APPLICATION NOTE

XAPP 306
XPLA Designer™ hierarchical PHDL design support

Author: Reno L. Sanchez

1998 Jun 22

Philips Semiconductors

PHILIPS

XPLA Designer™ hierarchical PHDL design support XAPP 306

Author: Reno L. Sanchez

INTRODUCTION

Hierarchical designs are designs which contain multiple levels of circuit descriptions. The upper-level design file usually contains a description of all functional blocks of the design and how these functional blocks are interconnected. The lower-level design file(s) usually contain the circuit details of each of the functional block(s) of the design. Hierarchical designs are very useful especially in two areas:

- Parsing very large designs into smaller more manageable functional blocks.
- When a design file uses the same functionality in multiple locations of the design.

The XPLA Designer™ supports hierarchical PHDL (Philips Hardware Description Language) designs. This application note provides the necessary information on how to generate hierarchical designs using hierarchical design files. This document first gives the required hierarchy syntax and then gives two design examples using this syntax.

After reading these hierarchical guidelines, you should be well on your way to implementing designs using the hierarchical feature supported by XPLA Designer™ and the PHD Language. This will enable you to take advantage of all the great features the Philips CoolRunner CPLDs offer.

Terminology

CoolRunner	Name of Philips first CPLD family
CPLD	Complex Programmable Logic Device
PHD	Philips Hardware Description
PHDL	Philips Hardware Description Language
PZ5032	Philips 5 Volt CoolRunner 32 Macrocell CPLD

HIERARCHY SYNTAX

Hierarchy declaration is supported in PHDL. An upper-level PHDL module can refer to a lower-level PHDL module. Modules must be defined in files whose names should be identical to the module name with the ".phd" extension. All files for the same design must be contained in the same directory.

To instantiate a PHDL module, some statements have to be written in the following steps:

In the upper-level source:

- Declare the lower-level module with a higher-level **Interface** declaration.
- Instantiate the module with **Functional_block** declarations.
- Specify port mapping in the equations section.

In the lower-level source:

- Identify lower-level I/O ports with a lower-level Interface statement.

A default value for each port can be specified in the interface declaration. However, *only the value specified in the top-level module will be used.* Any other default value in sub-modules will be ignored.

Port Mappings

There are three kinds of port mappings used in the equation section:

```
● instance_name.port_name     =  signal_name;
● signal_name                 =  instance_name.port_name;
● instance1_name.port_name    =  instance2_name.port_name;
```

Port_name can also be replaced by a group of port_names between [and];

```
● instance_name.[port1, port2, ...] = [signal_1, signal_2, ...];
```

However, dot extensions **cannot** be used with port signals and they **cannot** be used in the interface declaration. Port mappings **cannot** be used within an expression as an operand.

A work around for this condition is given in the design example shown in Appendix B. As seen in the equation below, all *.ackn signals are generated by lower-level design files and passed to the upper-design file, CPU_CTL, where these signals are "ORed" together to form the final !ackn signals.

Desired Equation:

```
!ackn = !bus_brm_gen.g_ackn
    # !bus_frm_gen.g_ackn
    # !bus_sif_gen.g_ackn
    # !bus_sio_gen.g_ackn
    # !bus_fld_gen.g_ackn
    # !bus_etc_gen.g_ackn;
```

In order to work around the condition where port mappings cannot be used within an expression as an operand, the upper-level design file, CPU_CTL, uses temporary nodes to create the equation for !ackn as shown below.

PHDL Equation:

```
    !ackn     =  !t_node1 # !t_node2 # !t_node3 # !t_node4 #
                 !t_node5 # !t_node6;
```

where:

```
    t_node1  =  bus_brm_gen.g_ackn;
    t_node2  =  bus_frm_gen.g_ackn;
    t_node3  =  bus_sif_gen.g_ackn;
    t_node4  =  bus_sio_gen.g_ackn;
    t_node5  =  bus_fld_gen.g_ackn;
    t_node6  =  bus_etc_gen.g_ackn;
```

Please note that the compiler will re-write the PHDL equation to match the desired equation (i.e. t_node1 through t_node6 will be eliminated if the node collapse feature is enabled). The compiler simply needs these temporary nodes to work around the restriction of not being able to have port mapping in the equation.

HIERARCHICAL DESIGN EXAMPLES

The following subsections contain hierarchical design examples.

Hierarchical Design Example: Two-bit Adder

Appendix A contains a hierarchical design example of a simple two-bit adder. The upper-level design file, adder2.phd, adds two bits by calling a one-bit adder contained in a lower-level design file, adder1.phd. This design example is given to illustrate the PHDL hierarchical syntax and give the reader an easy to understand hierarchical design example.

Hierarchical Design Example: CPU Control Circuitry

Appendix B contains a hierarchical design example of a complex CPU controller circuit. This design has been compiled and fit into a PZ5032 device using the XPLA Designer. This design example is given to illustrate more advanced PHDL hierarchical syntax and usage. Table 1 contains all 9 PHD files which make up the design. When examining this design example, please note how parameters are passed between the upper-level and lower-level design modules and in-between the lower-level design modules.

Table 1. Design Example PHDL Files

Design File	Hierarchy	Description
cpu_ctl	Upper Level	Top level PHDL files which calls all lower-level routines
addrsdec	Lower Level	Address Decode Logic
boot_gen	Lower Level	Boot ROM Single/Throttled Quad Read Cycle Generator
nda_gen	Lower Level	No Device Area Single Read/Write Cycle Generator
load_gen	Lower Level	Flash Loader Area Single Read/Write Cycle Generator
fmem_gen	Lower Level	Flash Memory Single/Throttled Quad Read Cycle Generator
sio1_gen	Lower Level	Serial I/O Single Read/Write Cycle GeneratoR – 1 wait state
sio3_gen	Lower Level	Serial I/O Single Read/Write Cycle Generator – 3 wait states
stat_gen	Lower Level	ROMCONT Status Code Generator

CLOSING

If you wish to learn more about PHDL, please refer to the XPLA Designer Users Manual. If you have any questions, please contact the Philips CoolRunner Applications Hotline by dialing toll free 1–888–COOLPLD or 1–505–858–2996.

APPENDIX A: HIERARCHY DESIGN – TWO-BIT ADDER

File: adder2.phd – Upper-Level Design File (Two-bit Adder)

```
Module           adder2;
Title            'Two-bit Adder'
Declarations
  a0, a1, b0, b1, cin, cout, s0, s1 pin;
  adder1 interface (a, b, cin –> sum, cout);
  adder1_0 functional_block adder1;
  adder1_1 functional_block adder1;
Equations
  adder1_0.a   =  a0;
  adder1_0.b   =  b0;
  adder1_0.cin =  cin;
  s0           =  adder1_0.sum;
  adder1_1.cin =  adder1_0.cout;
  adder1_1.a   =  a1;
  adder1_1.b   =  b1;
  s1           =  adder1_1.sum;
  cout         =  adder1_1.cout;
end
```

File: adder1.phd – Lower-Level Design File (One-bit Adder)

```
Module           adder1
Title            'One-bit Adder'
Declarations
  a, b, cin, cout, sum  pin;
Equations
  sum          =  cin & !a & !b
                  # !cin & !a & b
                  # cin & a & b
                  # !cin & a & !b;
  cout         =  cin & b
                  # cin & a
                  # a & b;
end
```

APPENDIX B: HIERARCHY DESIGN – CPU CONTROLLER

File: cpu_ctl.phd– Upper-Level Design File

```
Module cpu_ctl

Title ‘CPU Controller’

/*
************************************************
*                                              *
*   File Name       – cpu_ctl.phd              *
*   Function        – CPU Controller Circuit   *
*                                       *
************************************************
*/

Declarations
     ad27..ad21, rdn, wrn, burstn       pin;
     dataenn, sysclk0n, busgntn, resetn pin;
     brom_csn, brom_oen    pin  istype ’com’;
     flsh_csn, flsh_oen    pin  istype ’com’;
     flsh_wen              pin  istype ’reg_d’;
     sio_cs1n, sio_cs2n    pin  istype ’com’;
     sio_cs3n, sio_cs4n    pin  istype ’com’;
     sio_oen               pin  istype ’com’;
     sio_wen               pin  istype ’reg_d’;
     sif_csn, sif_oen      pin  istype ’com’;
     sif_wen               pin  istype ’reg_d’;
     ackn, rdcenn          pin  istype ’com’;
     g_sts3..g_sts0        node istype ’reg’;
     t_node0               node istype ’com,keep’;                        ”temporary nodes
     t_node12..t_node1     node istype ’com’;                             ”temporary nodes

”---------------------------------------”
”    Address Generation                 ”
”---------------------------------------”
     addrsdec interface (addrH3..addrH0, addrL2..addrL0, g_busgntn –>
                  fld_decn, sif_decn, sio_decn, sio_dec1n, sio_dec2n, sio_dec3n,
                  sio_dec4n, frm_decn, brm_decn, nrw_decn);
     addr_dec_gen functional_block addrsdec;

”---------------------------------------”
”    4bits Status Counter               ”
”---------------------------------------”
     stat_gen interface (g_clrn, g_clk –> g_sts3..g_sts0);
     bus_statusC functional_block stat_gen;

”---------------------------------------”
”    SIO  Bus Cycle Gen. (1wait)”
”---------------------------------------”
     sio1_gen interface (g_decn, g_clk, g_rdn, g_wrn, g_resetn, g_dataenn, g_c3..g_c0 –>
                  g_csn, g_oen, g_wen, g_ackn, g_rdcenn);
     bus_sif_gen functional_block sio1_gen;
```

```
"---------------------------------------"
"    SIO  Bus Cycle Gen. (3wait)"
"---------------------------------------"
     sio3_gen interface (g_decn, g_dec1n, g_dec2n, g_dec3n, g_dec4n, g_clk, g_rdn,
                    g_wrn, g_resetn, g_dataenn, g_c3..g_c0 ->
                    g_cs1n, g_cs2n, g_cs3n, g_cs4n, g_oen, g_wen, g_ackn, g_rdcenn);
     bus_sio_gen functional_block sio3_gen;

"---------------------------------------"
"    Flash Bus Cycle Gen. (3wait)      "
"---------------------------------------"
     fmem_gen interface (g_decn, g_clk, g_rdn, g_wrn, g_resetn, g_burstn, g_dataenn,
                    g_c3..g_c0 -> g_csn, g_oen, g_wen, g_ackn, g_rdcenn);
     bus_frm_gen functional_block fmem_gen;

"---------------------------------------"
"    Boot Bus Cycle Gen. (3wait)"
"---------------------------------------"
     boot_gen interface (g_decn, g_rdn, g_resetn, g_burstn, g_dataenn, g_c3..g_c0 ->
                    g_csn, g_oen, g_ackn, g_rdcenn);
     bus_brm_gen functional_block boot_gen;

"---------------------------------------"
"    Fload Bus Cycle Gen. (3wait)      "
"---------------------------------------"
     fld_gen interface (g_decn, g_dataenn, g_c3..g_c0 -> g_ackn, g_rdcenn);
     bus_fld_gen functional_block fld_gen;

"---------------------------------------"
"    No Device Cycle Gen. (1wait)      "
"---------------------------------------"
     nda_gen interface (g_decn, g_dataenn, g_c3..g_c0 -> g_ackn, g_rdcenn);
     bus_etc_gen functional_block nda_gen;
```

```
Equations
"
"    Set Input Address(addr) of Address Generator
"
     addr_dec_gen.addrH3      =  ad27;
     addr_dec_gen.addrH2      =  ad26;
     addr_dec_gen.addrH1      =  ad25;
     addr_dec_gen.addrH0      =  ad24;

     addr_dec_gen.addrL2      =  ad23;
     addr_dec_gen.addrL1      =  ad22;
     addr_dec_gen.addrL0      =  ad21;

     addr_dec_gen.g_busgntn   =  busgntn;

"
"    Set 4bits Status Counter
"
     !t_node0                 =  !resetn # (rdn & wrn);
     bus_statusC.g_clrn       =  t_node0;
     bus_statusC.g_clk        =  sysclk0n;

"
"       Create Control Signal(CSn,OEn,WEn) of Serial I/F
"
        Setup Signals

     bus_sif_gen.g_decn       =  addr_dec_gen.sif_decn;
     bus_sif_gen.g_clk        =  sysclk0n;
     bus_sif_gen.g_rdn        =  rdn;
     bus_sif_gen.g_wrn        =  wrn;
     bus_sif_gen.g_resetn     =  resetn;
     bus_sif_gen.g_dataenn    =  dataenn;

     bus_sif_gen.g_c0         =  bus_statusC.g_sts0;
     bus_sif_gen.g_c1         =  bus_statusC.g_sts1;
     bus_sif_gen.g_c2         =  bus_statusC.g_sts2;
     bus_sif_gen.g_c3         =  bus_statusC.g_sts3;

     sif_csn                  =  bus_sif_gen.g_csn;                        " CSn
     sif_oen                  =  bus_sif_gen.g_oen;                        " OEn
     sif_wen                  =  bus_sif_gen.g_wen;                        " WEn

"
"       Create Control Signal(CSn,OEn,WEn) of Serial I/O
"
"       Setup Signals

     bus_sio_gen.g_decn       =  addr_dec_gen.sio_decn;
     bus_sio_gen.g_dec1n      =  addr_dec_gen.sio_dec1n;
     bus_sio_gen.g_dec2n      =  addr_dec_gen.sio_dec2n;
     bus_sio_gen.g_dec3n      =  addr_dec_gen.sio_dec3n;
     bus_sio_gen.g_dec4n      =  addr_dec_gen.sio_dec4n;

     bus_sio_gen.g_clk        =  sysclk0n;
     bus_sio_gen.g_rdn        =  rdn;
     bus_sio_gen.g_wrn        =  wrn;
     bus_sio_gen.g_resetn     =  resetn;
     bus_sio_gen.g_dataenn    =  dataenn;

     bus_sio_gen.g_c0         =  bus_statusC.g_sts0;
     bus_sio_gen.g_c1         =  bus_statusC.g_sts1;
     bus_sio_gen.g_c2         =  bus_statusC.g_sts2;
     bus_sio_gen.g_c3         =  bus_statusC.g_sts3;

     sio_cs1n              =  bus_sio_gen.g_cs1n;                        ″ CSn
     sio_cs2n              =  bus_sio_gen.g_cs2n;
     sio_cs3n              =  bus_sio_gen.g_cs3n;
     sio_cs4n              =  bus_sio_gen.g_cs4n;
     sio_oen               =  bus_sio_gen.g_oen;                         ″ OEn
     sio_wen               =  bus_sio_gen.g_wen;                         ″ WEn

″
″       Create Control Signal(CSn,OEn) of Flash(Apli) ROM
″
″       Setup Signals

     bus_frm_gen.g_decn    =  addr_dec_gen.frm_decn;
     bus_frm_gen.g_clk     =  sysclk0n;
     bus_frm_gen.g_rdn     =  rdn;
     bus_frm_gen.g_wrn     =  wrn;
     bus_frm_gen.g_resetn  =  resetn;
     bus_frm_gen.g_burstn  =  burstn;
     bus_frm_gen.g_dataenn =  dataenn;
     bus_frm_gen.g_c0      =  bus_statusC.g_sts0;
     bus_frm_gen.g_c1      =  bus_statusC.g_sts1;
     bus_frm_gen.g_c2      =  bus_statusC.g_sts2;
     bus_frm_gen.g_c3      =  bus_statusC.g_sts3;

     flsh_csn              =  bus_frm_gen.g_csn;                         ″ CSn
     flsh_oen              =  bus_frm_gen.g_oen;                         ″ OEn
     flsh_wen              =  bus_frm_gen.g_wen;                         ″ WEn

″
″       Create Control Signal(CSn,OEn) of Boot ROM
″
″       Setup Signals

     bus_brm_gen.g_decn       =  addr_dec_gen.brm_decn;
     bus_brm_gen.g_rdn     =  rdn;
     bus_brm_gen.g_resetn  =  resetn;
     bus_brm_gen.g_burstn  =  burstn;
     bus_brm_gen.g_dataenn =  dataenn;
     bus_brm_gen.g_c0      =  bus_statusC.g_sts0;
     bus_brm_gen.g_c1      =  bus_statusC.g_sts1;
     bus_brm_gen.g_c2      =  bus_statusC.g_sts2;
     bus_brm_gen.g_c3      =  bus_statusC.g_sts3;
     brom_csn              =  bus_brm_gen.g_csn;                         ″ CSn
     brom_oen              =  bus_brm_gen.g_oen;                         ″ OEn
```

```
"
"       Create Control Signal of Flash Loader Area
"
"       Setup Signals

     bus_fld_gen.g_decn     =  addr_dec_gen.fld_decn;
     bus_fld_gen.g_dataenn  =  dataenn;
     bus_fld_gen.g_c0       =  bus_statusC.g_sts0;
     bus_fld_gen.g_c1       =  bus_statusC.g_sts1;
     bus_fld_gen.g_c2       =  bus_statusC.g_sts2;
     bus_fld_gen.g_c3       =  bus_statusC.g_sts3;

"
"       Create Control Signal(CSn) of No Device Area
"
"       Setup Signals

     bus_etc_gen.g_decn     =  addr_dec_gen.nrw_decn;
     bus_etc_gen.g_dataenn  =  dataenn;
     bus_etc_gen.g_c0       =  bus_statusC.g_sts0;
     bus_etc_gen.g_c1       =  bus_statusC.g_sts1;
     bus_etc_gen.g_c2       =  bus_statusC.g_sts2;
     bus_etc_gen.g_c3       =  bus_statusC.g_sts3;

"
```

```
”       Create Control Signal(ACKn,RDCENn)
”
”
”       ACKn – I had to re-write this using temporary nodes
”
”    !ackn                 =  !bus_brm_gen.g_ackn # !bus_frm_gen.g_ackn # !bus_sif_gen.g_ackn
”                             # !bus_sio_gen.g_ackn # !bus_fld_gen.g_ackn
”                             # !bus_etc_gen.g_ackn;

       t_node1             =  bus_brm_gen.g_ackn;
       t_node2             =  bus_frm_gen.g_ackn;
       t_node3             =  bus_sif_gen.g_ackn;
       t_node4             =  bus_sio_gen.g_ackn;
       t_node5             =  bus_fld_gen.g_ackn;
       t_node6             =  bus_etc_gen.g_ackn;

     !ackn                 =  !t_node1 # !t_node2 # !t_node3 # !t_node4 # !t_node5 # !t_node6;

”
”       RDCENn – I had to re-write this using temporary nodes
”
”    !rdcenn               =  !bus_brm_gen.g_rdcenn # !bus_frm_gen.g_rdcenn # !bus_sif_gen.g_rdcenn
”                             #  !bus_sio_gen.g_rdcenn # !bus_fld_gen.g_rdcenn
”                             #  !bus_etc_gen.g_rdcenn;

       t_node7             =  bus_brm_gen.g_rdcenn;
       t_node8             =  bus_frm_gen.g_rdcenn;
       t_node9             =  bus_sif_gen.g_rdcenn;
       t_node10            =  bus_sio_gen.g_rdcenn;
       t_node11            =  bus_fld_gen.g_rdcenn;
       t_node12            =  bus_etc_gen.g_rdcenn;

     !rdcenn               =  !t_node7 # !t_node8 # !t_node9 # !t_node10 # !t_node11 # !t_node12;

End
```

File: addrsdec.phd – Lower-Level Design File

```
Module addrsdec

Title ‘Address Decode Circuitry’

/*
****************************************************************************
*                                                                          *
*       File Name    – addrsdec.phd              *
*       Function     – Address Decode     Circuitry                        *
*                                                                          *
 **************************************************************************

     CPU Memory MAP
     Flash Loader     :    0x?180 0000 – 0x?1ff ffff (withCentronix)
     SIF              :    0x?980 0000 – 0x?9ff ffff
     SIO CN1          :    0x?b00 0000 – 0x?b1f ffff
     SIO CN2(ISR)     :    0x?b20 0000 – 0x?b3f ffff
     SIO CN3          :    0x?b40 0000 – 0x?b5f ffff
     SIO CN4          :    0x?b60 0000 – 0x?b7f ffff
     Flash ROM (Appli):    0x?f80 0000 – 0x?fbf ffff
     Boot ROM         :    0x?fc0 0000 – 0x?fff ffff
     No Device        :    0x?0c0 0000 – 0x?0ff ffff (Read/Write NG)
                           0x?500 0000 – 0x?8ff ffff (Read/Write NG)
                           0x?e00 0000 – 0x?f7f ffff (Read/Write NG)

*/
Declarations
     addrH3..addrH0, addrL2..addrL0, g_busgntn pin;
     fld_decn, sif_decn, sio_decn              pin istype ’com’;
     sio_dec1n, sio_dec2n, sio_dec3n, sio_dec4n pin istype ’com’;
     frm_decn, brm_decn, nrw_decn              pin istype ’com’;

     addrH          =  [addrH3..addrH0];
     addrL          =  [addrL2..addrL0];

Equations
     !fld_decn      =  g_busgntn & (addrH == ^b0001) & addrL2;
     !sif_decn      =  g_busgntn & (addrH == ^b1001) & addrL2;
     !sio_decn      =  g_busgntn & (addrH == ^b1011) & !addrL2;
     !sio_dec1n     =  g_busgntn & (addrH == ^b1011) & (addrL == ^b000);
     !sio_dec2n     =  g_busgntn & (addrH == ^b1011) & (addrL == ^b001);
     !sio_dec3n     =  g_busgntn & (addrH == ^b1011) & (addrL == ^b010);
     !sio_dec4n     =  g_busgntn & (addrH == ^b1011) & (addrL == ^b011);
     !frm_decn      =  g_busgntn & (addrH == ^b1111) & addrL2 & !addrL1;
     !brm_decn      =  g_busgntn & (addrH == ^b1111) & addrL2 & addrL1;
     !nrw_decn      =  (((addrH == ^b0000) & addrL2 & addrL1)
                       # ((addrH >= ^b0101) & (addrH <= ^b1000))
                       # (addrH == ^b1110)
                       # ((addrH == ^b1111) & !addrL2));
End
```

File: boot_gen.phd – Lower-Level Design File

```
Module boot_gen

Title ’ BOOT ROM Single/Throttled Quad Read Cycle Generator’
/*
****************************************************************************
*                                                                          *
*    File Name      – boot_gen.phd                                         *
*    Function       – Boot ROM Single/Throttled Quad Read Cycle Generator  *
*                                                                          *
****************************************************************************
*/

Declarations

     g_decn, g_rdn, g_resetn            pin;
     g_burstn, g_dataenn, g_c3..g_c0    pin;
     g_csn, g_oen, g_ackn, g_rdcenn     pin istype ’com’;
     g_c            = [g_c3..g_c0];

Equations

/*   This Gen. supports between single read and throttled quad read.
     This Gen. supports that Tacc(Access Time) is 100ns + Driver.
       Single bus(read/write) cycle span is 4 cycles (3wait).
       Throttled quad read cycle span is 15 cycles (3wait x 4).
*/

”    Create Control Signal(CSn,OEn)

     !g_csn         =  g_resetn & !g_rdn;
     !g_oen         =  !g_decn & !g_rdn & !g_dataenn;

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn        =  !g_decn & ((g_burstn & (g_c == ^b0011)) # (!g_burstn & (g_c == ^b1100)));
     !g_rdcenn      =  !g_decn & !g_dataenn & ((g_c == ^b0011)
                            # (!g_burstn & ((g_c == ^b0111) # (g_c == ^b1011) # (g_c == ^b1111))));
End
```

File: nda_gen.phd – Lower-Level Design File

```
Module nda_gen

Title ’ No Device Area Single Read/Write Cycle Generator’

/*
***************************************************************************
*                                                                         *
*    File Name      – nda_gen.phd                                         *
*    Function       – No Device Area Single Read/Write Cycle Generator    *
*                                                                         *
****************************************************************************
*/

Declarations

     g_decn                   pin;
     g_dataenn, g_c3..g_c0    pin;
     g_ackn, g_rdcenn         pin istype ’com’;

     g_c = [g_c3..g_c0];

Equations
/*
     This Gen. supports single read and single write bus transaction.
        Single read cycle span is 3 cycles (0wait).
        Single write cycle span is 3 cycles (0wait).
*/

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn        =  !g_decn & (g_c == ^b0001);
     !g_rdcenn      =  !g_decn & !g_dataenn & (g_c == ^b0001);

End
```

File: fld_gen.phd – Lower-Level Design File

```
Module fld_gen

Title ’ Flash Loader Area Single Read/Write Cycle Generator’
/*
***************************************************************************
*                                                                         *
*    File Name      – fld_gen.phd                                         *
*    Function       – Flash Loader Area Single Read/Write Cycle Generator  *
*                                                                         *
***************************************************************************
*/

Declarations

     g_decn                   pin;
     g_dataenn, g_c3..g_c0    pin;
     g_ackn, g_rdcenn         pin istype ’com’;

     g_c            =  [g_c3..g_c0];

Equations
/*
”    This Gen. supports single read and single write bus transaction.
”       Single read cycle span is 6 cycles (3wait).
”       Single write cycle span is 6 cycles (3wait).
*/

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn        =  !g_decn & (g_c == ^b0011);
     !g_rdcenn      =  !g_decn & !g_dataenn & (g_c == ^b0011);

End
```

File: fmem_gen.phd – Lower-Level Design File

```
Module fmem_gen

Title ’ Flash Memory Single/Throttled Quad Read Cycle Generator’

/*
****************************************************************************
*                                                                          *
*    File Name – fmem_gen.phd                                              *
*    Function – Flash Memory Single/Throttled Quad Read Cycle Generator    *
*                                                                          *
****************************************************************************
*/

Declarations

     g_decn, g_clk, g_rdn, g_wrn, g_resetn     pin;
     g_burstn, g_dataenn, g_c3..g_c0           pin;
     g_csn, g_oen                              pin istype ’com’;
     g_wen                                     pin istype ’reg_d’;
     g_ackn, g_rdcenn                          pin istype ’com’;

     g_c            =  [g_c3..g_c0];

Equations

/*   This Gen. supports between single read and throttled quad read, and, single write.
     This Gen. supports that Tacc(Access Time) is 120ns.
     Single read bus cycle span is 6 cycles (3wait).
     Throttled quad read cycle span is 19 cycles (3w x 4).
     Single write bus cycle span is 6 cycles (3wait).
*/

”    Create Control Signal(CSn,OEn)

     !g_csn         =  g_resetn & (!g_rdn # !g_wrn);                     ” CSn
     !g_oen         =  !g_decn & !g_rdn & !g_dataenn;                    ” OEn

”    Create Control Signal(WEn)

     g_wen.ap       =  !g_decn & !g_wrn;
     g_wen.clk      =  g_clk;
     !g_wen.d       =  (g_c <= ^b0010);

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn        =  !g_decn & ((g_burstn & (g_c == ^b0011)) # (!g_burstn & (g_c == ^b1100)));
     !g_rdcenn      =  !g_decn & !g_dataenn & ((g_c == ^b0011)
                           # (!g_burstn & ((g_c == ^b0111) # (g_c == ^b1011) # (g_c == ^b1111))));

End
```

File: sio1_gen.phd – Lower-Level Design File

```
Module sio1_gen

Title ’ Serial I/O Single Read/Write Cycle Generator – 1 Wait States’

/*
******************************************************************************
*                                                                            *
*    File Name – sio1_gen.phd                                                *
*    Function – Serial I/O Single Read/Write Cycle Generator – 1 Wait State  *
*                                                                            *
******************************************************************************
*/

Declarations
     g_decn, g_clk, g_rdn, g_wrn       pin;
     g_resetn, g_dataenn, g_c3..g_c0   pin;
     g_csn, g_oen                      pin istype ’com’;
     g_wen                             pin istype ’reg_d’;
     g_ackn, g_rdcenn                  pin istype ’com’;

     g_c            =  [g_c3..g_c0];

Equations
/*   This Gen. supports single read and single write bus transaction.
     This Gen. supports that Tacc(Access Time) is 42.1ns.
     Single read cycle span is 3 cycles (1wait).
     Single write cycle span is 3 cycles (1wait).
*/

”    Create Control Signal(CSn,OEn)

     !g_csn         =  g_resetn & (!g_rdn # !g_wrn);                       ” CSn
     !g_oen         =  !g_decn & !g_rdn & !g_dataenn;                      ” OEn

”    Create Control Signal(WEn)

     g_wen.ap       =  !g_decn & !g_wrn;
     g_wen.clk      =  g_clk;
     !g_wen.d       =  (g_c == ^b0000);                                    ” WEn

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn        =  !g_decn & (g_c == ^b0001);
     !g_rdcenn      =  !g_decn & !g_dataenn & (g_c == ^b0001);

End
```

File: sio3_gen.phd – Lower-Level Design File

```
Module sio3_gen

Title ’ Serial I/O Single Read/Write Cycle Generator – 3 Wait States’
/*
*********************************************************************************
*                                                                               *
*File Name         –  sio3_gen.phd                                              *
*    Function      –  Serial I/O Single Read/Write Cycle Generator –3 Wait States *
*                                                                               *
*********************************************************************************
*/

Declarations
     g_decn, g_dec1n, g_dec2n, g_dec3n, g_dec4n                       pin;
     g_clk, g_rdn                                                     pin;
     g_wrn, g_resetn, g_dataenn, g_c3..g_c0                           pin;
     g_cs1n, g_cs2n, g_cs3n, g_cs4n                                   pin istype ’com’;
     g_oen                                                            pin istype ’com’;
     g_wen                                                            pin istype ’reg_d’;
     g_ackn, g_rdcenn                                                 pin istype ’com’;

     g_c           =  [g_c3..g_c0];

Equations
/*   This Gen. supports single read and single write bus transaction.
     This Gen. supports that Tacc(Access Time) is 95ns.
       Single read cycle span is 5 cycles (3wait).
       Single write cycle span is 5 cycles (3wait).
*/
”    Create Control Signal(CS1n,CS2n,OEn)
     !g_cs1n       =  g_resetn & !g_dec1n & (!g_rdn # !g_wrn);            ” CS1n
     !g_cs2n       =  g_resetn & !g_dec2n & (!g_rdn # !g_wrn);            ” CS2n
     !g_cs3n       =  g_resetn & !g_dec3n & (!g_rdn # !g_wrn);            ” CS3n
     !g_cs4n       =  g_resetn & !g_dec4n & (!g_rdn # !g_wrn);            ” CS4n

     !g_oen        =  !g_decn & !g_rdn & !g_dataenn;                      ” OEn

”    Create Control Signal(WEn)
     g_wen.ap      =  !g_decn & !g_wrn;
     g_wen.clk     =  g_clk;
     !g_wen.d      =  (g_c <= ^b0010);                                    ”WEn

”    Create Control Signal(ACKn,RDCENn)

     !g_ackn       =  !g_decn & (g_c == ^b0011);
     !g_rdcenn     =  !g_decn & !g_dataenn & (g_c == ^b0011);
End
```

File: stat_gen.phd – Lower-Level Design File

```
Module stat_gen

Title ’ROMCONT status code generator’

/*
***************************************************************************
*                                                                         *
*    File Name      – stat_gen.phd                                        *
*    Function       – ROMCONT status code generator                       *
*                                                                         *
***************************************************************************
*/

Declarations
     g_clrn, g_clk          pin;
     g_sts3..g_sts0         pin istype ’reg_d’;

Equations
     !g_sts0.ar     =  !g_clrn;                 ” b0
     g_sts0.clk     =  g_clk;
     g_sts0.d       =  !g_sts0.q;

     !g_sts1.ar     =  !g_clrn;                 ” b1
     g_sts1.clk     =  g_clk;
     g_sts1.d       =  g_sts1.q $ g_sts0.q;

     !g_sts2.ar     =  !g_clrn;                 ” b2
     g_sts2.clk     =  g_clk;
     g_sts2.d       =  g_sts2.q $ (g_sts1.q & g_sts0.q);

     !g_sts3.ar     =  !g_clrn;                 ” b3
     g_sts3.clk     =  g_clk;
     g_sts3.d       =  g_sts3.q $ (g_sts2.q & g_sts1.q & g_sts0.q);
End
```

NOTES

XPLA Designer™ hierarchical PHDL design support AN066

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

print code Date of release: 07-98

Document order number: 9397 750 04149

Let's make things better.

Philips
Semiconductors

PHILIPS