APPLICATION NOTE

Xilinx has acquired the entire Philips CoolRunner Low Power CPLD Product Family.
For more technical or sales information, please see: www.xilinx.com

XAPP 303

Altera (AHDL) to Philips (PHDL) design conversion guidelines

Author: Reno L. Sanchez

1998 Jun 26

Philips
Semiconductors

PHILIPS

Altera (AHDL) to Philips (PHDL) design conversion guidelines

XAPP 303

Xilinx has acquired the entire Philips CoolRunner Low Power CPLD Product Family. For more technical or sales information, please see: www.xilinx.com

DOCUMENT SCOPE

This document provides information required to translate an Altera Hardware Description Language (AHDL) based design into a Philips Hardware Description Language (PHDL) based design. Designs which should be targeted for conversions are ones in which the customer system needs require one of Philips CoolRunner CPLD advanced features including: dramatic power savings, increased routability with fixed pins, and higher logic density, etc.

This memorandum first gives the key conversion factors which determine if the conversion is feasible. Next, the structural and language syntax differences between the AHDL and PHDL languages are given. Finally, a design example written in both AHDL and PHDL is given. This document also addresses pin capability issues between Altera's MAX7000 family and the Philips CoolRunner XPLA1 family.

Terminology

AHDL	Altera Hardware Description Language
CPLD	Complex Programmable Logic Device
CR32	CoolRunner 32 Macrocell CPLD
CR32CS	CoolRunner 32 Macrocell ISP and Enhanced Clocking CPLD
CR64	CoolRunner 64 Macrocell CPLD
CR64CS	CoolRunner 64 Macrocell ISP and Enhanced Clocking CPLD
CR128	CoolRunner 128 Macrocell CPLD
CR128CS	CoolRunner 128 Macrocell ISP and Enhanced Clocking CPLD
CR320	CoolRunner 320 Macrocell CPLD
CR960	CoolRunner 960 Macrocell CPLD
DFF	D type flip-flop
DFFE	D-type flip-flop with Clock Enable
JKFF	JK type flip-flop
JKFFE	JK-type flip-flop with Clock Enable
OE	Output Enable
PHDL	Philips Hardware Description Language
SRFFE	SR-type flip-flop with Clock Enable
TFF	T type flip-flop
TFFE	T-type flip-flop with Clock Enable
XPLA1	Philips CoolRunner CPLD family ranging from 32 to 128 Macrocells

KEY CONVERSION FACTORS

This section gives the key conversion factors which must be addressed before the design conversion is attempted. If these key conversion factors are not met, the design conversion has no possibility of success and the designer should not attempt the conversion. In other words, the factors given in this section must be satisfied or the design conversion is not possible (with fixed pins). If these factors are satisfied, then the designer should attempt the design conversion.

Number of Macrocells

First and foremost, one must ensure that the number of macrocells between an Altera CPLD and a Philips CPLD are equivalent. One should attempt to convert a 32 macrocell Altera CPLD (EPM7032) into a 32 macrocell Philips CPLD (PZ5032). In some cases however, it may be possible to fit a larger macrocell Altera CPLD (i.e. EPM7064) into a smaller Philips CPLD (i.e. PZ5032) if the design is logic (product term) constrained and not macrocell constrained.

Clocking

The clocking approach taken is dependent on which CoolRunner family is being used. The XPLA1 enhanced clocking CPLDs (CR32CS, CR64CS, and CR128CS) have more clocking capabilities than the XPLA1 non-enhanced clocking CPLDs (CR32, CR64, and CR128). The XPLA1 non-enhanced clocking CPLDs have either 2 or 4 clock pins, depending on macrocell count (see Table 2). The XPLA1 enhanced clocking CPLDs have the same clocks pins but also contain 2 additional control term clocks per logic block. These control term clocks can be used to make any input a clock resource. Therefore, it is much easier to match the pinout of a corresponding Altera CPLD to a XPLA1 enhanced clocking device than to a XPLA1 non-enhanced clocking device. The main constraint for the XPLA1 enhanced clocking devices to be pin compatibility (in the context of clocking constraints) with the Altera MAX7000 family is the number of available clocks. As long as the maximum number of clocks is not exceeded, the XPLA1 enhanced clocking CPLDs should be pin compatible. Table 1 gives the number of clocks provided by the XPLA1 enhanced clocking CPLDs.

The approach is much different with the XPLA1 non-enhanced clocking CPLDs. Before starting the design conversion, one must ensure that all clocks used in the design conform to the Philips CoolRunner pinout. Table 1 gives the Philips CoolRunner clock pinout for the CR32, CR64, and CR128 CPLDs. Please note that CLK0 is a Synchronous clock (must be driven by an external source) while CLK1, CLK2, and CLK3 can be used as either Synchronous clocks (driven by an external source) or Asynchronous clocks (driven by a macrocell equation).

If the design uses any pin as a clock that is different than the ones specified in Table 1, the design will not be pin compatible with the CoolRunner CPLD. However, as long as the number of clocks in the design does not exceed the number of clocks offered by the specific CoolRunner CPLD, the design will probably still fit. What you lose in this case is pin compatibility with the corresponding Altera MAX7000 CPLD. Your decision on whether or not to proceed with the design conversion depends on whether pin compatibility is important. If pin compatibility is a requirement, then there is no reason to convert the design. However, if pin compatibility is not a requirement, then you should convert the design.

Table 1. XPLA1 Enhanced Clocking Clock Resources

Device	# of Clocks Pins	# of Control Term Clocks	Total # of Clocks
CR32CS	2	4	6
CR64CS	4	8	12
CR128CS	4	16	20
CR3320	8	32	40
CR3960	8	96	104

Table 2. XPLA1 Non-enhanced Clocking Clock Pinouts

Package/Device	Clock Number	Clock Type	44 PLCC	44 TQFP	68 PLCC	84 PLCC	100 PQFP	100 TQFP	128 LQFP	160 PQFP
CR32	CLK0	Sync	43	37						
	CLK1	Sync/Async	4	42						
CR64	CLK0	Sync	43	37	67	83	89			
	CLK1	Sync/Async	24	18	36	44	42			
	CLK2	Sync/Async	21	15	33	41	39			
	CLK3	Sync/Async	4	42	4	4	94			
CR128	CLK0	Sync				83	89	87	114	139
	CLK1	Sync/Async				44	42	40	53	62
	CLK2	Sync/Async				41	39	37	50	59
	CLK3	Sync/Async				4	94	92	119	144

If the number of clocks used in the design exceeds the number of clocks contained within the targeted Philips CPLD, then the design will not fit. It may be possible to convert asynchronous clocks in an Altera design to synchronous clocks by modifying the original design. Of course, any design modifications must be tested to insure functionality.

Reset/Preset/Output Enable

The final restriction is reset/preset/oe functionality. The Philips CoolRunner CPLDs have the ability to provide either 36 AND terms or 36 SUM terms (control terms) for each preset/reset/oe function without using any of the macrocells. For example:

```
signal.ar  = A & B & C & ...   – up to 36 terms
signal.oe = A # B # C # ...    – up to 36 terms
```

However, if a combination of AND and SUM terms are needed to control reset/preset/oe, then a macrocell must be used as an internal node to generate the sum of product signal. For example:

```
node = (A & B) # (C & D);
signal.ar = node;
```

In the Altera MAX7000 family, a macrocell is not needed for a very limited set of combination of AND and SUM terms which control the reset/preset/oe function.

STRUCTURE TRANSLATION

This section describes the structural differences between the AHDL and PHDL languages.

AHDL Structure

An AHDL file is broken into several sections including: a Header section, a Design Section, a Subdesign section, a Variable section, and a Logic section. An example of the AHDL structure is given in Figure 1. Listed below is a brief description of each section:

- The Header section can contain the following items: Title Statement, a Constant Statement, a Function Prototype Statement, an Include Statement, and an Options Statement.
- The optional Design Section specifies pin, buried logic cell, chip, clique, logic option, and device assignments.
- The Subdesign Section declares the input, output, and bi-directional ports of the file.
- The Variable Section is used to declare any variable used in the Logic Sections. These variables include both external and internal logic.
- The Logic Section specifies the logical operations of the design file and is the body of the Subdesign Section.

PHDL Structure

A PHDL file is broken into four distinct sections: the Header, the Declarations, the Logic Description, and the End. An example of the PHDL structure is given in Figure 2. Listed below is a brief description of each section:

- The Header section contains descriptive information about the design. This section must contain a name for the PHDL file and it can contain title and property statements.
- The Declaration section is where constants, variables, signals, and macro functions are declared and initialized. The start of the declaration section is indicated by the reserved word “Declarations” placed by itself on one line and the declarations that follow.
- The Logic section is where the design is defined by establishing relationships between the inputs and outputs created in the Declaration section. The design may be defined using equations, state machines, or truth tables.
- All PHDL files must close with the reserved word “end”.

Key Structural Differences

The key differences between the PHDL and AHDL design file structure is the way pins and outputs of logic functions are defined. In PHDL, if the output of the logic function drives an external pin, the name of function can be the same as the pin name and only needs to be declared once. In AHDL, if the output of the logic function drives an external pin, the name of function must be different from the pin name and the two must be equated in the Logic section. This can be confusing in AHDL because it is not obvious which logic will drive external pins and which are used as buried logic until you examine the entire AHDL file. In PHDL, all node and pin declarations

are made in the Declaration section near the front of the PHDL file where the user can easily distinguish between logic which drives external pins and logic which drives internal nodes.

LANGUAGE TRANSLATION

The following subsections give the PHDL equivalent for keywords, operators, and ports for primary inputs. These subsections can serve as a quick reference as you begin creating designs with PHDL.

Keyword cross-reference

Table 3 lists AHDL keywords in the first column and gives the PHDL equivalents in the second column.

Table 3. AHDL – PHDL Keyword Cross Reference

AHDL Keyword	PHDL Equivalent or Cross Reference
CASE	CASE
DEVICE IS	DEVICE
ELSE, ELSIF, and END IF	ELSE
END CASE	ENDCASE
AHDL State Machine section	ENDWITH
AHDL Logic Section	EQUATIONS
AHDL Options Statement	FLAG
None	FUSES
AHDL State Machine Section	GOTO
IF	IF
AHDL Subdesign Section	IN
AHDL Primitives	ISTYPE
AHDL Function Prototype Statement	LIBRARY
AHDL Function Prototype Statement	MACRO
DESIGN IS	MODULE
NODE	NODE
@	PIN
WITH STATES	STATE
MACHINE OF BITS	STATE_DIAGRAM
THEN	THEN
TITLE	TITLE
TABLE	TRUTH_TABLE
WHEN	WHEN
AHDL State Machine Section	WITH

Operator Equivalents

Table 4 shows AHDL operators and their PHDL equivalents. Each operator's priority is listed in parentheses beside the symbol for both AHDL and PHDL operators. The operators are similar in AHDL and PHDL.

Table 4. AHDL – PHDL Operator Equivalents

AHDL	PHDL	Operation
– (1)	– (1)	negation
! (1)	! (1)	NOT (invert)
+ (2)	+ (3)	arithmetic addition
– (2)	– (3)	arithmetic subtraction
= = (3)	= = (4)	equal to
! = (3)	! = (4)	not equal to
< (3)	< (4)	less than
< = (3)	< = (4)	less than or equal to
> (3)	> (4)	greater than
> = (3)	> = (4)	greater than or equal to
& (4)	& (2)	AND
!& (4)	none	NAND (invert AND)
$ (5)	$ (3)	XOR (exclusive OR)
!$ (5)	!$ (3)	XNOR (exclusive NOR)
# (6)	# (6)	OR
! # (6)	none	NOR (invert OR)

Dot Extensions

In both AHDL and PHDL, dot extensions are used to connect the features of the macrocell. The ports of an instance of a function are declared in the following format:

<macrocell>.<dot extension>

Table 5 shows the AHDL and PHDL dot extension notations for connections to macrocell logic.

Table 5. AHDL – PHDL Dot Extensions

AHDL	PHDL	Function
.d	.D	D input to D flip-flop
.t	.T	T input to T flip-flop
.q	.Q	Register feedback
.j	.J	J input to JK flip-flop
.k	.K	K input to JK flip-flop
.s	.S	S input to SR flip-flop
.r	.R	R input to SR flip-flop
.clk	.C	Clock to flip-flop
.prn	.AP	Preset
.clrn	.AR	Clear
TRI	.OE	Tri-state buffer

AHDL has active-low Preset and Clear signals to all flip-flop types: DFF, DFFE, TFF, TFFE, JKFF, JKFFE, and SRFFE. You must explicitly use the TRI primitive when you create a tri-state output. If no port is explicitly used in AHDL and PHDL, the default port on the left-hand side of an equation is the primary data input to the instance of the primitive; the default port on the right-hand side of the equation is the primary output. JK and SR flip-flops always require an explicit port for all inputs.

PITFALLS

This section describes potential pitfalls (language differences) when converting between AHDL and PHDL.

Logic Synthesis

Both PHDL and AHDL do not need to represent a physical “polarity control” for an output. Both compilers automatically apply De Morgan’s inversion to all functions as part of logic synthesis. These compilers then compute the most appropriate configuration to obtain the logical behavior that has been defined.

Identifiers

Identifiers are case-sensitive in PHDL designs, while AHDL identifiers (symbolic names) are case insensitive.

Groups

You declare a set of signals in PHDL with an identifier followed by square brackets that enclose a comma-separated list of set members. Subsequent references to the set are made using the identifier only. AHDL notation is slightly different. In AHDL, you declare a group of signals with an identifier followed by an empty pair of square brackets.

The following examples show how a group of seven associated D flip-flops are declared and used in AHDL and PHDL.

AHDL Declaration:

```
countq[6..0] : DFF
```

AHDL Reference:

```
countq[] = countq[] & incadr & clrqdr & !carryq
# countq[] & !incadr & clrard
```

PHDL Declaration:

```
countq = [q6..q0];
```

PHDL Reference:

```
countq:= countq & incadr & clradr & !carryq
# countq & !incadr & clradr
```

Equations

All flip-flops must be explicitly specified before being used in AHDL. Equations in AHDL are used only to describe combinatorial logic. The explicit declaration of flip-flops in AHDL is somewhat analogous to PHDL’s ISTYPE declaration and makes registered assignment operators superfluous.

Comments

Comments in AHDL can span multiple lines and a comment must begin and end with a percent character (%). Comments in PHDL begin with either a quotation mark (“) or double slash (//) and continue to the end of the line. Comments in PHDL can span multiple lines with a /* */ enclosing the comments.

Active-Low Specification

The exclamation mark (!) in PHDL is used to declare active-low ports. In AHDL, however, you cannot create active-low ports in the Subdesign Section with the NOT (!) operator. In AHDL, the Logic Section may refer to signals that are either actual device pins or ports that connect to the next higher level of hierarchy. Therefore, the names of ports and their logical sense must agree for the design to compile without errors.

DESIGN CONVERSION EXAMPLE

This section contains a Counter design witch is implemented in AHDL and then re-implemented using PHDL. This should give the reader an idea of how to convert other designs.

AHDL Design Example

Figure 1 gives a Counter implemented in AHDL.

```
TITLE "Arbitrary–length counter with Carry Out";
CONSTANT PENULTIMATE_COUNT = 109

DESIGN IS "count110"
BEGIN
        DEVICE IS EPM5032"
        BEGIN
                clradr @ 28, incadr @ 27, osc @ 16          : INPUT
                a0 @3, a1 @ 4, a2 @ 5, a3 @ 6               :OUTPUT
                a4 @ 9, a5 @ 10, a6 @ 11                    :OUTPUT
                carrya @ 12                                 :OUTPUT
            END;
END;
%-------------------------------------------------------------------------------------------------------------------------------
        This counter uses registered look–ahead carry to implement an arbitrary length count.
        The input to the carryq register will be high at the 109 count. On the next Clock
        with incadr high, the carryq will be set and the count will advance to 110. The
        counter keeps incrementing as long as carryq is low, so the counter will return to 0
        on the next Clock with incadr high.
-------------------------------------------------------------------------------------------------------------------------------%
SUBDESIGN Count110
(
        osc, incadr, clradr                         :INPUT
        a[6..0], carrya                             :OUTPUT
)
VARIABLE
            carryq, count[6..0]                     :DFF

BEGIN
            countq[].clk = osc;
            countq[]  = (count[] + 1) & incadr  & clradr  & !carryq
                    # countq[]  & !incadr  & clradr;
            a[]   = countq[];
            carryq  = (countq[] == PENULTIMATE_COUNT) & incadr & clradr
                    # carryq  & !incadr  & clradr;
            carrya  = carryq;
END;
```

SP00514

Figure 1. AHDL Counter Design

PHDL Design Example

Figure 2 gives a Counter implemented in PHDL.

```
Module Count110
TITLE 'Arbitrary–length counter with Carry Out'

Declarations
        osc, incadr, clradr                 pin 16, 27, 28;
        A0, A1, A2, A3, A4, A5, A6          pin 3, 4, 5, 6, 9, 10, 11  istype 'reg_d';
        carrya                              pin 12     istype 'reg_d';
        counta = [A6, A5, A4, A3, A2, A1, A0];
        PENULTIMATE_COUNT = 109;


  " This counter uses registered look–ahead carry to implement an arbitrary length count.
  " The input to the carryq register will be high at the 109 count. On the next Clock
  " with incadr high, the carryq will be set and the count will advance to 110. The
  " counter keeps incrementing as long as carryq is low, so the counter will return to 0
  " on the next Clock with incadr high.

Equations
      counta.clk = osc;
      counta.d := ((counta.q + 1) & incadr   & clradr  & !carrya
            #   counta  & !incadr  & clradr);

      carrya := (counta == PENULTIMATE_COUNT) & incadr & clradr
           # carrya  & !incadr  & clradr;
END;
```

SP00515

Figure 2. PHDL Counter Design

TECHNICAL SUPPORT

With these guidelines, you should be well on your way to converting designs written in AHDL into design utilizing the PHDL language. This will enable you to take advantage of all the great features the Philips CoolRunner CPLDs offer. If you wish to learn more about PHDL, please refer to the XPLA Designer Users Manual.

This document was authored by Reno Sanchez, Applications and Architecture Development Manager. If you need more information, please contact me at 505–858–2790 or call the Philips CPLD Technical Support Line at 1–888–COOLPLD (1–888–266–5753) or 505–858–2996; or send email to coolpld@scs.philips.com.

Altera (AHDL) to Philips (PHDL) design conversion guidelines

AN057

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

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 04154

Let's make things better.

Philips Semiconductors

PHILIPS