Design Optimization Using *Warp*™ Synthesis Directives

Introduction

Cypress PLDs can implement a wide range of design densities and speeds because they have a flexible and clean architecture. *Warp* is Cypress's sophisticated PLD design tool that takes advantage of this flexibility and gives designers a number of techniques for optimizing design performance.

This application note introduces synthesis directives and shows the tradeoffs that can be made to gain the best possible densities and speeds for VHDL or schematic implementations. It discusses various *Warp* synthesis directives, their formats and the purpose of each directive.

Synthesis Directives

Synthesis directives may be used to influence the implementation of a design. They are used in an iterative fashion to refine, improve, or constrain the results of synthesis. Synthesis directives may be applied to components that have been either instantiated in a schematic or inferred by the synthesizer from VHDL code.

Design Flow and Strategy for Using Directives

After synthesis and fitting the design may fit in the desired device and meet timing goals. In this case the design is complete and no directives are necessary. If, however, after the initial iteration of synthesis and fitting, the design does not fit or meet timing goals, the design may need tuning (*Figure 1*).

Tuning is the process of (1) identifying and applying an appropriate directive that may help to reduce resource utilization or realize timing targets, (2) resynthesizing and fitting the design, and (3) verifying that the design meets area and speed goals. In some cases, this tuning process may have to be repeated in order to compare multiple implementations of the design.

Table 1, shown on the next page can be used to select an appropriate directive for tuning a design. Some directives are functional directives and can have a significant impact on the area and speed of a design while others are used for documentation purposes. The table summarizes the available directives and whether they can be used for area optimization, speed optimization, specific control, or documentation.

Scope and Inheritance

Each of the synthesis directives has a *scope*: some are intended for signals, others for components. Some of the directives also have an *inheritance*. A directive intended for a signal can be placed on an architecture or entity so that all signals defined in that architecture or entity inherit that directive. This is called hierarchical inheritance. Not all directives have an inheritance, however. Non-hierarchical directives are meant for the exact object to which they are attached and will be ignored if not applied to the appropriate object. Hierarchical directives have the following order of precedence (from least to greatest):

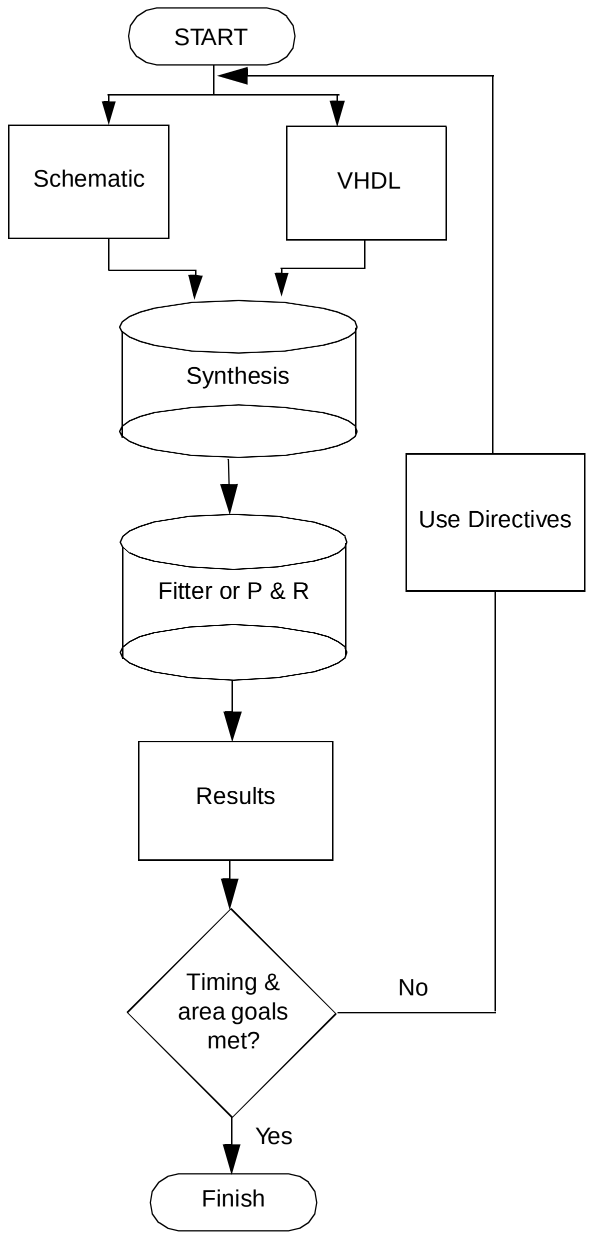

Figure1.

- entity
- architecture
- component declarations
- component instantiations
- signals

Table 1. Available Synthesis Directives

Directive	Used for			
	area	speed	control	doc
goal	x	x		
state_encoding	x	x		
synthesis_off	x	x	x	
dont_touch	x	x	x	
no_latch	x	x	x	
lab_force			x	
pin_avoid			x	
polarity			x	
sum_split			x	
node_num			x	
ff_type			x	
opt_level	x	x	x	
part_name	x	x		x
order_code	x	x		x
pin_numbers	x	x		x

Thus, a hierarchical directive placed on an architecture is overridden by a directive placed on a signal within that architecture. In other words, a hierarchical directive intended for a signal, if placed on an architecture, serves as a default for all signals within that architecture. Likewise, a hierarchical directive placed on a component instantiation overrides a directive placed on an architecture. This allows for an occurrence of a component to have a different directive value than the default directive for all components.

Applying Directives

Some directives are available via the command line or *Warp* GUI. *Warp* also provides three other methods for applying synthesis directives: with VHDL attributes, with schematic attributes, or with a top-level control file. Values of directives passed through the GUI or the command line act as default values. Directives applied using VHDL attributes, schematic attributes, or the control file override default values. The only exceptions are the part_name and order_code directives. The GUI or command line will override all part_name and order_code attributes.

Using the GUI or Command Line

Certain directives may be controlled from the GUI or command line. An example of this is the goal attribute which can be selected to provide area or speed optimization. If speed is selected, then it becomes the default value. If a component has a VHDL or schematic goal attribute applied to it, however, and the value of the attribute is area, then the speed value is overridden with the area value for that component.

Using VHDL Attributes

VHDL permits the use of user-defined attributes to attach information to objects. *Warp* has thus created a user-defined (as opposed to predefined) attribute for each directive. This permits a directive to be applied to an object with the use of an attribute. The general syntax of an attribute used to place a directive on an object is of the form:

```
ATTRIBUTE directive_name OF object: class IS value;
```

Such attributes are placed in the appropriate declarative region of the VHDL code, typically in either the entity declarative region or the architecture body declarative region. The object is the actual name or identifier of the entity, architecture, component instantiation label, or signal. Class is used to identify the class of the object (i.e., entity, architecture, or component instantiation label, or signal).

Using Schematic Attributes

Directives may be applied to objects in schematics (with *Warp3®*) using attributes by selecting the appropriate object and choosing *Attribute* from the *Add* menu. After selecting *Add->Attribute*, a dialog box appears in which the user may enter the directive in the form:

```
directive_name=value
```

The goal directive for area or speed optimization is not applied as an attribute. It is selected during the addition or modification of an LPM symbol. The directive selected here overrides the command line or GUI switch.

Using a Control File

A control file provides a common location for setting global synthesis directives for a given design. Cypress prefers the use of the control file since it gives the user detailed control over many aspects of synthesis while maintaining a device and vendor independent VHDL source file. In the case of conflict, directives placed in a control file override directives specified with VHDL or schematic attributes. Only one control file is allowed per design and the file should have the same base name as the top level design file name. Each directive may be applied in the control file using a syntax similar to that of attributes:

```
attribute directive_name [of] object[:class] is value[;]
```

The words in square brackets [ ] are optional and are simply ignored. Specifying the class is also optional. *Warp* also supports the ‘*’ wild-card character that allows pattern matching.

Area Optimization

This section describes the directives and techniques required to successfully implement a logic design using the minimum device resources.

The goal Directive

```
ATTRIBUTE  goal OF architecture_name : ARCHITEC-
TURE IS AREA;
```

The goal value of area indicates that all modules inferred from VHDL operators will be optimized for area. The *Warp* synthesizer will select an implementation that is optimized to use the minimum device resources.

<u>*Scope*</u>

Target: Architecture or Entity

Inheritance: None

Related Command-Line-Option: -yga

Applicable to: All Devices

The synthesis_off Directive

ATTRIBUTE synthesis_off OF signal_name : SIGNAL IS true;

When the synthesis_off directive is set to true, a signal is made into a factoring point for logic equations. This directive keeps the signal from being substituted out during the optimization process.

Synthesis_off is useful for the following reasons:

- It gives the user control over which equations or sub-expressions need to be factored into a node.
- It provides better results for designs where a signal with a large functionality is being used by many other signals. If left alone, the fitter would collapse all the internal signals (which is desirable in many cases) and may drive the design's resource requirements beyond the available limits.
- It helps reduce compile time for designs which have a lot of "signal redirection" (signals getting inverted or reassigned to other signals).
- This directive provides the logic optimizer better control over the optimization process, by reducing the number of signals it needs to process.

By using the synthesis_off directive, the user can assign the commonly used signal to a node and improve the resource utilization.

A side effect of using the synthesis_off directive is that the design will now take an extra pass through the array. The extra pass is usually required anyway, if more than 16 PTs are required.

This directive is recommended only on combinatorial signals. Registered signals are assigned to a node by natural factoring, and the synthesis_off directive on these signals is redundant.

This directive can be associated with signals declared both in VHDL and schematics. This directive allows the designer to force multiple passes through logic cells for optimal density.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: -v#

Applicable to: All devices

The following example requires 37 Macrocells and 376 unique product terms in a CY7C374i without using the synthesis_off directive (*Figure 2*). This same design requires 22 Macrocells and 146 unique product terms in a CY7C371i with the synthesis_off directive (*Figure 3*).

Example:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity cpldadd is port(
a: in std_logic_vector(7 downto 0);
b: in std_logic_vector(7 downto 0);
c: in std_logic_vector(7 downto 0);
    sum: out std_logic_vector(7 downto 0));
end cpldadd;
architecture areacpldadd of cpldadd is
    signal intsum: std_logic_vector(7 downto 0);
    attribute synthesis_off of intsum:signal is true;
begin
    intsum <= a + b;
    sum <= intsum + c;  -- without synthesis_off (a+b) would
 end areacpldadd;       --be substituted in the sum equation
```

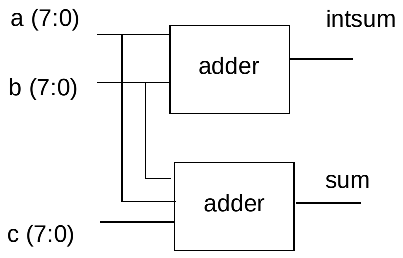

Figure 2. Without Synthesis_off directive

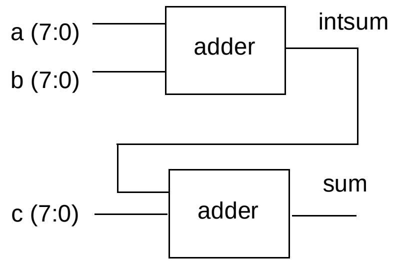

Figure 3. With Synthesis_off directive

The ff_type Directive

ATTRIBUTE ff_type OF signal_name : SIGNAL IS ff_opt;

The ff_type value of ff_opt tells *Warp* to synthesize the signal_name to the optimum flip-flop type for the logic implemented. A flip-flop is chosen based on the fewest resources required to implement the logic function. For instance, a D-type flip-flop may be chosen for register data storage functions, while a T-type (toggle) flip-flop may be chosen for counters. This option is recommended for all designs unless the designer has specific requirements to force the use of a different flip-flop type. The VHDL attribute is placed in the architecture body declarative region.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: -fo

Applicable to: All Devices

The state_encoding Directive

The state_encoding directive specifies the internal encoding scheme for values of an enumerated type.

```
ATTRIBUTE state_encoding OF type-name: TYPE IS
value;
```

The legal values of the state_encoding directive are:

- sequential
- one_hot_zero
- one_hot_one
- gray

When the state_encoding directive is set to *sequential*, the internal encoding of each value of the enumerated type is set to a sequential binary representation. The first value in the type declaration receives an encoding of 00; the second, 01; the third, 10; the fourth, 11; and so on. Sufficient bits are allocated to the representation to encode the number of enumerated type values included in the type declaration. When the state_encoding directive is set to *one_hot_zero*, the internal encoding of the first value in the type definition is set to 0. Each succeeding value in the type definition has its own bit position in the encoding. That bit position is set to 1 when the state variable has that value. Thus, a *one_hot_zero* encoding of an enumerated type with N possible values requires N – 1 bits. For example, if an enumerated type had four possible values, three bits would be used in its one_hot_zero encoding. The first value in the type definition would have an encoding of 000. The second would have an encoding of 001. The third would have an encoding of 010. The fourth would have an encoding of 100. *One_hot_one* state encoding works similarly to one_hot_zero, except that no zero encoding is used; every value in the enumerated type has a bit position, which is set to one when the state variable has that value. Thus, a *one_hot_one* encoding of an enumerated type with N possible values requires N bits. For example, if an enumerated type had four possible values, four bits would be used in its *one_hot_one* encoding. The first value in the type definition would have an encoding of 0001. The second would have an encoding of 0010. The third would have an encoding of 0100. The fourth would have an encoding of 1000. When the state_encoding directive is set to *gray*, the internal encoding of successive values of the enumerated type follows a *Gray* code pattern, where each value differs from the preceding one by only one bit.

Scope

Target: Type

Inheritance: None

Related Command-Line-Option: None

Applicable to: All Devices

Examples:

```
type state is (s0,s1,s2,s3);

attribute state_encoding of state:type is one_hot_zero;
```

The first statement in this example declares an enumerated type, called state, with four possible values. The second statement specifies that values of type state are to be encoded internally using a *one_hot_zero* encoding scheme. The VHDL attribute is placed in the architecture body declarative region.

Speed Optimization

This section describes the synthesis directives that may be used in optimizing a design for performance. In most cases, the techniques for speed optimization are device dependent.

The goal Directive

```
ATTRIBUTE goal OF architecture_name: ARCHITEC-
TURE IS speed;
```

The goal attribute value of *speed* indicates that all arithmetic modules inferred from VHDL operators will be optimized for speed. The *Warp* synthesizer will select an implementation that is optimized to achieve the best performance. This is a good first step to take when optimizing a design for performance. To demonstrate the goal directive, observe the performance delta in the following 8-bit adder example implemented in a FLASH371i CPLD:

Example:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity add8_a is port(
    a, b: in std_logic_vector (7 downto 0);
    sum: out std_logic_vector (7 downto 0));
end add8_a;
architecture archadd8_a of add8_a is
    attribute goal of archadd8_a: architecture is speed;
begin
    sum <= a + b;
end archadd8_a;
```

The maximum delay with the goal attribute set to area is 33 ns and with the goal attribute set to speed is 27 ns.

Directives for Specific Control

This section describes specific control features of the *Warp* synthesis tool.

The ff_type Directive

```
ATTRIBUTE ff_type OF signal_name : SIGNAL IS ff_d;
```

or command line option: -fd

The ff_type value of ff_d tells *Warp* to synthesize the signal_name using a D-type flip-flop. This will force the synthesizer to use a D-type flip-flop to generate signal_name. This directive will typically only be used if the *Warp* synthesis tool is not using the D-type flip-flop where the designer intends.

```
ATTRIBUTE ff_type OF signal_name : SIGNAL IS ff_t;
```

or command line option: -ft

The ff_type value of ff_t tells *Warp* to synthesize the signal_name using a T-type flip-flop. This will force the synthesizer to use a toggle flip-flop to generate signal_name. This directive will typically only be used if the *Warp* synthesis tool is not using a toggle flip-flop, which the designer intends for functional reasons.

The lab_force Directive

ATTRIBUTE lab_force OF signal_name : SIGNAL IS "string";

This attribute will force signal_name into the logic block specified by string. It can be used for floor planning purposes or when the user needs direct control of how product terms are allocated in a logic block. This directive should only be used if the user is intimately familiar with the target CPLD architecture. This directive can cause routing difficulties if logic is placed in an area that can block routing paths. The VHDL attribute is placed in the architecture body declarative region.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: None

Applicable to: CPLD Devices Only

Examples:

ATTRIBUTE lab_force OF ff_Q: SIGNAL IS "B2";

This will force the signal ff_Q to the lower half of logic block B in a FLASH370i device.

ATTRIBUTE lab_force OF ff_Q:signal IS "B1";

The signal ff_Q is forced to the upper half of logic block B.

The no_factor Directive

The no_factor directive prevents logic factoring within the *Warp* synthesis engine.

ATTRIBUTE no_factor OF signal_name: SIGNAL is value;

During the optimization phase, the *Warp* synthesis engine, aliases signals which have identical drivers (equations). This feature can be useful if the design constraints require certain identical logic to be duplicated or if the logic factoring algorithm is being overaggressive. The VHDL attribute is placed in the architecture body declarative region.

Scope

Target: Signal

Inheritance: Hierarchical

Related-Command-line-option: -fl

Applicable to: All Devices

Examples:

attribute no_factor of my_signal: signal is true;

This example prevents the signal my_signal from being aliased or from being factored.

attribute no_factor of my_architecture:architecture is true;

This example prevents all signals in my_architecture and its sub-architectures from being aliased or factored.

The no_latch Directive

The no_latch directive prevents latches from being synthesized automatically for the signal in question.

ATTRIBUTE no_latch of signal_name: SIGNAL IS value;

Normally, when exhaustive optimization is enabled (with the -o2 option), *Warp* tries to synthesize latches where possible for the FLASH370i family.

The following example creates a latch with y as the enable and a as the latched data for the equation x:

```
if (y = '1') then
    x <= a;
else
    x <= x;
end if;
```

Creating a latch in this case saves a product term for the x equation; however, this has certain other side-effects that might not be desirable:

- If the synthesizer also produced asynchronous resets/presets for the enable, this will cause more global resources to be used.
- Creating a latch might have caused a slower design and introduced setup/hold problems.

Using the no_latch directive would cause *Warp* to create simply a signal with a combinatorial delay.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: -yl

Applicable to: FLASH370i Devices Only

Example:

attribute no_latch of x: signal is true;

In this example, the directive causes latch detection to be disabled for signal x.

The node_num Directive

ATTRIBUTE node_num OF signal_name : signal IS integer;

The node_num directive locks a signal to a specific location in the target device. This directive overrides the default placement that the *Warp* tool would assign automatically. This directive applies to any combinatorial or sequential node within the design.

Scope

Target: Signal

Inheritance: None

Related Command-Line-Option: -fn [n=node location]

Example:

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY node_num_test IS PORT (
    clk, ff_D: IN STD_LOGIC; -- Flip-flop clock, D-input
     ff_Q: OUT STD_LOGIC); -- Flip-flop Q output
    ATTRIBUTE part_name of node_num: ENTITY IS "C374i";
    ATTRIBUTE node_num OF ff_Q:SIGNAL IS 398;
END node_num_test;
ARCHITECTURE arch_node_num_test OF node_num_test IS
```

```
BEGIN
    PROCESS
        BEGIN
            WAIT UNTIL clk = '1';
           ff_Q <= ff_D; -- Generate output
     END PROCESS;
END arch_node_num_test;
```

The previous code segment ensures the signal ff_Q is generated from the macrocell driving node 398 in a CY7C374i device. Node 398 refers to the first macrocell in logic block #1 in a CY7C374i. Refer to the FLASH370i appendix in the *Warp* Reference manual for specific node numbers. This directive is similar to the lab_force attribute but provides even more control.

The sum_split Directive

ATTRIBUTE sum_split OF signal_name : SIGNAL IS value;

The FLASH370i can generate 16 product terms in one pass through the array. To implement an equation with more than 16 product terms the design has to take an extra pass. The value of the sum_split attribute can be balanced or cascaded. The default value is balanced. Use the balanced value if reliable balanced timing is desired. *Figure 4* illustrates the balanced sum split concept:

ATTRIBUTE sum_split OF sum_signal: SIGNAL IS balanced;

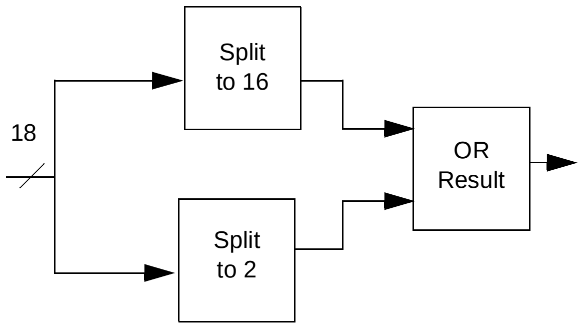

Figure 4. The Balanced Sum Split

The cascaded method (*Figure 5*) uses only two macrocells to implement an equation. There is no control over which product term is assigned to which macrocell. The signals that are not split into macrocell #1 will arrive at macrocell #2 sooner, thereby making the timing for the outputs different based on different arrival times which may result in a combinatorial glitch. If these output signals are registered, then of course the timing generated at the outputs is the same.

ATTRIBUTE sum_split OF sum_signal: SIGNAL IS cascaded;

Which sum_split method to use depends on the area constraints and how the design is implemented. Use the balanced method first and then the cascaded, if the design does not fit using balanced method.

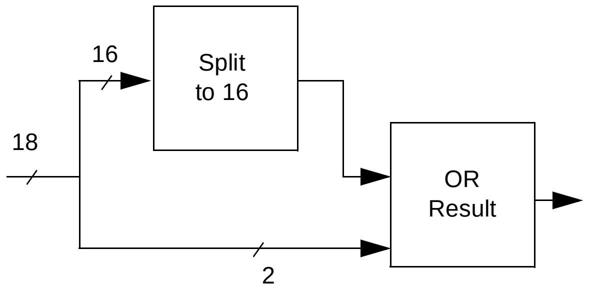

Figure 5. The Cascaded Sum Split

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line Option: None

Applicable to: CPLD Devices only

The polarity Directive

ATTRIBUTE polarity OF signal_name : SIGNAL IS value;

The polarity directive is used to select polarity for internal signals in a design. There are two options for polarity, pl_keep and pl_opt. The pl_keep option will instruct the *Warp* compiler to keep the polarity of a signal as currently specified in the design. The pl_keep option is useful to instruct the compiler about the desirable output sense of an internal signal at power up. When a circuit is initialized, it may be desirable to provide an output as a “1” or “0” and maintain this condition without the compiler changing the sense for optimization reasons. In another case, it may be desirable to keep signal senses in order to debug designs in the simulator without being concerned about compiler-induced internal inversions. In most cases, however, the pl_opt is the best choice. This option allows the compiler to change the sense of internal signals to provide the best optimization for a design.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: -fp or -fk

Applicable to: All PLDs

The opt_level Directive

The opt_level directive instructs *Warp* on the amount of effort that should be spent optimizing certain signals.

ATTRIBUTE opt_level OF signal_name: SIGNAL IS integer;

The integer represents the amount of effort. Currently, there are three levels of effort (0, 1 and 2). An opt_level of 0 instructs *Warp* to turn off all optimization on the signal specified. This directive is also passed along to the PLD/CPLD fitters which do the same thing. An opt_level of 1 causes *Warp* to perform a simple and quick optimization of equations. An opt_level of 2 causes *Warp* to perform the highest level of optimization available. An opt_level of 2 is recommended for all designs.

Scope

Target: Signal

Inheritance: Hierarchical

Related Command-Line-Option: -o#

Applicable to: All PLDs

Example: attribute opt_level of my_signal:signal is 0;

This directive disables all optimization on the signal my_signal.

The pin_avoid Directive

The pin_avoid directive is a string type directive that prevents the fitter from mapping any signals to the specified pins. This directive is only valid on the top-level entity of the design.

ATTRIBUTE pin_avoid OF entity-name: ENTITY IS "string";

The string used in the directive statement consists of one or more pin-numbers. Each pin-number must be separated by white space (spaces or tabs). This string can consist of several smaller, concatenated strings. This feature can be used if certain pins are being used for some special purposes (such as with In System Reprogrammable devices—ISR) or need to be reserved for some future functionality. When this feature is used, the report file indicates these pins as Reserved in the pin table. Such pins are named as Reserved# where # is an index. In the case of CPLDs where I/O pins have macrocells associated with them, this feature does not prevent the fitter from using the buried macrocell portion associated with that particular pin.

Scope

Target: Top-level Entity.

Inheritance: None

Related Command-Line-Option: None

Applicable to: FLASH370i Devices Only

Examples:

attribute pin_avoid of my_design: entity is "2 3 4";

attribute pin_avoid of my_design: entity is "A1 B1 C1";

The first example instructs the fitter to avoid the pins 2, 3 and 4 when trying to place the design into a device. The second example is a case where the package being used is a Pin-Grid-Array, in which the pin-numbers are alpha-numeric.

The pin_numbers Directive

ATTRIBUTE pin_numbers OF entity_name: ENTITY IS "string";

Once a design has been completed and the board is defined, it may be desirable to maintain the pin out configuration when modifications to the programmable logic design are made. Locking signals to a particular pin can be accomplished by using the pin_numbers directive in the design.

Example:

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY and5Gate IS
    PORT   (a: IN std_logic_VECTOR(0 TO 4);
            f: OUT std_logic);
ATTRIBUTE part_name of and5Gate:ENTITY IS "C371i";
ATTRIBUTE pin_numbers of and5Gate:ENTITY IS
    "a(0):2 a(1):3 "      --The spaces after 3 and 5 are
   & "a(2):4 a(3):5 "     --necessary for concatenation (&
    & "f:6";               --operator), signal a(4) will be
 END and5Gate;             --assigned a pin by Warp
ARCHITECTURE archpin_num OF and5Gate IS
BEGIN
    f <= a(0) AND a(1) AND a(2) AND a(3) AND a(4);
END archpin_num;
```

Even though this directive is called pin_numbers, it can also assign PGA package pin-numbers which are in fact alpha-numeric (such as "A1").

It is recommended that whenever possible, particularly the first time a design is fitted to a device, the pins of a device should not be locked. When the pins are not locked, the fitting tools are free to choose the optimal fitting arrangement within the device for performance and minimal resource utilization. In some rare occasions, certain pin arrangements can render a fit impossible. Once a design has been fitted to a device (and the tool has already chosen a working pin configuration), the pin assignments can be back-annotated to the design schematic or control file. The pin_numbers directive can also be used to set the pins of the design.

Documentation Directives

This section describes directives used for documentation purposes.

The part_name Directive

ATTRIBUTE part_name OF entity_name: ENTITY IS "part_name";

A user may want to specify a particular device so that the original design documents specify which device it was designed for. This directive will override any target device command line switch or a *Warp* GUI dialog box setting.

entity counter is port (a,b: in std_logic; ...);

attribute part_name of counter: entity is "c371i";

end entity counter;

The order_code Directive

ATTRIBUTE order_code OF entity_name: ENTITY IS "order_code";

A particular package and speed bin of a device can be specified to the *Warp* synthesis tool by using the directive order_code within the design to ensure timing information reflects the speed grade of the desired part. The order codes can be found in the Ordering Code column of the ordering information table for each device in the *Cypress Semiconductor Programmable Logic Data Book*. Timing delays for CPLDs are calculated according to the speed bin specified by this directive, or if no directive is specified in the VHDL code, the compiler will use the directive specified in the device window of Galaxy.

Example:

```
entity counter is port (
    a,b: in std_logic; ... );
```

attribute order_code of counter: entity is “CY7C371i-66JC”;

end entity counter;

Summary

Directives are a powerful mechanism to influence the synthesis process, but they should be used judiciously. Careless or excessive use of directives can, in fact, subvert the very design goals that are sought. Familiarity with the internal architecture of a PLD and careful use of the directives described in this application note, can help a designer to get the best performance out of Cypress programmable logic devices.The following table summarizes VHDL attribute formats, values, and command line switches.

Warp Directive Formats

Directive	VHDL Format	Values (D=Default)	Command line
goal	attribute goal of arch_name : architecture is *value*;	speed (D), area, or combinatorial	ygs,yga, ygc
state_encoding	attribute state_encoding of type_name : type is *value*;	sequential (D), one_hot_zero, one_hot_one, or gray	--
no_latch	attribute no_latch of signal_name : signal is *value*;	false (D) or true	yl
lab_force	attribute lab_force of signal_name : signal is *location;*	Example: “A1”	--
pin_avoid	attribute pin_avoid of entity_name : entity is *location;*	Example: “1 2 3”	--
polarity	attribute polarity of signal_name : signal is *value*;	pl_default (D), pl_keep, or pl_opt	fk, fp
sum_split	attribute sum_split of *signal_name* : signal is *value*;	balanced (D) or cascaded	--
node_num	attribute node_num of *signal_name* : signal is *value*;	nd_auto (D) or positive integer	fn
ff_type	attribute ff_type of *signal_name* : signal is *value*;	ff_default (D) , ff_d, ff_t , or ff_opt	fd, ft, fo
opt_level	attribute opt_level of signal_name : signal is *integer*;	2 (D), 1, or 0	o
part_name	attribute part_name of entity_name : entity is *string*;	Example: “c371”	d
order_code	attribute order_code of entity_name : entity is *string*;	Example: “PALC22V10-25HC”	p
pin_numbers	attribute pin_numbers of entity_name : entity is *string*;	Example: “sig1: ” & “sig2:2”	ff