Differences In ABEL and PHDL

XAPP312 (v1.0) October 22, 1999 Application Note

Summary

This document highlights the few major differences between ABEL and PHDL. All other PHDL constructs and syntax not discussed in this document are supported in ABEL. Most PHDL designs will be accepted in Xilinx Project Navigator with just a modification to the file extension.

Introduction

PHDL (formerly Philips Hardware Description Language) is a subset of ABEL and is one of the source file types accepted by XPLA Professional to target a CoolRunner™ CPLD. The Xilinx Project Navigator supports ABEL source files. The purpose of this application note is to document the difference between PHDL and ABEL so that customers can modify their CoolRunner designs for compilation using Xilinx Project Navigator.

Please note that for the majority of customers, the only modification to a PHDL source file required will be changing the file extension.

This document assumes that readers have a working knowledge of PHDL and ABEL and therefore does not attempt to provide detailed information about either language.

File Extensions

ABEL files are denoted by `.ABL` as the extension, PHDL files are denoted with a `.PHD` extension. The file extension of PHDL files must be changed to `.ABL` for compilation in Xilinx Project Navigator.

Dot Extensions

Table 1 shows the dot extensions supported by ABEL and the level of support for these dot extensions in PHDL.

PHDL also supports two additional dot extensions, `.X1` and `.X2`. These dot extensions are used to support the hardware XOR function in CoolRunner XPLA2 (960 and 320 macrocell devices) CPLDs.

XPLA, XPLAE, and XPLA2 families of CoolRunner CPLDs do not architecturally support both asynchronous reset and asynchronous preset of a register. Therefore, use of both `.AR` and `.AP` for a single register is not allowed in designs targeting these families of devices.

Table 1: ABEL Dot Extensions

ABEL Dot Extensions	Function	Directly Supported	Indirectly Supported	Not Supported
.AP, .ASET, .PR	Asynchronous preset	✓		
.AR, .ACLR, .RE	Asynchronous reset	✓		
.CE	Clock enable		✓	
.CLK, .C	Clock input to register	✓		
.CLR, .SR	Synchronous reset	✓		
.COM	Combinatorial feedback	✓		
.D	Data input to D FF	✓		
.FB, .Q	Register feedback	✓		
.FC	Flip-flop mode control			✓
.J	Data input to a JK FF		✓	
.K	Data input to a JK FF		✓	
.LD	Register load input			✓
.LE	Latch enable input			✓
.LH	Latch enable (High)			✓
.PIN	Pin feedback	✓		
.OE	Output enable	✓		
.R	Input to an SR FF		✓	
.S	Input to an SR FF		✓	
.SET, .SP	Synchronous preset	✓		
.T	Input to a T FF	✓		

Directives

Directives are accepted in PHDL but are ignored.

Keywords and Declarations

Most ABEL keywords and declarations are supported in PHDL. Table 2 shows the ABEL keywords and declarations that are either ignored or not supported in PHDL.

Table 2: Keywords and Declarations

ABEL Keywords / Declarations	Accepted In PHDL But Ignored	Not Supported In PHDL
TEST_VECTORS keyword	✓	
TRACE keyword	✓	
FUSES keyword		✓
LIBRARY keyword		✓
STATE declaration		✓
STATE_REGISTER declaration		✓
ASYNC_RESET keyword		✓
SYNC_RESET keyword		✓
XOR_Factors keyword		✓

Signal Assignments

In ABEL, the ":=" assignment operator should be used when writing pin-to-pin registered equations. The "=" assignment operation should be used for registered equation using detailed dot extensions. PHDL does not require the correct use of the ":=" assignment operator, the "=" assignment operator could be used for registered equations.

Use of the ":=" assignment operator for signals with dot extensions as shown in Figure 1 cause warnings in ABEL, however, these are allowed in PHDL..

```
dareg.d := insig.pin;
```

Figure 1: Signal Assignments with Dot Extensions

The assignment of a set to a non-set element was allowed in PHDL, but is not allowed in ABEL, therefore designs containing code like the example in Figure 2 will need modification.

```
siga node istype ‘com’;
off = [0,1];
...
...
siga = off;
```

Figure 2: Assignment of a Set to a Non-set Element

PHDL also allows a numerical literal to be used in a boolean operation where the bits of the literal are assigned to the corresponding bits of the variable, node, bus, or pins. Note, however, that in ABEL, when a numeric literal is used in a boolean operation, only the least significant bit of the numeric value is used. For example, the code shown in Figure 3.

```
out7..out0 node istype ‘reg’;
out = [out7..out0];
...
...
out.d = ^b10101010;
```

Figure 3: Signal Assignments using Numeric Literals

In PHDL assigns `out7.d` to ’1’, `out6.d` to ’0’, etc. In ABEL, however, all bits of out are assigned ’0’ because ABEL evaluates ’= `^b10101010`’ as a single binary value instead of performing the desired set assignment. The proper way to perform this function in ABEL is shown in Figure 4. PHDL files that use numeric literals in signal assignments will need to be modified.

```
out7..out0 node istype ‘reg’;
out = [out7..out0];
...
...
out.d = [1,0,1,0,1,0,1,0];
```

Figure 4: Correct ABEL Assignment of Numeric Constant

Pin Assignments

ABEL requires the use of single-quotes for alpha-numeric pin assignments. This was not required by PHDL. PHDL designs targeting devices in BGA-type packages will require minor modifications to include single-quotes in the pin assignment as shown in Figure 5.

```
reset pin ‘N23’;
```

Figure 5: Alpha-Numeric Pin Assignment

Attributes

Table 3 describes the attributes supported by ABEL and indicates if these attributes are supported in PHDL.

It is important to note that the PHDL compiler will do syntax checking on signals that have register attributes (reg, reg_d, reg_g, reg_jk, reg_sr, and reg_t) to insure that these signals have a clock (`.clk`) assignment and a consistent input assignment (`.d` for reg_d, `.t` for reg_t). If a signal with a register attribute does not have clock assignment or has an inconsistent input assignment statement, an error is generated.

ABEL does not require that a clock equation exist for registered signals. It will assume the signal on global clock 0 is the clock for registered signals without a clock signal defined. Also, ABEL does not require that detailed dot extension (`.D, .T`) equations exist, but does require that the correct assignment operator is used.

Also note that CoolRunner CPLDs do not have an inverter between the macrocell and the output pin. The INVERT attribute results in the inverter being emulated in the equations for the register input and equations for AR and AP are swapped. However, since CoolRunner 22V10s do have an inverter between the macrocell and the output pin, the INVERT attribute is handled as expected. The INVERT attribute has no meaning for combinatorial signals.

Table 3: Attributes Supported by ABEL

ABEL Attribute	Description	Supported In PHDL	Not Supported In PHDL
buffer	Use non-inverted output.	✓	
collapse	Collapse this signal during logic synthesis.	✓	
com	Combinatorial output.	✓	
keep	Do not collapse this signal during logic synthesis.	✓	
reg	Clocked memory element.	✓	
reg_d	D type flip-flop clocked memory element.	✓	
reg_g	Gated D type flip-flop clocked memory element.	✓	
reg_jk	JK type flip-flop clocked memory element.	✓	
reg_sr	SR type flip-flop clocked memory element.	✓	
reg_t	T type flip-flop clocked memory element.	✓	
retain	Do not minimize this output. Preserve redundant product terms.	✓	
invert	Use inverted output.	✓	
dc	Unspecified logic is "don’t care".		✓
pos	Unspecified logic is "1".		✓
neg	Unspecified logic is "0".		✓
xor	XOR gate in target device.		✓

Reserved Words

The words in Table 4 are reserved in ABEL, but not in PHDL. PHDL designs using these words will need modification.

Table 4: Reserved Words

enable	options
external	state_register
flag	sync_reset
fuses	test_vectors
library	trace

Note: ABEL does not allow strings over 324 characters.

Properties

ABEL allows for the setting of device specific properties. The property statement specifies information about the design to the compiler and fitter for the target device. The syntax for specifying these properties in ABEL is shown in Figure 6.

```
Property_id PROPERTY 'string';
```

Figure 6: Property Syntax in ABEL

For CoolRunner CPLDs, the `property_id` is XPLA. The syntax for the property statement is shown in Figure 7.

```
XPLA PROPERTY 'string';
```

Figure 7: XPLA Property Statement

Figure 8 provides example property statements.

```
XPLA property 'fm_group a b c d';
XPLA property 'config_master_serial';
XPLA property 'maxpt bit0:12 bit1:12';
```

Figure 8: Example Property Statements

Table 5 lists available XPLA properties and gives a brief description of these properties.

Table 5: Available XPLA Properties

Property	Description
dut off	Disable global tri-state function (default).
dut on	Enable global tri-state function (GTS pin is now active).
isp on	Reserve TCK, TMS, TDI , and TDO pins for ISP usage (default).
isp off	Disable ISP capability of the device; allows TCK, TMS , TDI, and TDO to be used as general purpose I/O.
maxpt	Specify the maximum product terms for a specific pin or node.
keep	Specify the keep attribute for a specific signal.
retain	Specify the retain attribute for a specific signal.
tri-state all	Disable the weak pulldown on unused I/O (all devices except 32 macrocells).
fm_group	Group the specified signals within a fast module (XPLA 2).
lb_group	Group the specified signal within a logic block.
slow_slew_rate	Assign slow attribute to output buffers to specify the slow slew rate, default is fast slew rate.
config_master_serial config_master_parallel config_slave_serial config_slave_parallel config_sync_peripheral	Specify the configuration mode so that the fitter will not use the pins required by this mode unless it is necessary to fit the design (XPLA2).

Comments

PHDL and ABEL will accept either a double quotation mark (") or double forward slash (//) to begin a comment (Figure 9).

```
"This is a comment using a quotation mark ending with end of line accepted by PHDL and ABEL
"This is a comment using beginning and ending quotation marks accepted by PHDL and ABEL"
//This is a comment using the double forward slash accepted by PHDL and ABEL
```

Figure 9: Comments - Double Quotation Mark, Quotation Mark and Double Forward Slash

PHDL will also allow comments to begin with a slash-star and end with a star-slash - just like comments in "C" programs. This type of comment can extend over more than one line. This type of comment is not supported by ABEL. PHDL files that use this style of comments will have to be modified (Figure 10).

```
/* This is a ‘C’ style comment that is accepted in PHDL. It may extend
over more than one line. IT IS NOT SUPPORTED BY ABEL.*/
```

Figure 10: "C" Style Comment

Hierarchy and Macros

For signals that cross a hierachical boundary, PHDL allows signal attributes to be defined in both the upper and lower level modules. In ABEL, duplication of signal attributes in both levels of hierarchy is not allowed. Signal attributes can only be defined in the lower-level module. When instantiating the lower-level source module, any signal attributes (implicit or explicit) of the lower-level signals are inherited by the higher-level signals that map to these. Therefore, attributes should not be specified for the signals in the higher-level module.

The optional inclusion of the INTERFACE statement in the lower-level modules is supported in ABEL but is not supported in PHDL.

Special Constants

PHDL supports only the most common ABEL special constants as shown in Table 6.

Table 6: ABEL Special Constant’s

ABEL Special Constant	Description	Supported In PHDL	Not Supported In PHDL
.C.	Clocked input (Low-High-Low transition)	✓	
.D.	Clock down edge (High-Low transition)		✓
.F.	Floating input or output signal		✓
.K.	Clocked input (High-Low-High transition)		✓
.P.	Register preload		✓
.SVn	n = 2 – 9. Drive the input to super voltage 2 – 9		✓
.U.	Clock up edge (Low-High transition)		✓
.X.	"Don’t care" condition	✓	
.Z.	Tri-state value	✓	

Revision History

Date	Version #	Revision
10.22.99	1.0	Initial Xilinx release.