM7128SQC100-15(CPLD) と、EPF6016-3(FLEX) の結果を見ます。

但し、全ての場合についてピンの配置はフィッタに任せます。

8 ビット加算回路：ＣＰＬＤでの比較

① ＶＨＤＬ記述・レオナルド速度優先／ＥＤＩＦ出力を使った場合

フィッタをＦＡＳＴとし、全てデフォルト条件でフィットした時
動作速度は <CLOCK PERIOD=**39nS**> となりました。

Chip/POF	Device	Input Pins	Output Pins	Bidir Pins	Shareable LCs	Shareable Expanders
adder_8	EPM7128SQC100-15	17	8	0	**27**	**10**

フィッタをＦＡＳＴとし、ソフトバッファ無視、
ADVANCED オプションをすべてチェックしてフィットすると、
動作速度は <CLOCK PERIOD=**31nS**> となります。

Chip/POF	Device	Input Pins	Output Pins	Bidir Pins	Shareable LCs	Shareable Expanders
adder_8	EPM7128SQC100-15	17	8	0	**34**	**9**

② 回路図記述を使い、ＬＰＭマクロでのＦＡＳＴフィッティングした結果は
動作速度は <CLOCK PERIOD=**30nS**> となり、最高速です。

Chip/POF	Device	Input Pins	Output Pins	Bidir Pins	Shareable LCs	Shareable Expanders
lpmadd8	EPM7128SQC100-15	17	8	0	**26**	**14**

面積／速度のどれを取っても ＣＰＬＤ デバイスで、この程度の規模の場合には、回路図入力から得られた結果のほうが「やや」優れている、と言えそうです。

VHDL ソース

VHDL ソースは以下のとおりです（このソースは簡単ですから添付しません）

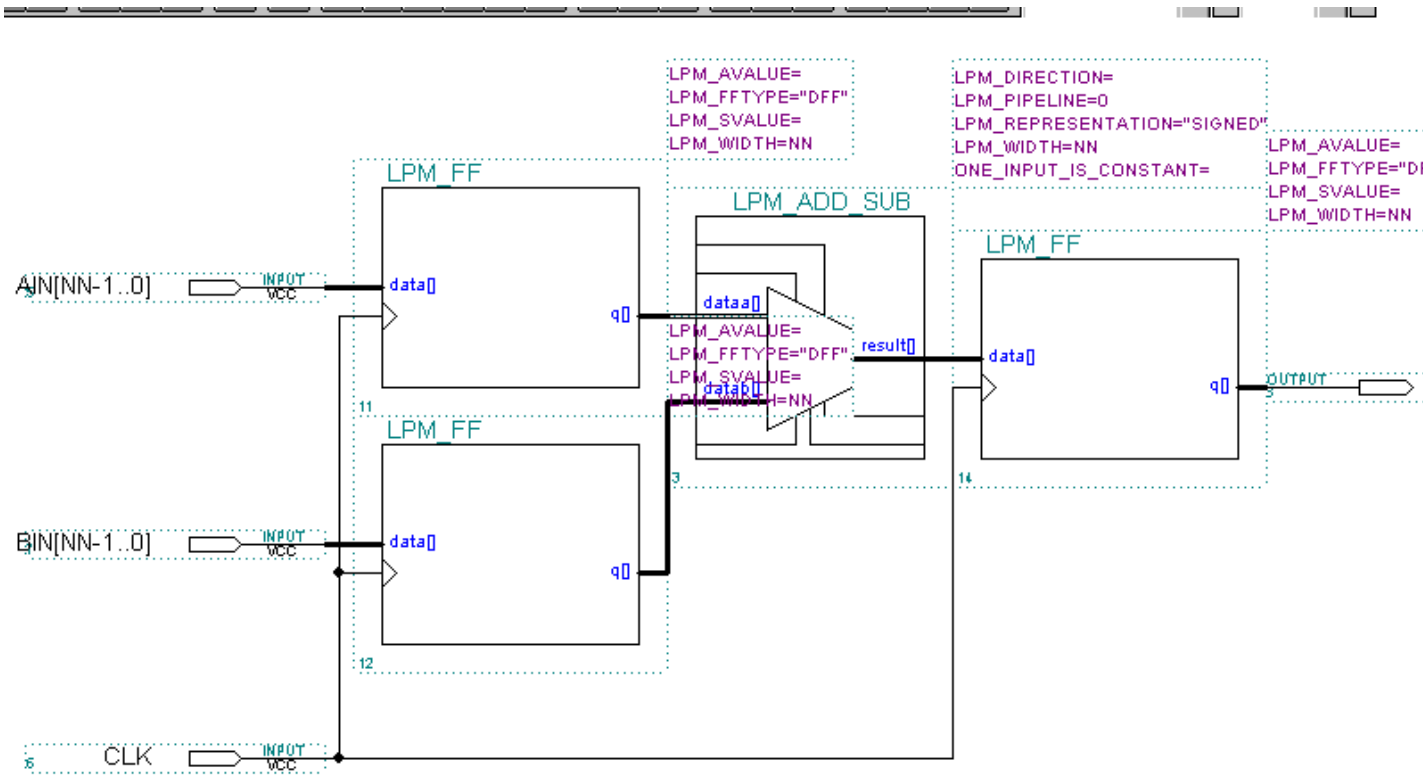
シミュレーションを行う場合（言う迄もないとは思いますが）入力信号のレンジは、符号付きの出力ビット数を考慮しておくことを忘れないでください。

```
------------------------------------------------------------------
--miki example adder.vhd
------------------------------------------------------------------
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_signed.all ;

entity adder is
  generic( nn : integer:=  8);
  port ( AIN,BIN: in  std_logic_vector( nn-1 downto 0);
         CLOCK:   in  std_logic;
         QOUT:    out std_logic_vector( nn-1 downto 0)-- nn
         );
end adder;

------------------------------------------------------------------
architecture a of adder is
------------------------------------------------------------------
Begin
  PROCESS(CLOCK,AIN,BIN)
  variable AREG,BREG,SUM  : integer range -2**(nn-1) to 2**(nn-1);
  Begin
    SUM  := (AREG+BREG);
    if(CLOCK'EVENT and CLOCK='1') then
        QOUT <= Conv_Std_logic_vector(SUM,nn);    -- nn+1
        AREG := Conv_integer(AIN);
        BREG := Conv_integer(BIN);
    end if;
  end process;
end a;
```

回路図入力内容

アルテラの maxplus2 で .gdf（回路図）として与えた図面内容は次ページに示すとおりです。　　図中で使用している加算器などの ＬＰＭ マクロは、ザイリンクスでの ＸＢＬＯＸ と同じようなものです。

同図では、ピン／ビット並びの数 ＮＮ をグローバル・パラメタとして与えてありますが、これは ＶＨＤＬ でのジェネリックとよく似たパラメータの与えかたですので、詳細はアルテラの maxplus2 ヘルプを参照してください。



8 ビット加算器：ＦＰＧＡ での比較

ＶＨＤＬ 設計では

同じ設計をレオナルドから Epf 6016 に与え、WYSIWYG でコンパイルすると

動作速度は <CLOCK PERIOD= １２．２ nS > となります

```
LEONARDO{99}: optimize .work.adder_8.a -target flex6 -effort Standard -chip -delay
-- Start optimization for design .work.adder_8.a
Ignoring wire table
                est est
      Pass      LCs Delay DFFs TRIs  PIs POs    --CPU--
                                                  min:sec
      1          24    14   24    0   17   8      00:00
      2          24    14   24    0   17   8      00:00

Info, Pass 1 was selected as best.
```

アルテラフィッタのレポートファイル内容は以下のとおりです。

```
** DEVICE SUMMARY **
Chip/                       Input    Output    Bidir
POF        Device           Pins     Pins      Pins       LCs
adder_8    EPF6016QC240-3    17        8         0        24
```

ＬＰＭ回路図面入力では

動作速度は <CLOCK PERIOD= １１．８ｎＳ > で、ほぼ同等の値 となりました

```
** DEVICE SUMMARY **

Chip/                          Input    Output    Bidir
POF        Device              Pins     Pins      Pins       LCs
lpmadd8    EPF6016QC240-3       17        8         0        24
```

上に得られた２つの結果での速度の相違は、フィッタ／ルータでの配線ルートの走り方による違いに過ぎない程度のようです。

設計規模が大きくなると：２４ビット幅に適用する

ＦＰＧＡでは、差が無くなる事がある

ＦＰＧＡの場合、アルテラの VSN8.3 フィッターでの結果は、回路規模が大きくなると、ＬＰＭでの結果と VHDL での結果との差は小さくなっています。

（変更するにはＶＨＤＬではジェネリック値 nn を２４に書き換えます。）

ＬＰＭ回路図入力（XBLOX) の場合は以下の結果を得ました。

```
maxplus2 - VSN8.3 での結果
Chip/                          Input    Output    Bidir
POF        Device              Pins     Pins      Pins       LCs
lpmadd24   EPF6016QC240-3       49        24        0        72
動作速度は＜２２．１ｎＳ＞

VSN9.0 では
Chip/                          Input    Output    Bidir      LCs
POF        Device              Pins     Pins      Pins       LCs
lpmadd24   EPF6016QC240-3       49        24        0        72
動作速度は＜１４．２ｎＳ＞
```

ＶＨＤＬ入力の場合（WYSIWYG コンパイル）は以下のようになり、

```
maxplus2 - VSN8.3 での結果
Chip/                          Input    Output    Bidir
POF        Device              Pins     Pins      Pins       LCs
adder_24   EPF6016QC240-3       49        24        0        72
動作速度は＜２２．１ｎＳ＞

VSN9.0 では
Chip/                          Input    Output    Bidir
POF        Device              Pins     Pins      Pins       LCs
adder_24   EPF6016QC240-3       49        24        0        72
動作速度は＜１９．０ｎＳ＞
```

VSN ８．３では、まったく同じ速度／面積が得られていることが判りますが、

一方では、アルテラの最新版のフィッタ VSN9.0 は、レオナルドの合成結果に対して再び差をつけているように見えます。

（注意：ＬＰＭマクロを使った場合でも、フィッタ設定がノーマルの時は、動作速度は３８ｎＳ、ＬＣ数も１４４で、EDIF 結果より劣った結果が出ました。）

ＨＤＬ／回路図と言う「入力方法の違いに依る差」は、一面において、ＶＨＤＬコンパイラが内蔵するモジュール･ジェネレータと、ターゲット･デバイスのフィッタ（バージョン）とのマッチングの問題、つまり、ベンダーの提供する「配置配線テクニックの優劣の差」の事であるように見えます。

もしも、このような一面を指して「入力方法の差」と一緒に議論している場合があるとすれば、それはどのベンダーのツールを使うべきかと言う次元の議論に落ち着くような内容であるのかもしれません。

つまらない議論ですが、もう少しだけ見てみましょう。

ＣＰＬＤでの結果を見ると

先ほどと同じようにＣＰＬＤについての比較をしますと、ＬＰＭ入力とＨＤＬからの入力との速度差はますます大きくなっています

ＬＰＭ回路図入力（XBLOX 相当）の結果は以下のようになります。

```
maxplus2 - VSN8.3 での結果
Chip/                    Input  Output  Bidir         Shareable
POF       Device          Pins    Pins     Pins    LCs  Expanders
lpmadd24 EPM7128SQC100-15 49       24        0    125      40
動作速度は＜６１．０ｎＳ＞

VSN9.0 では
Chip/                   Input   Output  Bidir         Shareable
POF      Device          Pins    Pins     Pins     LCs  Expanders
lpmadd24 EPM7128SQC100-15 49       24        0     125      40
動作速度は＜３２．０ｎＳ＞
```

レオナルドからＥＤＩＦで入力した結果は以下のようになります。

```
maxplus2 - VSN8.3 での結果
Chip/                    Input  Output  Bidir         Shareable
POF       Device          Pins    Pins   Pins   LCs  Expanders
adder_24  EPM7128SQC100-15 49       24      0     93      34
動作速度は＜７５．０ｎＳ＞

VSN9.0 では
Chip/                    Input  Output   Bidir         Shareable
POF       Device          Pins   Pins     Pins    LCs  Expanders
adder_24  EPM7128SQC100-15 49       24       0     101     54
動作速度は＜５６．０ｎＳ＞
```

この局面では、ＣＰＬＤ に関するレオナルドのモジュール・ジェネレータの性能がアルテラの ＬＰＭ に対して速度面で「かなり」劣った結果を出していることになります。

これまでの結果から見ると、アルテラ社の VSN9.0 のフィッタでは、ＬＰＭマクロに対する更なるチューニングが行われた形跡があり、同社自身の製品 VSN8.3 でのフィッティング結果に対しても大きな差をつけています。

「言語による設計がより劣った結果を生む」と言う議論では、このようなフィッターとモジュール・ジェネレータの性能のマッチング局面について、を指して言っている場合があるかもしれません。

今回の比較が、

レオナルド：　VSN ４．２２（１９９８年3月１７日版）と

アルテラ：　VSN ８．３ （１９９８年4月２日版）

の比較でほぼ同等な結果が出たのに対し

アルテラ：　VSN ９．０ （１９９８年7月３０日版）

での結果を照合して、両者の性能差が無視できるほどに小さくなる場合と、歴然とした違いがてくる場合の経緯を考えてみると、この「差」というものは技術競争の瞬間風速的な途中経過を意味しているのではないかと思われます。

ある特定のデバイスについて、最適の配置配線テクニックを真っ先に提供できるのは通常はデバイス・ベンダーである筈ですから、そのような特定のターゲットに対し ＨＤＬ 記述を与えておいて、フィッティングでは最高の成果を得たいという場合には、前に紹介した例題のように構造記述を使って LPM マクロや XBLOX マクロをそのまま記述し、マクロの良さをそのまま利用する方法を採用すれば良い訳ですし、このような場面では、コンフィギュレーションの切替えが使われることは既によくご存知のとおりです。

なを、余談になりますが、アルテラの maxplus2 で ＥＤＩＦ ファイルをコンパイルしたとき、フロアプラン・エディタ画面などでチェックすると、入力 ＥＤＩＦ に定義されていて実フィット時には削除されてしまうような素子があります。

これらの大部分は、レオナルドが生成したエキスパンダやソフトマクロのインスタンスで、ルータが不要と判断したもので、同じように不要として削除されるインスタンスがザイリンクスでのフィッティング・レポート内でも見られますが、これらについては無視するだけでよいのではないかと思います。

「この程度の比較検討で、いったい何がわかると言うのか」大いに疑問の残るところですが、自分が開発に使うツールについては、なるべく多くの知識を得ておきたい筈ですから、実際にやって見て、それなりに得た結果をそれなりに参考にするのは悪い事ではないように思います。

言語設計か、回路図設計か という議論の焦点は、おそらく別の次元の事を指している場合もある筈で、たとえば、設計者の意図をより正しくターゲットに反映させるにはどちらの方法がよいかという事が大きなポイントになるかも知れません。

これについての例題を、最後にご紹介しますので検討して自分なりの結論をだされるようお勧めします。

ＦＰＧＡ の設計について注意したい事

実装効率について

実際の ＦＰＧＡ では、高速になるに従って消費電力の問題がより大きくなってきますので、使用できる ＣＬＢ／ＬＣＥＬＬ などの数はそれほど多くとれるものではありませんので、あまりコンパイラの実装効率を問題にしても仕方が無いような気がします。

現在の ＦＰＧＡ デバイスで、特にパッケージあたり２００００ゲート規模以上のデバイスを検討する場合には、フィッタの実装効率よりもむしろ、消費電力がそのパッケージと見合っているか否かの判定のほうが重要な意味を持っているケースが多いのではないかと思います。

ＦＰＧＡ の電力積算の資料では、一般的な論理回路での内部 ＦＦ のトグル率を１２．５％程度に考えて処理することが多いように見えますが、例えばデジタル・フィルタなどのようにパイプ・ラインを組んだ回路で、微弱な ＡＣ 信号の符号付き固定小数点演算などを想定した場合、この１２．５％という値が事例に対して適切であるか否か、慎重に判定する必要が出てくる事は明らかです。

４００００ゲート規模以上の ＦＰＧＡ を使う場合、本当に高速な回路を実現するのであれば、５ Ｖ 電源デバイスを採用する事は非常に不利であると言えます。

テスタビリティ

テスト一般論を述べ立てるほどの力量を持っている訳ではありませんので、どうしても断片的な物言いになってしまいますが、最終回ということであえて続けてみます。

デバッグを考慮した実装設計を行う場面では、レジスタのリード・バック回路のような機構を設計内容に加えて組込むことも（最悪の場合には）採用せざるを得なくなる事があります。このような場合の為の参考として「ＶＨＤＬ６」でＪＴＡＧについての簡単な解説をしておきましたので参照してください。

固定値リードバック回路を組込み、データバスが正常に接続されているか否かを判断することは比較的小規模な回路の追加で可能ですが、たとえば、レジスタアドレスが正しくデバイス内部に伝達されているか否かを調べる機構を組込む事は、本来の設計に与えられるべきスペース、特に貴重な ＦＦ、を占有してしまう事になりますので、どうしても実施できない場合がでてきます。　このような場合には、余程小さい回路規模であれば別ですが、予め基板にテスタをかけておくしか方法がないものと思います。

どれほど複雑であっても、明確な設計意図がある限り、ターゲットの回路設計それ自体は比較的容易な作業になるものだと思います。　これに対し、そのテスト設計がより困難な作業になる原因の中に、不精確なブロック図や検査仕様の欠落にその発端がある事も無視できないのではないでしょうか。

このシリーズでは、ＨＤＬ 設計で論理合成に使用できない言語仕様としてシミュレータ用記述があることを、TEXTIO の実例で幾度か紹介してきています。

このシミュレータ用の記述方法は、システム設計意図を第３者に受け渡す方法の１つとして、比較的手軽な利用ができるものだろうと思います。

無論、もっと手軽なアルテラ社のタイミング・シミュレータを利用する方法もあるのですが、こちらの場合は、実設計ができていないと、そもそも利用すること自体ができませんので、仕様書的な利用方法としては不適切です。

テストポイントは、普通は個別のデバイス選択信号ピンの近傍に置きます。

「高速信号」のテストポイントは、必ず抵抗によるデバイダを介して使用するように設計し、かつ、使用するプローブの先端をグリップできるように Tektro などの ＳMC ライクなプローブ・チップ・ホルダを設置します。　プローブのＧＮＤ を最短距離で落とせるような ＧＮＤ パターンをテスト・ポイントにペアで設置しておくようにする事で、より正しい測定結果が得られます。

１本の伝送線路上に複数のデバイス・ピンが接続されているような基板の場合には、できれば ＩＢＩＳ モデルを使った簡単な伝送線のシミュレーションをかけておくと、波形の劣化についての大雑把なイメージがつかめます。

サンプリング・オシロのような本格的測定器を持っている場合には、もっと詳細な実データを得ることができますので（プローブを壊さないよう、適切なパワー

デバイダを装着して測定します）実基板のデータから知識を得る事には大きな価値があるだろうと思います。　今回のテーマと同じように、実際にやってみて初めて概要が把握できる事が、他にもまだまだあるはずだと思います。

マルチ・クロックの回路では

基本的にはメタスタビリティについてよく検討しておくべきだと思います。

例外的に、複数のクロックの周波数の間に、最大公約数的な値を設定できる（本質的には同期回路）ような場合の設計であれば簡単ですが、複数のクロックの周波数の間に大小関係が一定して成立しないような、互いに非同期なクロックを持つようなシステムでは、信号の受け渡しを行うそれぞれの回路ブロック間の双方向について、ハンドシェーク回路が必要となると思います。

このような場面では、とりわけ、「互いのクロック周波数／位相が交差する場面」で、ハンドシェーク回路内の ＦＦ に発生するメタスタビリティについての認識と解析とが必要で、信号を接続するだけで、これを簡単に解決してくれるような便利な素子を実装している ＦＰＧＡ の実例については、残念ながら私はまだ見た事がありません。

最も確実に設計全体を検証する為には、クロックの大小関係を変化させたシミュレーションを行い、クロックに載せた信号パルスの数が正しく対方のクロック回路に渡される事を確認する方法を使うしかないのではないかと思います。

この実例は既に提出してあるソースファイル中に幾度も繰り返し組込まれているので参照してみてください。

互いに受け渡す信号の性格・意味が、パルス的な信号であるか、フラグ（セマフォ）的な信号であるか、によって回路の実装方法が異なりますし、それを使うシステム側が互いにその時点で待ち合わせを行うのか、単に通過するだけでよいのか、も検討しておくべきだろうと思います。

ステートマシン設計での最大の注意事項としては、空きピンの適切な処理や、マルチ・クロックなどに起因する周辺 ＦＦ のメタ・スタビリティ条件を含め、入力信号のクロッキングに完璧を期する努力が必要になると思います。

本来有り得ない others 文節への分岐が実際に起きたとすると、この回路動作上の問題を引き起こす原因は、ステートマシンの条件を設定する入力信号がセット

アップ・ホールド時間を守らない（守れない）設計・仕様（あるいは基板パターン）になっている場合ではないでしょうか。

ここでもし、電源性のノイズやアルファ線、などに言及することになれば、もはやデバイス内部の回路設計という前提要件が崩れた所で議論している事になり、設計論議の枠から解脱してしまうでしょうし、反射ノイズなどによってクロックの波形品質が崩れているような基板条件では、どんなに ＦＰＧＡ の回路設計をいじくり回しても、その結果に責任はとりにくいだろうと思います。

ステートマシンの設計での注意事項

ジョンソンカウンタについて、ブービー・トラップを詳細に力説している書籍の記憶がありますが、要点を一言で言えば「冗長性を持つ論理では、その冗長部分についての適切な処理方法を用意しなさい」という事でしょう。

トラップが発生した時、その回路は確実に誤動作をおこしているはずです。

トラップ回路や when 構文での Others 文節は誤動作抑止法ではありません。

次の例題ではこの冗長性という事について考えます。

組込まれたジョンソン・カウンタが、セットアップ・ホールドを正しく守っていても誤動作を起こすような環境条件では、カウンタであれ、どのような回路であれ、やはり誤動作を起こす可能性を持つ事になると思いますが、それを検定する方法を私は現在、知りません。

その回路を使うシステムにとって、トラップを起こさせて、ただ単に回路を復旧させる事が必要・十分な仕様であるのか、あるいは、トラップが発生した事を知ってシステムに警報を出して対応を取る事が必要なのか、という事も考えなければ無価値な論理を作成することになるかもしれません。

ブービー・トラップがジョンソン・カウンタで特に重要な意味を持つのは、システムがそこに一旦落ち込んだ後は二度と脱出ができないような回路構成になってしまっている場合で、システム全体として適切な外部リセットなどの方法が用意されていれば必ずしも必要なものとは言えないでしょう。

ブービー・トラップ回路についてのこの議論は、元々が回路の冗長性に起因するものですから、ステートマシン記述でよく出会う構文、Case / when / others 文節の使用でも同じ事が言えるでしょう。

VHDLでの設計の場合ですと、教科書どおりにCase構文などで列挙型のステート・タイプを使って記述した時に、もしすべての Case 条件が when 構文で受け切られてしまっていると、コンパイラはこの others 文節を、論理的に存在し得ないステートであるとして削除し、警告して回路を合成します。

つまり、論理的に存在し得る全てのステートが定義され、次のステートへの遷移条件として使われているならば、そのステート変数には冗長性がない訳ですからトラップ回路は論理的に存在しません。

バイナリ・タイプのステートマシンを組んで、その可能なデコード選択肢全てを when 構文で受け切ってしまった場合にもトラップの余地は当然ありません。

when 構文で必ず others 文節を設定するという事の意味は、C 言語の場合と同様、設計を検証する時に見落しをチェックし易いという理由がありますが、コンパイラはこのような不要な構文に対しては警告を出しますので、逆にこの警告内容を確認する事で、合成された回路を検証するやりかたも在るかも知れません。

再び Enum 型を考える

ステートマシンの定義では、通常、ステート変数のタイプとして Enum を使う場合が殆どではないかと思います。

Enum とは単に列挙型という意味で、その型を指定された変数がインスタンス化された時、バイナリ・コード化された FF 群として実現されるか、独立したビット単位の FF の集合（ワン・ホット）になるか、については何も言っていない事に注意します。

では、この Enum 型のステート変数を Case / when / others 文節で受けるときワン・ホット形式に展開される場合と、それ以外の、たとえばバイナリ形式に展開される場合とで、others 文節はそれぞれどのような論理になるか、を考えてみましょう。

例題として、StateRTL を取り上げます、

StateRTL は４つ（最終的に５つに変更します）のステートを持つマシンです。

Enum の宣言では（s0、s1、s2、s3）などのように定義されているかも知れませんが、これを使ったステート・レジスタはバイナリ形式で 2 ビットに実現されるのでしょうか、それとも其々のステートに対応した 4 ビットのレジスタになるでしょうか。

答えは、「どちらにでもなる」という事です。

レオナルド上での VHDL 記述では、ステートマシン毎に「アトリビュート」を使い、コンパイラに対してワン・ホット／バイナリなどの実装方法を指示する事ができますが、このアトリビュート値はパッケージ exemplar.vhd の中で以下のように宣言されています。

```
-- New attributes in 2.2 release
--  Attributes for encoding of enumerated types.

    type encoding_style is (BINARY, ONEHOT, GRAY, RANDOM) ;
    attribute TYPE_ENCODING_STYLE : encoding_style ;

-- Example of using type_encoding_style for an enumerated type :
--      type state_t is (PLAY, WAIT_FOR_MOVE, END_OF_GAME);
--      attribute TYPE_ENCODING_STYLE of state_t:type is ONEHOT ;

    attribute TYPE_ENCODING : exemplar_string_array ;
```

このような指示を、レオナルドの GUI 環境から制御するには、ソースコード読込み段階で ADVANCE ダイアログを開き、指定したいスタイルを設計全体に対して指定することでも処理できます。

個別のステートマシンに対して別々の指示を出すには、コマンド・プロンプトから set_attribute を使って個別のビュー毎に指定する方法も使えますし、ソースコード内で上の引用中 赤で示してあるような記述を行う事でも処理が可能です。(当然、これはコンパイラ依存で、かつ、このパッケージを読込んである前提ですから、他社の VHDL リーダにはこのままでは原則無効です）

レオナルドでは、ワン・ホットを指示してコンパイルした場合とバイナリを指示した場合とで、パッケージに定義したファンクション文中の when others 文節がどのように異なった処理に展開されているかを見ることができます。

アルテラでのコーディング・スタイルの処理の選択肢は、グローバル・プロジェクト・ロジック・シンセシスのメニューで「ワン・ホット項目」をチェックするか否か、だけなのですが、その結果は常にバイナリコーディングに展開されるように見えますので、私は現時点でかなり不審に思っており「信頼できる処理系ではないのではないか」と不安を持っています。

まとめの例題

今回の例題では、実際にバイナリとワンホットが混在する簡単なステートマシンを作成して、レオナルドでの合成の変化を見ることにしましょう。

Enum の復習 StateRTL

今回の目的に合わせ、ワン・ホット型とバイナリ型のステートマシンを（太字表示）下リストのようにパッケージ「MikiPack」で組込んでおきます。

```
--=====================================================================
--A STATE MACHINE
--=====================================================================
use work.MikiPack.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
---------------------------------------------------------------------
Entity StateRTL is
---------------------------------------------------------------------
  port( RESET :in Std_Logic;
        DS    :in Std_Logic;
        ACK   :outStd_Logic;
        SRQ   :in Std_Logic;
        SGRNT :outStd_Logic;
        SCLK  :in Std_Logic;
        CountNow:buffer Count_Type;   --   <--- (^^;)/
        XERROR:out Std_Logic          -- Watch this result carefully.
        );
End StateRTL;
---------------------------------------------------------------------
Architecture RTL of StateRTL is
---------------------------------------------------------------------
  signal StateNow,StateNXT: State_Type;
  signal DSFF,SRQFF,FATAL: Std_Logic;
Begin
  XERROR <= FATAL;
  MainSEQ:Process (SCLK,RESET,DS,SRQ,StateNow,StateNXT,CountNow)
  Begin
    ONEHOTPLA( DSFF,SRQFF,StateNow,StateNXT,FATAL);
    if(RESET='1') then
        DSFF<= '0';
        SRQFF<= '0';
        StateNow<= Idle;
        CountNow<= S0;
    elsif(SCLK'EVENT and SCLK='1') then
        DSFF <= DS;
        SRQFF<= SRQ;
        StateNow<= StateNXT;
        CountNow<= BINARYPLA(DSFF,CountNow);
    end if;
  end Process MainSEQ;-----------------------------------------------

  with StateNow select
    ACK <= '1'when TWAIT,'0'when others;
  with StateNow select
    SGRNT <= '1'when TSRQ,'0'when others;
End RTL;
```

これまでのシリーズの例題で既にパッケージについては実例を見てきていますので特に説明はしません。ただ、プロシージャを使った場合には複数の外部値に影響を与える副作用に気を付けなさいという指摘が殆どの書籍中に明記してありますので、その点には注意します。

ここでは、バイナリにエンコードするシーケンサ **BINARYPLA** は関数として、もう一つのワン・ホットにエンコードするシーケンサ **ONEHOTPLA** はプロシージャとして作成してありますが、別にどのような組合わせを行っても良い筈です。

このパッケージについて、others 文節に関連する実験をしてみますが、その前に簡単にパッケージの中をチェックしておきます。

パッケージ宣言部にこの実験の目的を記述してあります。

```
--=============================================================--
PACKAGE MikiPack is
--=============================================================--
Type State_Type is (  Idle,T0,TWAIT,TSRQ
                     ,TDUMMY
                     );
  --  If U reject this TDUMMY line from the ENUM listing,"when others"
  -- clause will be ignored and will not trap any failure anyhow U've
  -- assigned ONE-HOT or BINARY or ANY TYPE of encoding styles.
  -- Notice: This is a very LOGICAL result.
   attribute TYPE_ENCODING_STYLE of State_Type:TYPE is ONEHOT ;

Type Count_Type is (S0,S1,S2,S3);
   attribute TYPE_ENCODING_STYLE of Count_Type:TYPE is BINARY ;
-- When U have chenged above BINARY setting to ONEHOT, then, leonardo
-- willgenerate a 4-bit ONEHOT Counter. Provided U've followed notices
-- above.

Procedure ONEHOTPLA(  DSFF:       in Std_Logic;
                      SRQFF:      in Std_Logic;
                      StateNow:   in  State_Type;
                  signal StateNXT:out State_Type;
                  signal FATAL:   out Std_Logic   );

function BINARYPLA(   CEN :   in Std_Logic;
                  CountNow:   in Count_Type ) return Count_Type;
end MikiPack;
```

ここに５つ目の **TDUMMY** を入れた意味を実験します。

ここでは、アトリビュートを使って、エンコーディングスタイルをレオナルドに与える方法の実例として赤表示の部分に、また、実際に実験をする為に、紫表示の部分にも注意しておいて下さい。

パッケージ宣言では以下の呼び出しを行っておきます。

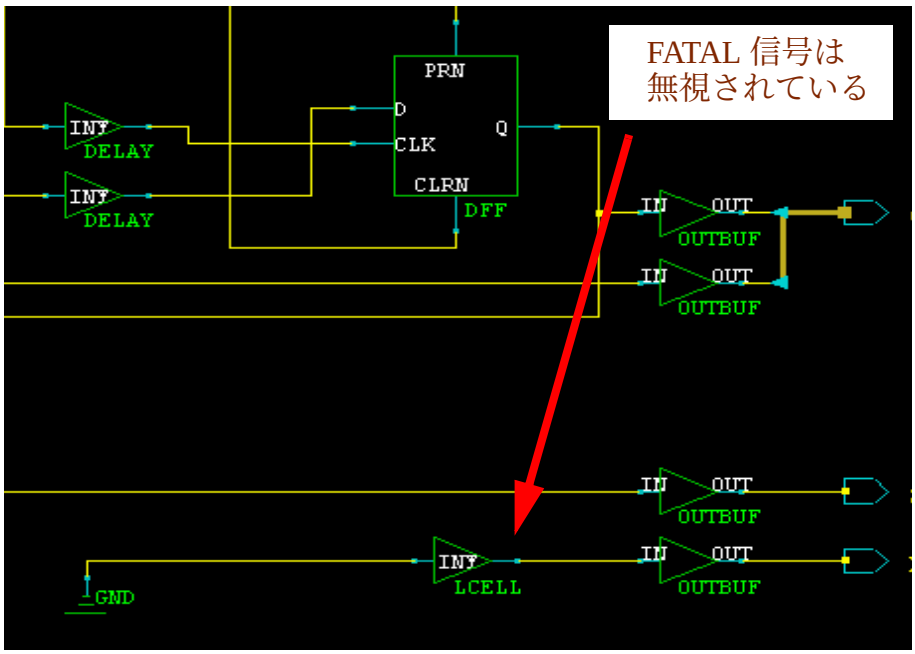
```
use work.exemplar.all;
```

パッケージの内容（PACKAGE body）は、次ページに示してあります。

パッケージの定義部

```
--=================================================================--
PACKAGE body MikiPack is
--=================================================================--
Procedure ONEHOTPLA(DSFF:   in Std_Logic;
                    SRQFF:  in Std_Logic;
                    StateNow: in  State_Type;
            signal StateNXT : out State_Type;
            signal FATAL :    out Std_Logic
            ) is
    variable NextS: State_Type;
  Begin
    CASE StateNow is
        when Idle=>
            if(DSFF='1') then       NextS:=T0;
            elsif(SRQFF='1') then   NextS:=TSRQ;
            else                    NextS:=Idle;
            end if;
            FATAL <='0';
        when T0 =>                  NextS:=TWAIT;
            FATAL <='0';
        when TWAIT=>
            if(DSFF='1') then       NextS:=TWAIT;
            else                    NextS:=Idle;
            end if;
            FATAL <='0';
        when TSRQ=>
            if(SRQFF='1') then      NextS:=TSRQ;
            else                    NextS:=Idle;
            end if;
            FATAL <='0';
        when OTHERS=>               NextS:=Idle; -- NextS:=TDUMMY;
            FATAL <='1';
    end CASE;
    StateNXT <= NextS;
end ONEHOTPLA;

function BINARYPLA( CEN:        in Std_Logic;
                    CountNow:   in Count_Type) return Count_Type is
  variable NextC:   Count_Type;
  Begin
    if( CEN='1') then
        CASE CountNow is
            when S0 =>NextC := S1;
            when S1 =>NextC := S2;
            when S2 =>NextC := S3;
            when S3 =>NextC := S0;
        end CASE;
   end if;
   return NextC;
end BINARYPLA;
```

パッケージ宣言で TDUMMY として追加してある部分をコメントアウトしてコンパイルし、その結果、FATAL / XERROR 信号がどのように展開されたかを見ます。

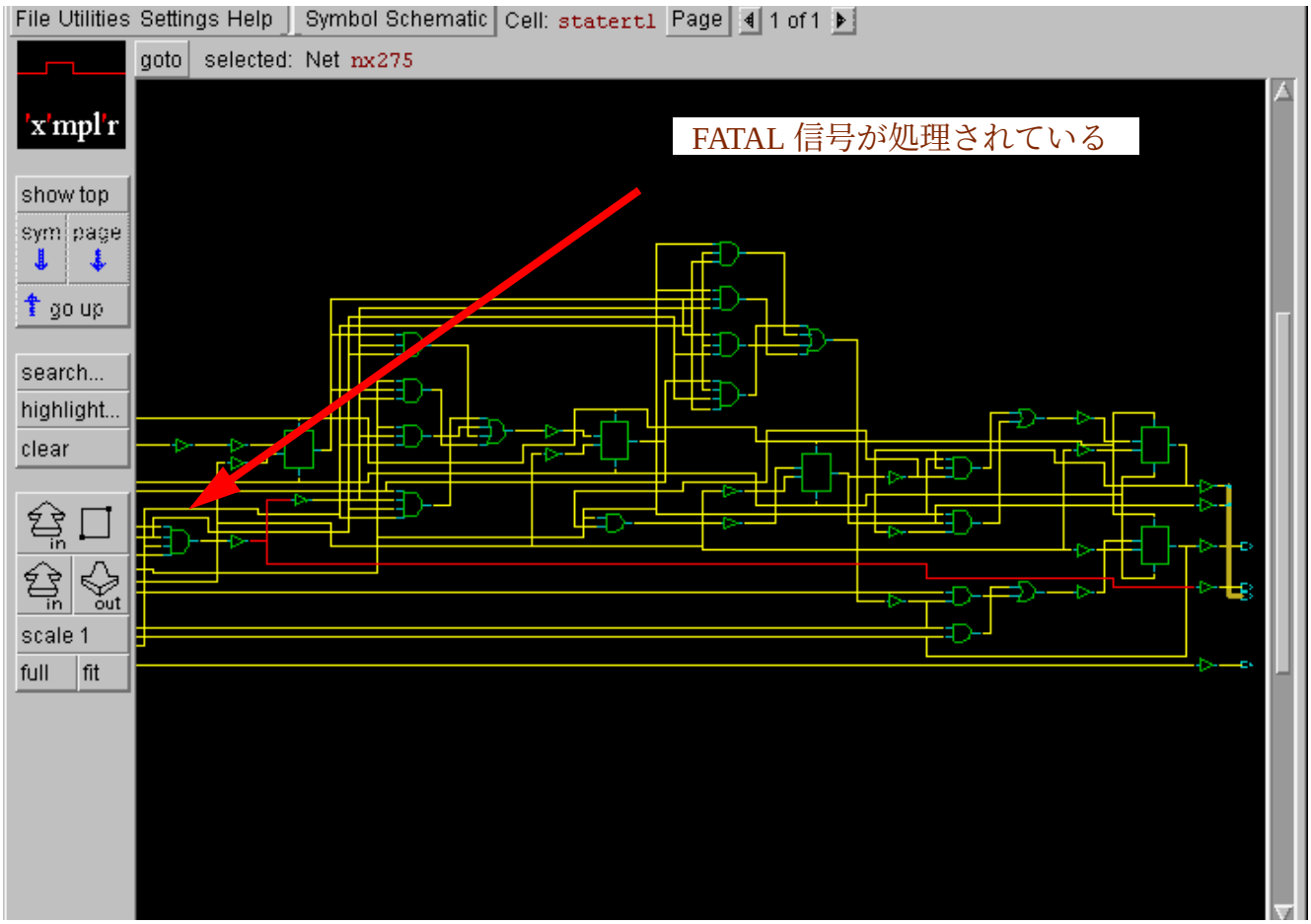


このように、存在し得ない論理条件から派生する信号について、いくら Enum タイプを宣言し、ワン・ホットを指定して、小手先で信号を強引に引き出したとしても、具体的な意味を持った論理には展開されない事がわかります。

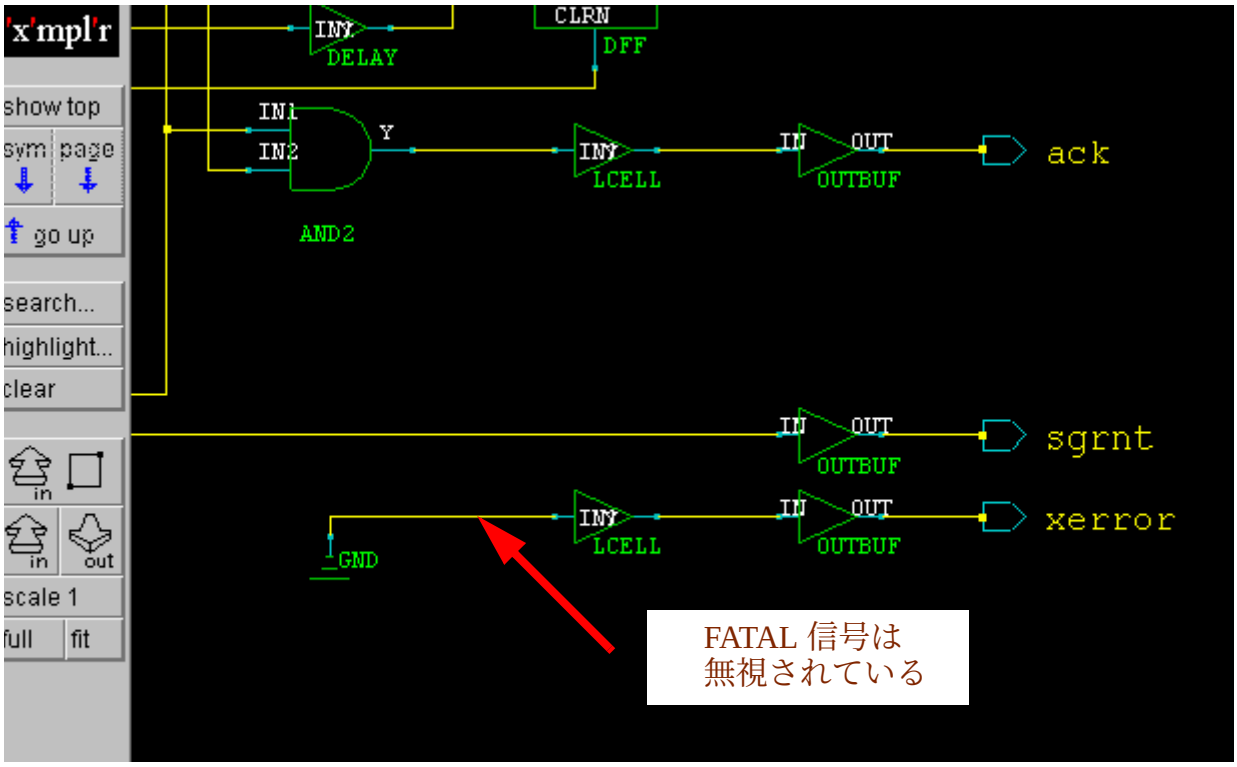
レオナルドは警告するだけで、これをエラーとして扱わないことには十分に注意する必要があります。

"D:/miki.dir/statertl.vhd",line 70: Warning, others clause is never selected.

コメントを外し「論理的な」冗長性を持たせた結果の同ネットを赤で示します。



ONEHOTPLA として宣言したステートマシンをバイナリに作り替えるのは、単にパッケージ宣言部で指定したアトリビュートを書き換えるだけですが、では、果たしでそれだけで十分でしょうか。結果を示します。



つまり、パッケージ定義部分で、CASE　StateNow　is として各分岐条件を設定している部分で、全ての意味を定義し切ってしまっている所を考えると、論理的に、第５のステート **TDUMMY** への遷移は有り得ないことになっている事に気がつくでしょう。

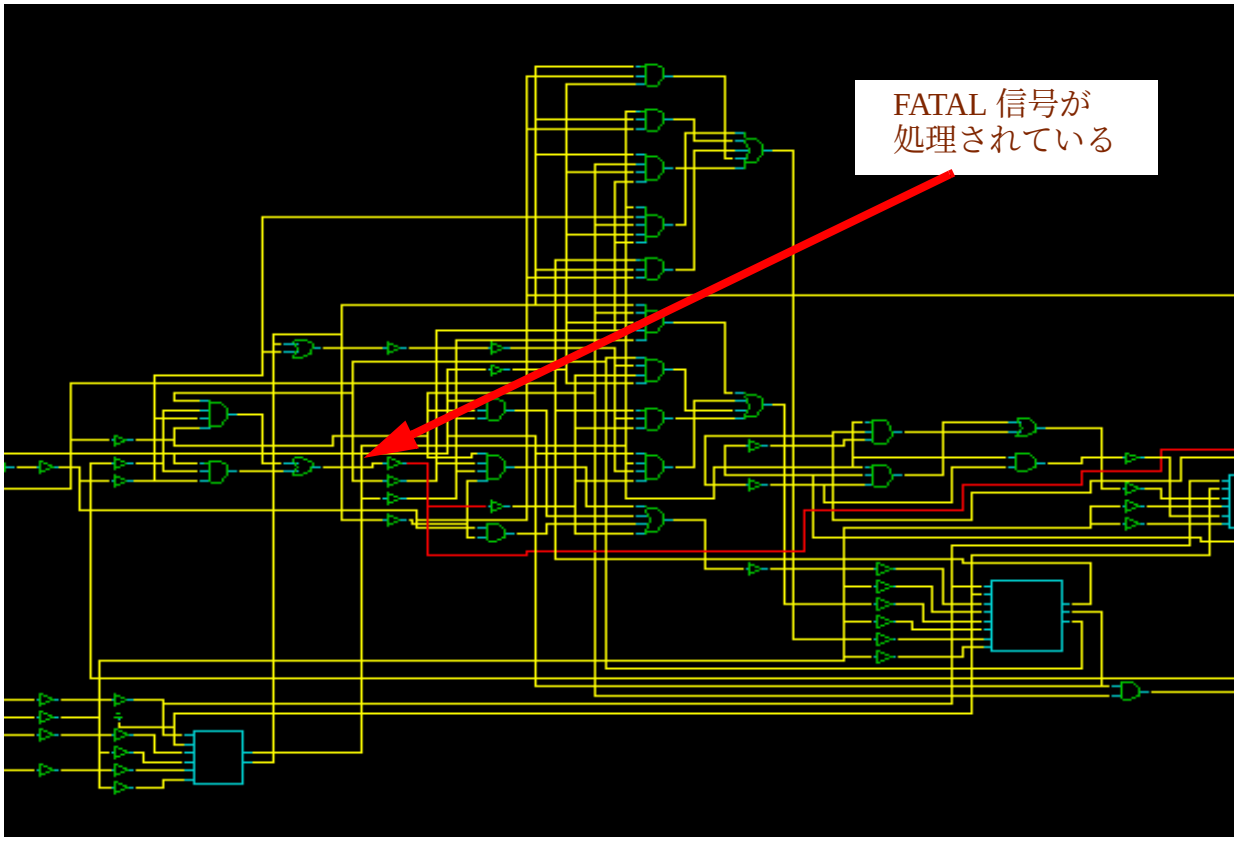
```
when OTHERS=>   NextS:=Idle;
      FATAL <='1';
end CASE;
```

この部分を意図的に **TDUMMY** に接続し、バイナリコーディングを指定します。

```
when OTHERS=>   NextS:=TDUMMY;
      FATAL <='1';
end CASE;
```

結果は次ページに示すとおり、ステート用の ＦＦ は３ビットのバイナリに展開され、目的の FATAL 信号は出力ピンにそれなりの論理回路を経由して接続されるようになっている事が判ります。

なを、次ページの図はデザインブラウザから ＤＦＦ をグループ化して回路全体を見易くなるように処理したもので、この方法は既に ＶＨＤＬ １１で説明してある通りです。



同じような操作を行って、カウンタをワン・ホットスタイルに実装することもできますし、それぞれのケースについて納得のゆく結果を得るように繰り返し実験（これはあくまで私なりのやりかたに過ぎませんが）されるようお勧めします。

誤解されては困るのですが、ここではあくまでＶＨＤＬでの「論理的な考え方」というものについて、Enum を例にとって検証を行っているだけで、この程度の回路を設計したからといってＦＰＧＡの誤動作を検出できる回路を作成したのではありません。

ここでの議論は、本稿の冒頭で
「言語設計か、回路図設計か という議論の焦点は、おそらく別の次元の事を指している場合もある筈で、たとえば、設計者の意図をより正しくターゲットに反映させるにはどちらの方法がよいかという事が大きなポイントになるかも知れません」
と言った部分について、そのひとつの例証を見てみただけの事です。

運よくこのトラップに落ちるような障害があれば、何かの検出ができるかも知れませんが、どのような回路を組み立てようと、ＦＰＧＡ のクロック規約やセットアップ／ホールド規格を崩した場合には、「誰も何も保証できない」筈です。

非常に大切な事で、市販の書籍では殆ど触れられていない項目であるように思われましたので（それとも、常識項目なので説明は不要とか (--;))、Enum についてづいぶん時間を割いてしまいました。

最後のしめくくりに、非同期マルチクロックでのハンドシェークの例題を取り上げたいと思っていたのですが、考えていたよりも時間が経過してしまっていますので、これについては最終回の後半部分（^^;) に例題を挙げて検討してみます。

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.exemplar.all;  -- DEFAULT D:\exemplar\leonardo\data\exemplar.vhd

--============================================================================--
PACKAGE MikiPack is
--============================================================================--

Type State_Type is (Idle,T0,TWAIT,TSRQ
                    ,TDUMMY
                    );
    --  If U reject this TDUMMY line from the ENUM listing,"when others"
    -- clause will be ignored and will not trap any failure anyhow U've
    -- assigned ONE-HOT or BINARY or ANY TYPE of encoding styles.
    -- Notice: This is a very LOGICAL result.

    attribute TYPE_ENCODING_STYLE of State_Type:TYPE is BINARY ; -- ONEHOT ;

Type Count_Type is (S0,S1,S2,S3);
    attribute TYPE_ENCODING_STYLE of Count_Type:TYPE is BINARY ;
    -- When U have chenged above BINARY setting to ONEHOT, then, leonardo will
    -- generate a 4-bit ONEHOT Counter. Provided U've followed notices above.

Procedure ONEHOTPLA(DSFF    : in Std_Logic;
                    SRQFF   : in Std_Logic;
                    StateNow: in  State_Type;
              signal StateNXT : out State_Type;
              signal FATAL    : out Std_Logic
              )  ;

function BINARYPLA( CEN     : in Std_Logic;
                    CountNow: in Count_Type ) return Count_Type;

end MikiPack;

--============================================================================--
PACKAGE body MikiPack is
--============================================================================--

Procedure ONEHOTPLA(DSFF    : in Std_Logic;
                    SRQFF   : in Std_Logic;
                    StateNow: in  State_Type;
              signal  StateNXT: out State_Type;
              signal  FATAL   : out Std_Logic
              ) is
    variable NextS: State_Type;
  Begin
    CASE StateNow is
        when Idle   =>
            if(DSFF='1') then       NextS:=T0;
            elsif(SRQFF='1') then   NextS:=TSRQ;
            else                    NextS:=Idle;
            end if;
            FATAL <='0';
        when T0     =>              NextS:=TWAIT;
            FATAL <='0';
        when TWAIT  =>
            if(DSFF='1') then       NextS:=TWAIT;
            else                    NextS:=Idle;
            e       end if;
            FATAL <='0';
        when TSRQ   =>
            if(SRQFF='1') then      NextS:=TSRQ;
            else                    NextS:=Idle;
            end if;
            FATAL <='0';
        when OTHERS =>              NextS:=TDUMMY; -- NextS:=Idle;
```

```
                FATAL <='1';
            end CASE;
            StateNXT <= NextS;
        end ONEHOTPLA;
75

        function BINARYPLA(     CEN : in Std_Logic;
                            CountNow: in Count_Type ) return Count_Type is
            variable NextC: Count_Type;
80        Begin

            if( CEN='1') then
                CASE CountNow is
                    when S0 =>  NextC := S1;
85                  when S1 =>  NextC := S2;
                    when S2 =>  NextC := S3;
                    when S3 =>  NextC := S0;
                end CASE;
           end if;
90         return NextC;
        end BINARYPLA   ;

        end MikiPack;   ------------------------------------------------------------------

95
        --=============================================================================--
        --  A STATE MACHINE
        --=============================================================================--
        use work.MikiPack.all;
100
        library ieee;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_unsigned.all;
        ------------------------------------------------------------------------------
105     Entity StateRTL is
        ------------------------------------------------------------------------------

            port(   RESET   :in  Std_Logic;
                    DS      :in  Std_Logic;
                    ACK     :out Std_Logic;
110                 SRQ     :in  Std_Logic;
                    SGRNT   :out Std_Logic;
                    SCLK    :in  Std_Logic;
                    CountNow:buffer Count_Type;     --   <--- (^^;)/
                    XERROR  :out Std_Logic          -- Watch this result carefully.
115                   );
        End StateRTL;


        ------------------------------------------------------------------------------
120     Architecture RTL of StateRTL is
        ------------------------------------------------------------------------------
            signal StateNow,StateNXT    : State_Type;
            signal DSFF,SRQFF,FATAL     : Std_Logic;
        Begin
125
            XERROR <= FATAL;

            MainSEQ:Process (SCLK,RESET,DS,SRQ,StateNow,StateNXT,CountNow) ------------
            Begin
130
                ONEHOTPLA( DSFF,SRQFF,StateNow,StateNXT,FATAL );

                if(RESET='1') then
                        DSFF    <= '0';
135                     SRQFF   <= '0';
                        StateNow<= Idle;
                        CountNow<= S0;
                elsif(SCLK'EVENT and SCLK='1') then
                        DSFF    <= DS;
```

```
                    SRQFF   <= SRQ;
                    StateNow<= StateNXT;
                    CountNow<= BINARYPLA(DSFF,CountNow);
                end if;
            end Process MainSEQ;    ------------------------------------------------
140
            with StateNow select
                ACK     <= '1'  when TWAIT, '0' when others;
            with StateNow select
                SGRNT   <= '1'  when TSRQ,  '0' when others;
150
        End RTL;
        ------------------------------------------------------------------------------
```