VHDL- 3

この文書の目的

前回のＶＨＤＬ－２ではメモリーの「シミュレーション」というテーマで入門的な説明をしたのですが、今回のメモリーは「実デバイスへの実装」についての説明です。

アルテラ社の１０Ｋファミリーに固有のテクノロジーである「ＥＡＢ」にメモリーを実装する事を考えてみます。

ＲＯＭ／ＲＡＭを包括した「メモリー」について考えれば、基本的にはＦＰＧＡ内部にこれを実装するのは、現状ではごく限られた条件の下での限定利用に止まるべき（一般的なＦＰＧＡでは１つのＦＦが１ビットのメモリー機能に対応します）だろうと思われます。

現行のレオナルド４．２２では、VHDL 記述の文脈から「メモリー」を推論し、これを任意のターゲットに（？）コンパイルすることができるように説明しています。この「推論されるメモリー」という記述は、実際には「ＲＡＭ」機能に限定されている事を十分に認識しておく必要があります。

アルテラ１０ＫのＥＡＢにＲＯＭを実装することのメリットは、ＲＯＭ内容の変更に対し、その内容を定義する myrom.mif などのテキストファイル内容の修正を行って再フィットだけで処理が完了できる事にあります。

この場合にはＲＯＭサイズ自体に対する変更がなければ、そのＦＰＧＡの他の部分のＰＬＡ／レジスタ回路に対する変更は発生しませんので、設計全体のタイミング性能は基本的には変化しないと考える事ができます。

ＥＡＢに実装しなかった場合、もしもＲＯＭデータの内容自体が論理回路によって実現されている場合には、このデータの変更は全ての論理回路の再合成を必要とすることになりますので、設計全体のタイミング性能は全面的に検証しなおす必要が発生します。

このような事態を回避するためには、ＲＯＭの内容を論理回路によって構成するのではなく、初期化されるレジスタとアドレス選択用マルチプレクサとによって構築するような、ターゲット側の特性に依存した別の手段（例えば memgen／ブロックスなど）を採用する事になり、本質的には今回の例題のＥＡＢへのフィッティングと同じような議論になります。

同様に、サイズの限定されたRAMブロックをアルテラ１０ＫのＥＡＢに実装することができた場合、大幅な速度の向上とＦＰＧＡ関連デバイス・回路のドラスティックな整理・縮小効果が期待されます。

今回は、このようなターゲット依存の明示的な実装方法と、レオナルドの文脈推論によるＥＡＢ／ＲＡＭメモリーの自動合成とを説明しようと思います。

これまでにシューティング・ユニットの開発で作成してきたＶＨＤＬソース記述では、これに類似した事例が幾つもでてきています。　例えばザイリンスクやアクテルのクロック・バッファの割り当て方法や、その配線資源の接続の記述でも同じような手法（要するにテクノロジ・ライブラリからコンポーネントを持ってきて構造記述でネットを付ける、という方法）を使ってあります。　このような単純なコンポーネントを接続するにはポート名を知るだけでも十分だったのですが、今回の例題ではもう少しだけ複雑に、目的のコンポーネントに引き渡すパラメータについての記述が入ってきます。

説明主文のオリジナルはソースファイルに英文で表記してありますので、そちらも合わせて参照してください。

前提要件

前回同様、作業用ディレクトリを作成します。　今回は d:\boo として、ここに添付したサンプル・ファイル lpm_ram.vhd をコピーしておきます。

実デバイスへのフィッティング・テストができるように、アルテラのコンパイラ環境が c:\maxplus2 にフル・インストールされているものとします。

この時、c:\maxplus2\vhdl87 (または vhdl93) \ lpm ディレクトリには、アルテラ社が提供する lpm 用ライブラリが揃っています。

次の作業の簡便の為 d:\boo には、lpm_ram.vhd と c:\maxplus2\vhdl87 \lpm \lpm_pack.vhd というパッケージもコピーしておくと良いでしょう。　これはＣ言語で言うヘッダー・ファイルのような役割を果たしてくれます。アルテラ側でのオブジェクトに相当する .dls ファイルは maxpuls2 でのフィッティング時に使用されるもので、レオナルドとの直接的な関係はありません。

レオナルドを起動して

いつものように作業ディレクトリを設定します。今回は d:\boo です。Menu/File/ChangeWorkingDirectry を選択し、このディレクトリに移動して

```
cd D:/boo
```

FlowGuide ボタンから altera flex10k を選び RunFlowGuide ボタンを押します。
表示されたダイアログボックスから、テクノロジライブラリ、モジュールジェネレータを Altera FLEX10K を指定して読込みます。

load_library flex10

load_modgen flex10

つぎに、予めコピーしてある lpm_pack.vhd のほうを先に読込み、

auto_read ALTERA -work work -format VHDL D:/boo/Lpm_pack.vhd

最後に lpm_ram.vhd を読込んで

auto_read ALTERA -work work -format VHDL D:/boo/lpmram.vhd

オプティマイズをかけます。

optimize .work.lpmram_16_8.alterafit -target flex10 -effort Quick -chip -area

どのように展開されたか確認してみましょう

report_area -cell_usage -all_leafs

```
*****************************************************

Cell: lpmram_16_8    View: alterafit    Library: work

*****************************************************


Cell       Library  References     Total Area

OUTBUF     flex10    16 x      1     16 OUTBUF
INBUF      flex10    27 x      1     27 INBUF
lpm_ram_dq_16_LPM_RAM_DQ_8_UNUSED_UNUSED_REGISTERED_REGISTERED_REGISTERED_UNUSsed   work      1
x                                                 1                                          1
lpm_ram_dq_16_LPM_RAM_DQ_8_UNUSED_UNUSED_REGISTERED_REGISTERED_REGISTERED_UNUSED

 Number of ports :                      43
 Number of nets :                       86
 Number of instances :                  44
 Number of references to this view :     0

Total accumulated area :
 Number of OUTBUF :                     16
 Number of INBUF :                      27
Numberoflpm_ram_dq_16_LPM_RAM_DQ_8_UNUSED_UNUSED_REGISTERED_REGISTERED_REGISTERED_UNUSED:     1
```

このように、lpm マクロに展開されている事がわかります。
作成された設計を EDIF で出力しますが、念のため、回路ビューを見てみるのも良いアイデアかも知れません。

view_schematic

（作成された回路図は lpm.pdf として添付してあります。）

auto_write -format EDIF ./lpmram_16_8.edf

```
-- Applying renaming rule 'ALTERA' to database
Warning, Renaming will cause your database to change
-- Calling set_altera_eqn to set up writing Equations
Info: setting edif_eqn_or to +
Info: setting edif_eqn_and to
Info: setting edif_eqn_not to '
Info: setting edif_eqn_not_is_prefix to FALSE
Info: setting edif_function_property to lut_function
-- write -format EDIF ./lpmram_16_8.edf
-- Writing file ./lpmram_16_8.edf
```

アルテラ側でこれを受け取るには

このように作成された lpmram_16_8.edf ＥＤＩＦファイルは、maxpuls2 でテキスト・デザイン・ファイルとして新規に開き、カレントデザインに指定しますが、この時、コンパイラメニューの "Edif Reader Setting” で Vendor: Exemplar を指定し、Show LMF Mapping Messages をチェックしておくと良いでしょう。

デバイス指定を１０Ｋ１０あたりの適当なデバイスにしてコンパイルさせ、(特別にグローバル・プロジェクト・ロジック・シンセシスから ＥＡＢ へのフィッティングを指示する必要はありません)、その結果をフロアプランエディタで開いてみれば、この ＲＡＭが１つの ＥＡＢブロックにフィットされている事が分かります。

一度フィットしてしまえば、アルテラ環境の中でのリ・ターゲットは maxpulus2 の中でかなり自由に処理できます。

この例題では、lpm マクロを ＶＨＤＬ 記述で直接的に指定した設計を行いましたが、そもそもこのコンポーネントは、アルテラの ＶＨＤＬ ライブラリから利用できるものを探し出してそれを利用している訳です。

-- package lpm_component is in c:/maxplus2/vhdl87/lpm/lpm_pack.vhd

このパッケージを使うことで、引き渡すべきパラメータやポートの名称を知る事ができますが、その意味や利用方法はアルテラのヘルプ・ファイルを見て理解する事になります。

推論による ＥＡＢ／ＲＡＭの自動合成

もし単に、小さな ＲＡＭ を自由にリ・ターゲットできるような ＶＨＤＬ 記述を考えるのであれば、レオナルドの例題や、前回の例題をコンパイルするだけの事ですので、キーポイントとして記憶部分を配列（アレイ）で定義する事に注意すれば済みます。　このように記述されたメモリに対してレオナルドが提供するサービスは、ターゲット・テクノロジに応じたマクロへの展開機能だけです。

参考の為、ソースファイル mem.vhd を添付しておきます。

まとめ

この例題と同様に lpm マクロを直接に構造記述した方法で ＲＯＭを作成した場合には、ＲＯＭ のデータ内容を設定するのはアルテラのフィッター仕様に依る（myrom.mif など）ものとなって ＶＨＤＬ 記述の対象とはならず、Vsystem でのシミュレーションは、バックアノテートファイル（ .VHO）からのみ可能となります。

従って、トップダウン的なシミュレーションを行いたい場合には、これとは別のアーキテクチャを設計し、そのバイディングをシミュレータ上で指定して実行する事になります。

このコンフィギュレーションを規定する ＶＨＤＬ 記述部分をソースから切り離して別ファイルとして、状況に応じ必要なコンフィギュレーション部分を読込むようにすることで、効果的な設計ができるようになると思います。

次回は、TexiIO について Vsystem 上での説明をしてみたいと思います。

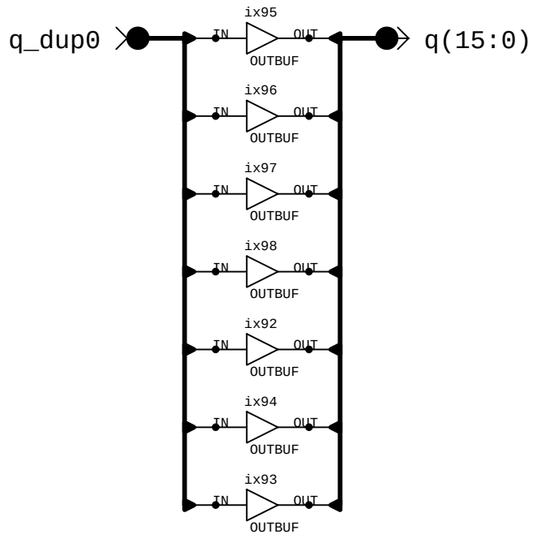
simple lpmram

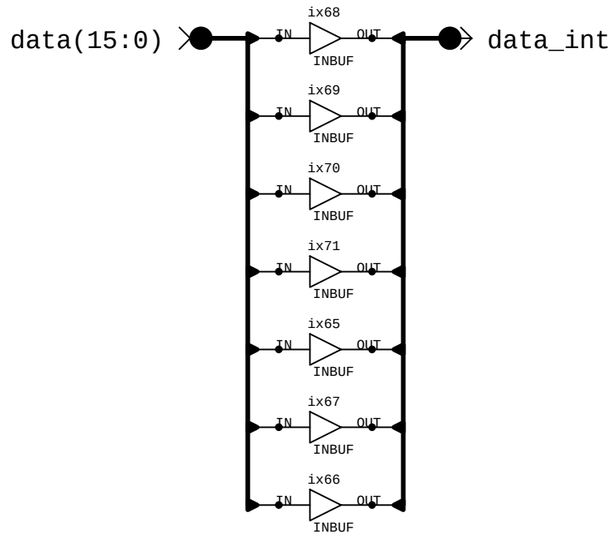
```
--------------------------------------------------------------------------------
--  MaxPlusII basic method to compile RAM into 10K EAB
--------------------------------------------------------------------------------
--  This design is copied from maxplus2 help window and adjusted so that to
--  compile with leonardo synthesis. ( Actually in such cases, there will be
--  nothing to synthesis at all)
--
--   Please be noticed that such description, calling altera's component library
--  and just simply connect signals to it, surely is a non-re-targettable (almost
--  meaningless) vhdl style.
--   Still, if this is the first step of your learning VHDL synthesis, it might
--  be a good example to see how it controls ALTERA-fitting  mechanisms.
--
--   If this design is read into leonardo, and is targeted to 8K or 7K device,
--  the edif output will be tightly related to ALTERA's 8K or 7K device.
--   Though target family is prefixed by EDIF file, if once you compiled it and
--  fitted it to  prefixed device family, you can switch the target device  to
--  any device within ALTERA's product, provided its size and resorces.
--
--   It will be a fun to see this lpmram.vhd fits into a single 10K10 device
--  in 10 seconds, and to see what happens when targetted to 81500, a largest
--  device in 8K family.
--
--  Provided you read the lpm_pack.vhd first then lpmram.vhd next, leonardo
--  compiles the design into edif which is fittable back in maxplus2.
--
--  The simplest way is to copy a lpm_pack.vhd from c:/maxplus2/vhdl87/lpm
--  and read it into leonardo.
--
--    ======== leonardo    ================================================
--       ( Here you prefix target to ALTERA / 10K)
--
--             lpm_pack.vhd    <----  Appliable to ALTERA device only.
--      +)       lpmram.vhd
--      ------------------------
--        lpm_ram_16_8.edf
--                .
--                .
--                .
--    ======== maxplus2 ( edif reader + vhdl-writer )=================
--     ( As prefixed in leonardo phase , device should be 10K)
--
--         lpmr~9cc.XXX   ... vho/pof/sof/etc (  <== lpm_ram_16_8.vho  )
--                .
--                .
--                .
--    ======== Vsystem ============================================
--      cd d:/vsys_doc/lpmram.dir/leo_ram.edf  : change directory
--      project/new d:/vsys_doc/lpmram.dir/leo_ram.edf/vsystem.ini
--      vlib work
--      vcom lpmr~9cc.vho
--      vsim
--      ( Will be a good idea to prepare some test-bench in advance.)
--


library ieee;
use ieee.Std_Logic_1164.all;

library lpm;
use lpm.lpm_components.all;
--
-- To compile this VHDL, you should set @Compile /Menu/Options/UserLibraries
--      Add library path        c:/maxplus2/vhdl87/lpm
--
-- package lpm_component is in  c:/maxplus2/vhdl87/lpm/lpm_pack.vhd
-- lpm_ram_dq  component is in this package.
```

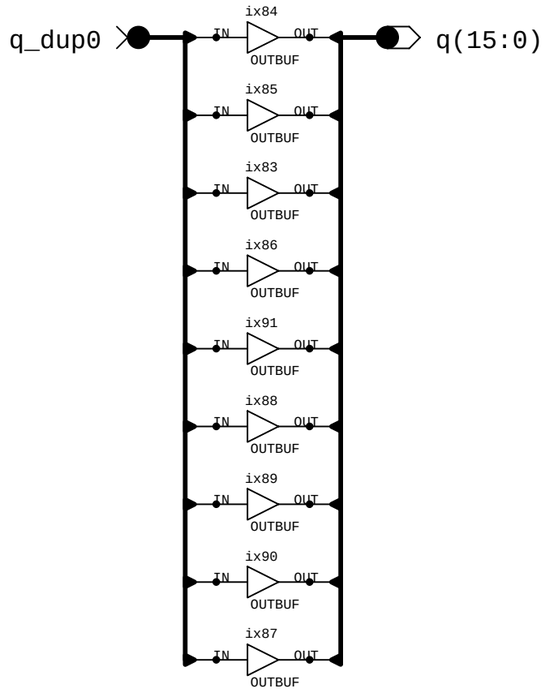
simple lpmram

```
        --    You will find port-names and its-types of the component in this file.

70      -- library work;
        -- use work.ram_constants.all;
        --
        -- This "ram_constants" seems to be generated by higher wrapper
        -- (could be .gdf) and is of no use for vhdl-top design such like
75      -- this example.
        --

        ENTITY lpmram is
            GENERIC(    DATA_WIDTH : natural :=16;      -- Remind you can assign these
80                      ADDR_WIDTH : natural :=8  );    -- values from higher layer.
            PORT(
                data    : IN Std_Logic_Vector (DATA_WIDTH-1 downto 0);
                address : IN Std_Logic_Vector (ADDR_WIDTH-1 downto 0);
                we      : IN Std_Logic;
85              inclock : IN Std_Logic;
                outclock: IN Std_Logic;
                q       : OUT Std_Logic_Vector (DATA_WIDTH - 1 downto 0));
        END lpmram;

90      architecture ALTERAFit of lpmram is

        Begin
           inst_1: lpm_ram_dq   -- This is the KEY to lpm functions.
              Generic Map(  lpm_widthad =>  ADDR_WIDTH,
95                          lpm_width =>    DATA_WIDTH  )
              PORT Map( data => data,
                        address => address,
                        we => we,
                        inclock => inclock,
100                     outclock => outclock,
                        q => q
                     );

        END ALTERAFit;
105
        --------------------------------------------------------------------------------
        --   The component "lpm_ram_dq" in the package "lpm_component" describes an
        --  interface between VHDL and AHDL/TDF descriptions as listed below.
        --------------------------------------------------------------------------------
110     --
        -- component LPM_RAM_DQ
        --         generic (LPM_WIDTH: positive;
        --                  LPM_TYPE: string := L_RAM_DQ;
        --                  LPM_WIDTHAD: positive;
115     --                  LPM_NUMWORDS: string := UNUSED;
        --                  LPM_FILE: string := UNUSED;
        --                  LPM_INDATA: string := REGISTERED;
        --                  LPM_ADDRESS_CONTROL: string := REGISTERED;
        --                  LPM_OUTDATA: string := REGISTERED;
120     --                  LPM_HINT : string := UNUSED);
        --         port (DATA: in STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0);
        --               ADDRESS: in STD_LOGIC_VECTOR(LPM_WIDTHAD-1 downto 0);
        --               WE: in STD_LOGIC := '1';
        --               INCLOCK: in STD_LOGIC := '1';
125     --               OUTCLOCK: in STD_LOGIC := '1';
        --               Q: out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
        -- end component;
        --
```

A
B
C
D
1
2
3
4
5
6
address(7:0)
ix76
IN
OUT
INBUF
ix73
ix79
ix78
ix77
ix72
ix75
ix74
ix61
ix64
ix63
ix62
ix56
ix60
ix57
ix59
ix58
ix81
ix82
ix80
inclock
outclock
we
data(15:0)
inst_1
address(7:0)
data(15:0)
inclock
outclock
we
q(15:0)
data_int
q_dup0
lpm_ram_dq_16_LPM_RAM_DQ_8_UNUSED_UNUSED_REGISTERED_REGISTERED_REGISTERED_UNUSED
lpmram_16_8
lpm_ram.vhd
Systems Workshop Inc.
4Aug98 14:15
SHEET 1 OF 2



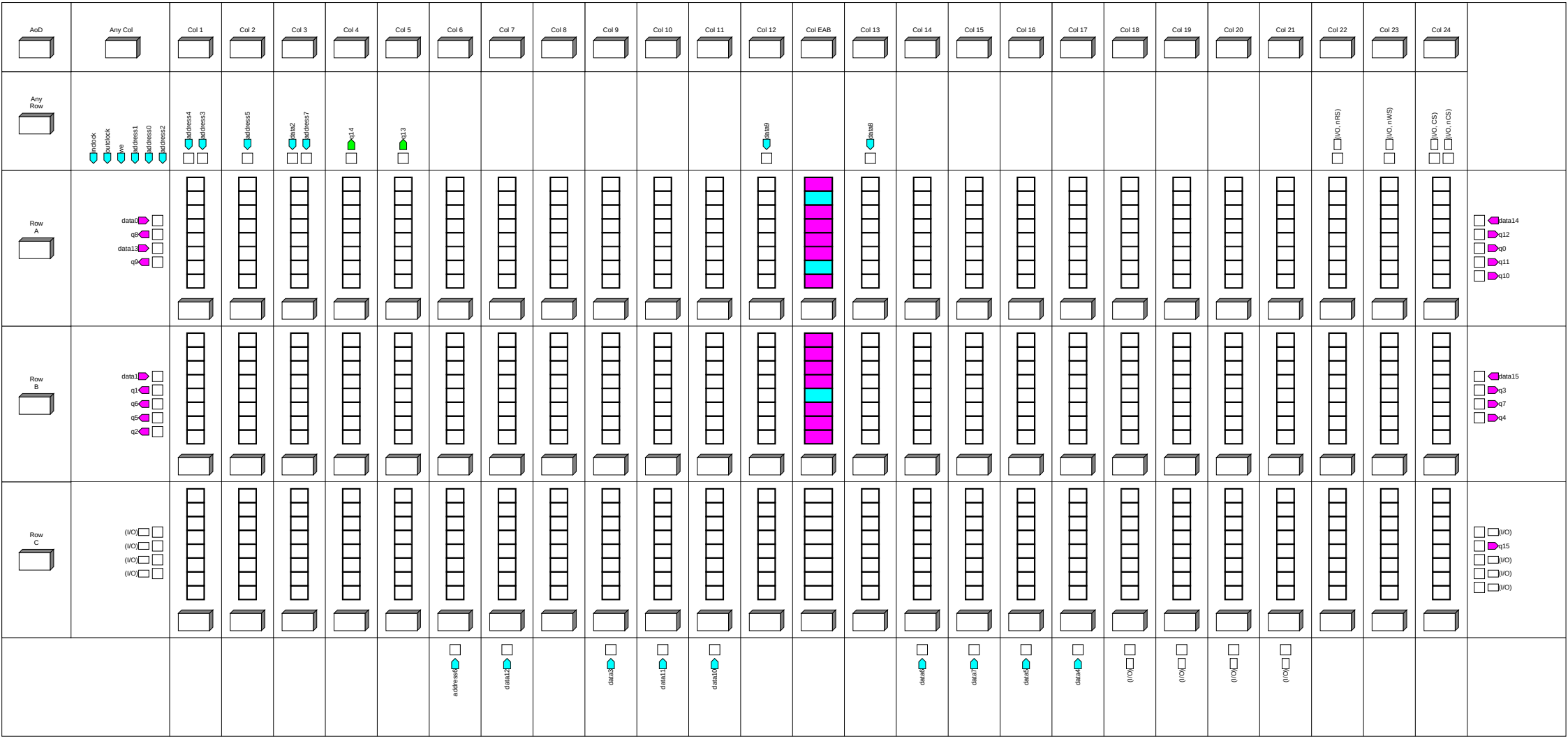




lpmram_16_8	
lpm_ram.vhd	
Systems Workshop Inc.	
4Aug98 14:15	SHEET 2 OF 2

simple ram

```
--------------------------------------------------------------------------
--  VHDL-3
-- ------------------------------------------------------------------------
--      Mem.VHD             1998/8/2 Systems Workshop Inc.
-- ------------------------------------------------------------------------

-------------------------------------------------------------------------------
--  TARGET DESIGN (A Memory MODEL have to be described as an array of latch)
-------------------------------------------------------------------------------
use std.standard.all;

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

Entity Memory is
    Generic(Adr_Bits    : integer:= 8 ;
            Data_Bits   : integer:= 16 );
    Port(   AdrIn   : in  Std_Logic_vector(Adr_Bits-1  downto 0);
            Din     : in  Std_Logic_vector(Data_Bits-1 downto 0);
            Dro     : out Std_Logic_vector(Data_Bits-1 downto 0);
            CS      : in  Std_Logic;
            nWrite  : in  Std_Logic     );

    subType word is Std_Logic_vector(Data_Bits-1 downto 0);
    constant nwords :integer :=2 **Adr_Bits;
    Type Ram_Type  is array( 0 to nwords-1 ) of word;
End;

-------------------------------------------------------------------------------
Architecture RTL of Memory is
-------------------------------------------------------------------------------
BEGIN
    ---------------------------------------------------------------------------
    RamProc:Process ( CS, AdrIn, nWrite, Din )  -- Array of LATCH for MEM/EAB.
    ---------------------------------------------------------------------------
    variable Ram    : Ram_Type;     -- Ram definition.
    variable Address: integer;
    Begin
        Address := conv_integer(AdrIn);

        if( CS='1') then
            Dro <= Ram(Address);
            if( nWrite ='0') then   Ram(Address) := Din;
            end if;
        else
            Zout: For I in 0 to Data_Bits-1 Loop
            Dro(I) <= 'Z';
            end Loop Zout;
        end if;
    End Process RamProc;
    ---------------------------------------------------------------------------


End RTL;
-------------------------------------------------------------------------------
```


AoD
Any Col
Col 1
Col 2
Col 3
Col 4
Col 5
Col 6
Col 7
Col 8
Col 9
Col 10
Col 11
Col 12
Col EAB
Col 13
Col 14
Col 15
Col 16
Col 17
Col 18
Col 19
Col 20
Col 21
Col 22
Col 23
Col 24
Any Row
inclock
inclock
we
address1
address0
address2
address4
address3
address5
data2
address7
q14
q13
data9
data8
(I/O, nRS)
(I/O, nWS)
(I/O, CS)
(I/O, nCS)
Row A
data0
q6
data13
q9
data14
q12
q0
q11
q10
Row B
data1
q1
q6
q5
q2
data15
q3
q7
q4
Row C
(I/O)
(I/O)
(I/O)
(I/O)
(I/O)
q15
(I/O)
(I/O)
(I/O)
address6
data12
data3
data11
data10
data6
data7
data5
data4
(I/O)
(I/O)
(I/O)
(I/O)