Getting Started Converting .ABL Files to VHDL

Introduction

This application note is intended to assist *Warp*™ users in converting designs written in DATA I/O's ABEL™7 hardware description language to IEEE 1076 VHDL. It contains several language cross reference tables and many helpful hints. It also includes two real-world designs that have been converted from MACH™ 210−ABEL descriptions to FLASH371−VHDL descriptions.

VHDL versus ABEL

VHDL is different from ABEL and virtually all other popular hardware description languages in one very significant way. It is an open language based on IEEE standard number 1076.

VHDL is different in other ways, too. VHDL is a high-level language. As such, it is much more powerful than ABEL. For instance, it supports hierarchical design entry, structural (low-level instantiation of components) and behavioral (IF-THEN-ELSE) design entry. VHDL supports process and concurrent process statements. It also supports various types of signals such as integer, real, character, bit, Boolean, physical unit, and any others that a user can define. It supports sequential and concurrent statements, variables, and signals. VHDL supports sub-programs, FOR loops, WHILE loops, arrays, concurrent procedure calls, and more.

Surprisingly, certain aspects of VHDL related to low-level behavioral logic description are very similar to ABEL. In fact, some key words and relational operators are identical or logically similar.

Conversion Preparation

Preparing to convert an ABEL (.ABL) file should include the following steps:

1. Locate and have at hand one good VHDL language reference book. See the *Warp* documentation for a bibliography.
2. Obtain copies of the *Warp* VHDL design examples titled Basic, Intermediate, and Advanced.
3. Locate two Cypress application notes, one titled "Designing State Machines with *Warp2*™ VHDL" and another titled "VHDL Techniques for Optimal Design Fitting." Both are contained in the *Cypress Applications Handbook* (1993).
4. Install the *Warp* VHDL compiler on your hardware platform.

Conversion Approach

There are many different ways to convert a given design. The same design may be expressed in a number of different ways, all yielding compiled designs with the same functionality. The general approach suggested for converting ABEL (.ABL) files to VHDL (.VHD) consists of five basic steps:

1. Analyze the design and determine:
   a. Which signals are registered and which are not. Group them into two categories.
   b. Which types of design entry the .ABL file includes: state machines, comparators, counters, decoders, multiplexers, adders, multipliers, shift registers, or state tables.
   c. Whether or not group (set) declarations are used.

d. Which signals are input, output, I/O, and/or active LOW.

2. Replace as many of the keywords and operators with the corresponding VHDL keywords and operators using your favorite SEARCH and REPLACE text editor and a backup copy of the .ABL or .DOC file.

3. Add the VHDL entity (black box inputs and outputs), architecture (description of the logic circuitry), and process (encapsulates a set of sequential-behavioral functions) statements.

4. Add the proper library USE statements to the file such as USE WORK.CYPRESS.ALL.

5. Iteratively compile the design revising incomplete or incorrectly converted syntax.

Some designs will be much easier to convert than others. The more regular the design the easier it will be to convert. The most efficient and highest level of conversion will be achieved by using the source (.ABL) file, the five steps above, and the cross reference information below.

The simpler approach is to use the .DOC file exclusively. Using the .DOC file works but does have one significant drawback. The .DOC file tends to be verbose. It is verbose because it describes the design at a low level. A converted ABEL (.DOC) design thus results in unnecessarily verbose VHDL. In other words, it results in inefficient code.

When converting using the .DOC file, place all of the registered signals into a process with a WAIT UNTIL CLOCK = '1' and place all of the combinatorial signals outside the process. This avoids the necessity of PROCESS sensitivity lists and IF-THEN-ELSE statements. The converted designs below and all of the *Warp* example designs attempt to describe functions behaviorally and at a higher level. For this reason no low-level design conversion examples are included.

Refer to the following sections and tables for helpful information when converting ABEL .ABL and .DOC descriptions.

Comments

Comments in ABEL are denoted by the quote symbol (”). Comments in VHDL are denoted by two consecutive dashes −−. For example:

ABEL	VHDL
”Inputs	−−Inputs
”Outputs	−−Outputs

VHDL-ABEL Special Constant Cross Reference

ABEL	VHDL	Description
.C.	requires user definition	Clocked input (0 −>1−> 0)
.D.	requires user definition	Clock down edge (1 −> 0)
.F.	requires user definition	Floating input or output
.K.	requires user definition	Clocked input (1−> 0−> 1)
.P.	requires user definition	Register preload
.SVn.	requires user definition	Super voltage ($2 \le n \le 9$)
.U.	requires user definition	Clock up edge (0 − > 1)
.X.	requires user definition	Don't care condition
.Z.	requires user definition	High impedance

In VHDL a constant is an object whose value may not be changed. The syntax for declaring a constant in VHDL is:

constant identifier_list : type [:=expression];

An example of this is:

```
TYPE stvar is bit_vector (0 to 1);
constant State0 : stvar  := “00”;
```

This example declares a constant that is identified by the name State0, is of type stvar, which has been previously defined as a bit_vector subtype of length 2. This constant is given the value 00. Defining a constant of user-defined type called state variable (stvar) is useful when designing state machines in VHDL. See the State Machine section of this application note.

Special constants in ABEL are used for simulation vectors that are included in the source file (.ABL). *Warp* does not provide simulation support directly within the source file. So, the conversion recommendation for files containing simulation vectors is to delete or comment them out of the .VHD file. *Warp* provides simulation separately from the design file (.VHD). Simulation can take one of two forms. The first, functional waveform based design verification using NOVA. Second, full AC timing verification via VIEWSIM and VIEWTRACE. VIEWSIM and VIEWTRACE support both pattern files and waveforms. Both forms of *Warp* VHDL simulation exceed the capabilities of ABEL simulation.

VHDL-ABEL Dot Extension Cross Reference

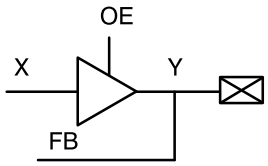
ABEL	VHDL	Function
.CLK	none	Clock input to flip-flop
.PIN	none	Pin feedback
.FB	none	Register feedback
.D	none	D-type flip-flop
.J	none	J input to JK flip-flop
.K	none	K input to JK flip-flop
.S	none	S input to SR flip-flop
.R	none	R input to SR flip-flop
.T	none	T-type flip-flop
.Q	none	Register feedback
.PR	none	Register preset
.RE	none	Register reset
.SP	none	Synchronous reg preset
.SR	none	Synchronous reg reset
.LE	none	Latch-enable input
.LH	none	Latch-enable input (HIGH)
.LD	none	Register load input
.CE	none	Clock enable input
.AP	none	Asynchronous preset
.AR	none	Asynchronous reset
.OE	none	Three-state output enable
.CLR	none	Synchronous clear
.ACLR	none	Asynchronous clear

VHDL-ABEL Dot Extension Cross Reference (continued)

ABEL	VHDL	Function
.SET	none	Synchronous set
.ASET	none	Asynchronous clear
.COM	none	Combinational feedback
.FC	none	Flip-flop mode control

Although VHDL is capable of supporting these constructs, it directly does not. Indirectly, through behavioral description, structural description, and an intelligent compiler, all of these constructs are supported. For example, *Warp* does provide predefined register transfer level (RTL) components (such as D- and T-type flip-flops). These RTL components can be structurally instantiated to model the ABEL extensions listed above. Specifically, to model the .OE ABEL extension, use an RTL component called *bufoe*. The syntax and port map (inputs and outputs) of a *bufoe* component is the following:

Label: BUFOE PORT MAP(X, OE, Y, FB)



X, OE, Y, and FB are sample signal names. The position each one occupies in the port map is the mechanism VHDL uses to correctly connect the signal names to the actual component in the architecture of the target device.

The behavioral equivalent of structurally instantiating a bidirectional buffer is called a behavioral three-state. Behavioral three-states are presently not supported, but will be in the future.

If the ABEL description equation is written in .T (T-type) flip-flop style, the recommended conversion method is to rewrite the equations as D-type (XOR the original equation with the flip-flop output signal name) and let *Warp* optimize the equation for either D- or T-type. See the real-world design conversion example in Appendix A called FLAGCTLR.

IS_TYPE Attribute Cross Reference

ABEL	VHDL	Description
'buffer'	none, may use RTL – buf	Macrocell has no inverter between reg and pin
'com'	none, may use RTL – buf	Signal is combinatorial
'invert'	none, NOT RHS of equation	Macrocell has an inverter between reg and pin
'neg'	none, NOT signal in equation	Complement Sum of Products
'pos'	none	Do not Complement Sum of Products
'reg'	none, place sig in process	Generic flip-flop
'reg_D'	none, may use RTL – dff	D-type flip-flop
'reg_T'	none, may use RTL – tff	T-type flip-flop
'reg_SR'	none, may use RTL – srff	SR-type flip-flop
'reg_JK'	none, may use RTL – jkff	JK-type flip-flop
'reg_G'	none	D-flip-flop w/gated clock
'xor'	none, may use RTL – xbuf	Target architecture has XOR

Operator Cross Reference

ABEL	Order of Precedence	VHDL	Order of Precedence	Operation
!	1	NOT	Context dependent	NOT (invert)
&	2	AND	1	AND
*	2	*	5	Multiplication
/	2	/	5	Division
%	2	mod	5	Modulus
<<	2	none		Shift left
>>	2	none		Shift right
+	3	+	3	Arithmetic addition
−	3	−	3	Arithmetic subtr.
$	3	XOR	1	XOR
!$	3	NOT XOR		XNOR
#	3	OR	1	OR
==	4	=	2	Equal
!=	4	/=	2	Not equal
<	4	<	2	Less than
<=	4	<=	2	Less than or equal
>	4	>	2	Greater than
>=	4	>=	2	Greater or equal
none		NAND	1	NAND
none		NOR	1	NOR
none		&	3	Concatenation
none		rem	5	Remainder
none		abs	6	Absolute Value
none		**	6	Exponentiation
?		+	4	Sign
?		−	4	Sign
= or :=		<=		Signal assignment
none		:=		Variable assignment
=		<=		Comb. assignment
:=		<=		Reg. assignment
none		=>		Association

The only ABEL operators without a direct VHDL counterpart are the >> and <<, (shift right and shift left). To directly (structurally) describe logic that performs an N-bit shift left or right function see the *Warp* design example titled advanced SHIFTN.VHD. To emulate (behaviorally describe) a shift function in VHDL use bit_vectors or arrays and index the elements using LOOPs. Another technique is to use *2 and /2 (multiply by 2 and divide by 2).

Keyword (Statement) Cross Reference

ABEL Keyword	VHDL Equivalent
CASE	CASE
DECLARATIONS	Note 1
DEVICE	ATTRIBUTE PART_NAME IS ...
ELSE	ELSE
ENABLE (Obsolete)	none
ENDCASE	END CASE
ENDWITH	Note 2
EQUATIONS	Note 3
FLAG (Obsolete)	none
FUSES	Note 4
GOTO	EXIT − Note 5
IF	IF
IN (Obsolete)	none
ISTYPE	Note 6
LIBRARY	USE
MACRO	FUNCTION − Note 7
MODULE	FUNCTION − Note 8
NODE	SIGNAL
OPTIONS	Note 9
PIN	Note 10
PROPERTY	none
STATE	Note 11
STATE_DIAGRAM	Note 11
TEST_VECTORS	Note 12
THEN	THEN
TITLE	Note 13
TRACE	Note 14
TRUTH_TABLE	Note 15
WHEN	WHEN
WITH	Note 16
ASYNC_RESET	Note 17
SYNC_RESET	Note 17
STATE_REGISTER	Note 18
XOR_FACTORS	Note 19

VHDL does not use the term keyword. Analogous to ABEL's use of the term keyword, VHDL uses the terms statement, reserved word, and identifier.

Notes

1. There is not a DECLARATIONS keyword in VHDL. However, the DECLARATIONS keyword is analogous to declaring an ENTITY in VHDL. Within the ENTITY construct inputs, outputs, and I/Os are declared with appropriate mode and type. Mode loosely refers to the pin drive direction, which can be IN, OUT, or INOUT. Refer to your language reference book for a more formal definition of the terms mode, IN, OUT, and INOUT.
2. ENDWITH is part of the WITH-ENDWITH transition structure used with IF-THEN-ELSE or CASE keywords. In VHDL conditional transition is handled via an IF-THEN-ELSE or CASE statement within a PROCESS. The process statement may or may not use a sensitivity list and instead use a WAIT UNTIL (condition) statement. See the application note titled "Designing State Machines with *Warp2* VHDL."
3. Equations in VHDL are listed within an architecture statement.
4. VHDL and *Warp* do not provide predefined fuse-level program specification.
5. VHDL does not have a GOTO keyword (statement). It provides an EXIT keyword for stopping execution of loops entirely.
6. The IS_TYPE keyword (statement) defines attributes and/or characteristics of pins and nodes. VHDL provides these attributes through behavioral specification. Additionally, *Warp* provides a set of predefined attributes and VHDL provides a mechanism for declaring new attributes. See the attribute table below.
7. VHDL provides function call and return capability. MACRO is more of a low-level substitution technique such that, wherever the MACRO_id occurs, the text associated with that macro will be substituted.
8. The MODULE ... END statement(s) are source file requirements in ABEL. In VHDL the ENTITY-ARCHITECTURE pair are the basic source file requirements. Both the ENTITY and ARCHITECTURE constructs require an END statement.
9. OPTION is a string of processing options that affect the way in which the ABEL source file is processed by the language processor. The analogous control in VHDL is not done in the source file. It in fact is not part of the VHDL language. It is simply a menu of compiler options that are set when using *Warp* to synthesize the design.
10. PIN is used to declare input and output signals that must be available on a device I/O pin. The analogous PIN specification is implied in VHDL via the port map list in the ENTITY construct. All signal names listed in the entity port map are input, output, and I/Os of the entity.
11. See application note titled "Describing State Machines with *Warp*2 VHDL."
12. Test vectors are not directly supported by VHDL. However, both behavioral simulation and full AC timing simulation are available for design verification.
13. The TITLE statement is used to give an ABEL MODULE a title that will appear as a header in both the programmer load file (the JEDEC file) and the documentation file. When compiling a VHDL design using *Warp*, the filename of the VHDL (.VHD) design file is passed through to the programmer load file (.JED) as well as the documentation file (.RPT).
14. The TRACE statement controls the display features of ABEL's simulator. There is not a similar keyword in VHDL because simulation is separate from the source file description.
15. The TRUTH_TABLE keyword is used in ABEL to specify outputs as functions of different input combinations in a tabular form. VHDL does not directly provide a TRUTH_TABLE keyword. However,

in the common library (directory) included in *Warp*, there is a file called LIBSTATE.VHD that contains a FUNCTION called TTF. TTF is a predefined truth table function that can be used for both combinatorial truth tables and for state transition tables. See the application note titled "Describing State Machines in *Warp*2 VHDL" regarding use of the TTF function.

16. WITH is part of the WITH-ENDWITH transition structure used with IF-THEN-ELSE or CASE statements. In VHDL, conditional transition is handled via an IF-THEN-ELSE or CASE statement within a PROCESS. The process statement may use a sensitivity list or may include a WAIT UNTIL (clock = '1') statement.
17. ASYNC_RESET and SYNC_RESET statements are used in Symbolic State descriptions. They symbolically specify what state the machine should asynchronously or synchronously reset to, based upon a signal or an expression. In VHDL, asynchronous and synchronous resets are best handled from a behavioral perspective the Resets and Presets section of this note for more detail.
18. STATE_REGISTER is a mechanism whereby specific states of a machine can be identified symbolically. See the State Machine section of this note for more detail.
19. XOR_FACTORS is a keyword that is useful for factoring logic designs that target a device which features XOR gates. There is not an analoguos keyword in VHDL. HOwever, the functional aspect of this keyword is part of the *Warp* Compiler Option menu. For more details see the *Warp* Compiler Options Documentation.

Predefined Attributes Supported by *Warp*

Value Attributes	'Left 'Right 'High 'Low 'Length
Function Attributes (types)	'Pos 'Val 'Leftof 'Succ 'Rightof 'Pred
Function Attributes (objects)	'Left 'Right 'High 'Low 'Length
Function Attributes (signals)	'Event
Type Attributes	'Base
Range Attributes	'Range 'Reverse_range

Other user-defined attributes include: Enum-encoding, Flip-flop-type, Order_code, Part_name, Pin_numbers, Polarity, State_encoding, and State_Variable. See *Warp* documentation for details.

Number Representations

ABEL	VHDL	Radix
^b	b“ ” or “ ” or ‘ ’(default)[20]	Binary
^o	o“ ”	Octal
^d (default)	Note 21	Decimal
^h	x“ ”	Hexadecimal

Notes

20. The default number representation in ABEL is decimal. The default number representation in VHDL is binary.

21. The default number representation in VHDL is binary. Decimal representations of numbers in VHDL require the user to define a signal or variable with type integer or use an integer number and then type-convert it to bit_vector. This is easier than it sounds. In the common library directory within *Warp* there is a file called LIBINT.VHD that contains a predefined function called i2bv. This function takes an integer and returns a bit_vector. So, using a decimal number is not too difficult, but one must know that an integer must be used and then type-converted to bit_vector.

For example:

ABEL syntax	VHDL syntax	Description
^b1	’1’	binary 1
^b0	’0’	binary 0
^b101010000	”10101000”	binary 10101000
^hF	x”F”	hex F
^hF1	x”F1”	hex F1
^hAAA	x”AAA”	hex AAA
^oF0F0	o”F0F0”	octal F0F0
^d23	i2bv(23,5)	decimal 23
^d99	i2bv(99,7)	decimal 99

Polarity Conventions

VHDL does not know whether a signal name should be interpreted as an active HIGH or an active LOW. Therefore, a signal named SHIFT4 will be interpreted logically the same as one named L_SHIFT4, and as one named SHIFT4_NOT. In other words, the behavioral equations must test with the proper level and assert with the desired level.

During logic synthesis and optimization, the software may determine that by flipping the polarity of a function the logic required will be optimized.

Identifiers

VHDL is not case sensitive, so a signal named SHIFT0 is identical to one named sHIFT0.

Resets and Presets

Although there are a variety of ways to specify a reset or preset, the best method is behavioral specification. If the reset or preset is asynchronous, use the following:

Place an IF-THEN-ELSIF-ENDIF inside a process with a (CLK'EVENT and CLK='1') placed as the condition for the ELSIF. In the first IF, place your reset and preset condition test and your signal assignments. In other words, the first part of the IF contains the asynchronous or combinatorial logic description and the second part, the ELSIF, contains the clocked logic description. In the process statement use a sensitivity list that includes the clock, and reset/preset for the design. Don't forget that statements in a process are considered sequential and are only updated upon changes in signals listed in the sensitivity list. See the basic example called COUNTER2.VHD and the real-world converted design example called FLAGCTLR below.

If the reset or preset is synchronous, place the condition inside the clocked portion of the IF-THEN-ELSIF-ENDIF mentioned above and perform the appropriate signal assignments.

This methodology ensures that behavioral operation is preserved and no device specific attributes are required.

Groups

ABEL allows declaration of groups or sets. Sets are groupings of signals. For example a bus is a set of signals. To create a set of signals in VHDL use the bit_vector type declaration. To perform Boolean operations on these new sets use IF-THEN-ELSE and FOR LOOPs to index the individual elements. See the special type conversion function and the real-world examples below.

Special VHDL Type Conversion Function (Advanced)

VHDL is a strongly typed language. ABEL on the other hand is not a strongly typed language. ABEL allows a user to mix Boolean operations with relational operations on sets. To concisely convert ABEL equations that contain relational operations on sets (converted to VHDL type BIT_VECTOR) combined with Boolean operations on signals (converted to VHDL type BIT), use the following type-conversion function. All equations requiring this type-conversion function call can be modified easily with a SEARCH and REPLACE text editor.

```
------------------------------ cut here ----------------------------------

FUNCTION frbl_to_b (in1:Boolean)     RETURN bit IS
BEGIN
IF (in1=TRUE) THEN
    RETURN '1';
ELSE
    RETURN '0';END IF;
END frbl_to_b;
--  This type conversion function converts a signal or relational operation
--  result from type BOOLEAN to type BIT.  A Boolean can have a value of
--  either 'TRUE' or 'FALSE'.  A bit can have a value of either '0' or '1'.

------------------------------ cut here ----------------------------------
```

For example if you had an equation in ABEL such as:

```
ramwr = !addren & ba16 & !write & ((addr ==^h210)
    # (addr==^h212)
    # (addr==^h214)
    # (addr==^h216));
```

Where addr is a set of 16 address bits,

This equation could be converted to VHDL in at least two ways:

```
ramwr <= not addren and ba16 and not write and (fr_bl_to_b(addr =x"210")
     or fr_bl_to_b(addr=x"212")
     or fr_bl_to_b(addr=x"214")
     or fr_bl_to_b(addr=x"216"));
```

OR,

```
ramwr <= not addren and ba15 and not write and(
     (not addr(11) and not addr(10) and addr(9) and not addr(8) and not addr(7)
      and not addr(6) and not addr(5) and addr(4) and not addr(3) and not addr(2)
      not addr(1) and not addr(0))
   OR  (not addr(11) and not addr(10) and addr(9) and not addr(8) and not
       addr(7) and not addr(6) and not addr(5) and addr(4) and not addr(3) and not
       addr(2) and addr(1) and not addr(0))
   OR  (not addr(11) and not addr(10) and addr(9) and not addr(8) and not
      addr(7) and not addr(6) and not addr(5) and addr(4) and not addr(3) and
       addr(2) and not addr(1) and not addr(0))
   OR (not addr(11) and not addr(10) and addr(9) and not addr(8) and not addr(7)
      and not addr(6) and not addr(5) and addr(4) and not addr(3) and addr(2) and
      addr(1) and not addr(0)));
```

This example assumes all of the signals from the ABEL equations are converted to signals of type BIT except the set called 'addr', which is converted to type BIT_VECTOR.

This special type-conversion function has a obvious advantage and is well suited for use in converting descriptions to VHDL. By no means is it a requirement that descriptions use this function. It should be used for one reason only, to make a VHDL description concise. See the real-world design example in Appendix A called FLAGCTLR.

State Machines

See the *Warp* design examples titled intermediate TRAFFIC.VHD, intermediate DRINK.VHD and advanced TTF.VHD. See the application note titled "Describing State Machines in *Warp*2 VHDL." Also refer to pitfall numbers five and seven below.

Decoders

See the *Warp* design example titled basic DECODER.VHD and the special type-conversion function above.

Comparators

See the *Warp* design examples titled intermediate COMPARE.VHD and COMPARE2.VHD.

Counters

See the *Warp* design examples titled basic COUNTER.VHD, basic COUNTER2.VHD, intermediate COUNTER3.VHD, advanced COUNTER4.VHD, and advanced COUNTER5.VHD.

Multiplexers

Use the truth table function that is shown in the application note titled "Describing State Machines in *Warp*2 VHDL" or create a multiplexer using Boolean equations.

Shift Registers

See the *Warp* design example titled advanced SHIFTN.VHD. This example illustrates the use of the GENERATE statement.

Adders

See the *Warp* design examples titled basic ADDER1.VHD and basic ADDER2.VHD.

Repetitive Logic

The VHDL GENERATE statement lends itself to regular or repetitive logic structures. For example, n-bit registers, n-bit counters, n-bit shift registers, n-bit multiplexers, n-bit adders, and n-bit comparators may be concisely described by using the GENERATE statement. See the *Warp* design examples titled advanced SHIFTN.VHD and advanced COUNTER4.VHD.

Pitfalls

There are potential pitfalls. Some of the common mistakes made during conversion are:

1. Incorrect order of precedence of operators. For instance, all of the logical operators in VHDL have the same level of precedence. In other words, an equation that has both AND and OR operators requires parenthesis around the ANDed terms for proper logic synthesis. Refer to the cross reference and order of precedence table above.
2. Incomplete separation of clocked signals from combinatorial signals. Two simple ways to ensure proper logic synthesis of clocked signals and combinatorial signals are:
   a. Use a process for all signals, but use an IF-THEN-ELSIF-ENDIF within the process that groups all combinatorial signals under the IF, and groups all registered signals under the ELSIF. See the real-world design example in Appendix A called FLAGCTLR.
   b. Place all registered signals within a process (using a WAIT UNTIL CLOCK ='1') and place all combinatorial signals outside the process.
3. Using loops and variables outside of a process. VHDL requires that loops and variables be used inside a process. If there is more than one process, signals communicate between processes.
4. Using the incorrect mode for either output or bidirectional signals. Refer to your language reference book for a formal definition of mode.
5. Incomplete state specification for state machines. When designing a state machine, you **MUST** do one of the following:
   a. Specify all output values in each state of the machine.
   or
   b. Specify default values for all outputs at the beginning of the process.

The reason for this has to do with the way a process works. Each time a process is run (i.e., a clock event has occurred) the outputs that are specified in the particular pass through the process are updated. If a branch exists within the states of the machine that allows a pass through the process with one or more outputs not assigned a value, the logic synthesis engine either (a) assumes that the last statement for an unassigned output is valid and should be latched, or (b) that it is allowed to change with the clock. In other words, it is legal in VHDL to not specify all output values in each state of the machine, or not specify default values for all outputs at the beginning of the process, or not specify either one. If this subtle detail is overlooked, the design will compile and appear to synthesize successfully, but functional operation may not be correct. It is also possible that the logic synthesized will not be minimal. In other words, use defaults or specify the value of all outputs within each state of the machine.

6. Incorrect set or reset operation found in simulation. Polarity optimization settings used during logic synthesis and fitting can cause set and or reset operations to appear to operate inconsistently. During logic synthesis and fitting, the fitter can decide, by flipping the polarity of a function, the logic required will be minimized. This can have an adverse effect on the user selection of set or reset. (Note this pitfall only applies to 22V10s and FLASH370 where the polarity inversion is located between the output of the register and the pin.) See the polarity attribute in the *Warp* documentation for more details.

7. Failure to close, or complete, IF-THEN-ELSE-ENDIF and CASE statements. In other words, design descriptions that contain an IF must contain an ELSE, and descriptions containing a CASE-WHEN (condition), must contain a WHEN-OTHERS statement. This is required so that unnecessary implicit memory elements are not synthesized. See the application note titled "VHDL Techniques for Optimal Design Fitting" for more information.

Logic Synthesis

Proper logic synthesis is the goal of conversion. If the converted design compiles and synthesizes without errors, but the logic equations in the report file are not as expected (or simulation results are not as desired) consult the pitfalls section above. Also, consult your *Warp* - GALAXY compiler options documentation and *Warp* - NOVA user's guide. If all else fails, contact your local Cypress field application engineer.

Real-World Converted Designs

The designs in Appendix A originally were intended to fit into MACH 110s. However, due to product term and internal fanout requirements, MACH 210s were required. The designs were later converted to FLASH371s. Consult your Cypress data book for more information on the CY7C371's architecture.

Summary

Any design that has been described in Data I/O's ABEL language can be converted to VHDL. From an overall capability perspective, VHDL can be considered a superset of ABEL. Two designs documented in Appendix A were successfully converted using the cross reference tables and helpful hints contained within this application note.

Appendix A. Real-World Converted Designs

```
---------------------------------- cut here ---------------------------------

  Module FLAGCTLR
  Title 'Flag Controller 1 – Uxx_xx
  Revision 01'

  "ALGORITHM
  "
  "

  FLAGCTLR           device 'mach210a';

  "Inputs:

      R_40MHZ      pin ; "
      H_FEP_S0     pin ; "
      H_FEP_S1     pin ; "
      H_FEP_S2     pin ; "
      H_FEP_S3     pin ; "
      H_FEP_SET    pin ; "
      L_FEP_WE     pin ; "
      H_PPA_S0     pin ; "
      H_PPA_S1     pin ; "
      H_PPA_S2     pin ; "
      H_PPA_S3     pin ; "
      H_PPA_SET    pin ; "
      L_PPA_WE     pin ; "
      H_PPB_S0     pin ; "
      H_PPB_S1     pin ; "
      H_PPB_S2     pin ; "
      H_PPB_S3     pin ; "
      H_PPB_SET    pin ; "
      L_PPB_WE     pin ; "
      L_RESET      pin ; "

  "Outputs:

      H_FA0     pin  istype 'reg,buffer'; "
      H_FA1     pin  istype 'reg,buffer'; "
      H_FA2     pin  istype 'reg,buffer'; "
      H_FA3     pin  istype 'reg,buffer'; "
      H_FA4     pin  istype 'reg,buffer'; "
      H_FA5     pin  istype 'reg,buffer'; "
      H_FA6     pin  istype 'reg,buffer'; "
      H_FA7     pin  istype 'reg,buffer'; "

      H_FB0     pin  istype 'reg,buffer'; "
      H_FB1     pin  istype 'reg,buffer'; "
      H_FB2     pin  istype 'reg,buffer'; "
      H_FB3     pin  istype 'reg,buffer'; "
      H_FB4     pin  istype 'reg,buffer'; "

      H_AB0     pin  istype 'reg,buffer'; "
      H_AB1     pin  istype 'reg,buffer'; "
```

Appendix A. Real-World Converted Designs (continued)

```
    H_AB2     pin  istype 'reg,buffer'; "
    H_AB3     pin  istype 'reg,buffer'; "
    H_AB4     pin  istype 'reg,buffer'; "

Declarations

    X = .X.;
    C = .C.;
    Z = .Z.;

    FA =    [H_FA7,H_FA6,H_FA5,H_FA4,
             H_FA3,H_FA2,H_FA1,H_FA0];

    FB =    [H_FB4,H_FB3,H_FB2,H_FB1,H_FB0];
    AB =    [H_AB4,H_AB3,H_AB2,H_AB1,H_AB0];
    PPA_SEL  = [H_PPA_S3,H_PPA_S2,H_PPA_S1,H_PPA_S0];
    PPB_SEL  = [H_PPB_S3,H_PPB_S2,H_PPB_S1,H_PPB_S0];
    FEP_SEL  = [H_FEP_S3,H_FEP_S2,H_FEP_S1,H_FEP_S0];

Equations

  FA.CLK = R_40MHZ;
  FB.CLK = R_40MHZ;
  AB.CLK = R_40MHZ;

  FA.RE  = !L_RESET;
  FB.RE  = !L_RESET;
  AB.RE  = !L_RESET;


  H_FA0.T = (!H_FA0.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h0)
           #  H_FA0.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h0)
           # !H_FA0.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h0)
           #  H_FA0.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h0));

  H_FA1.T = (!H_FA1.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h1)
           #  H_FA1.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h1)
           # !H_FA1.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h1)
           #  H_FA1.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h1));


  H_FA2.T = (!H_FA2.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h2)
           #  H_FA2.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h2)
           # !H_FA2.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h2)
           #  H_FA2.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h2));

  H_FA3.T = (!H_FA3.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h3)
           #  H_FA3.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h3)
           # !H_FA3.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h3)
           #  H_FA3.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h3));


  H_FA4.T = (!H_FA4.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h4)
           #  H_FA4.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h4)
           # !H_FA4.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h4)
```

Appendix A. Real-World Converted Designs (continued)

```
          #  H_FA4.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h4));

H_FA5.T = (!H_FA5.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h5)
          #  H_FA5.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h5)
          # !H_FA5.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h5)
          #  H_FA5.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h5));


H_FA6.T = (!H_FA6.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h6)
          #  H_FA6.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h6)
          # !H_FA6.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h6)
          #  H_FA6.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h6));

H_FA7.T = (!H_FA7.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h7)
          #  H_FA7.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h7)
          # !H_FA7.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h7)
          #  H_FA7.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h7));


H_FB0.T = (!H_FB0.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h0)
          #  H_FB0.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h0)
          # !H_FB0.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h8)
          #  H_FB0.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h8));

H_FB1.T = (!H_FB1.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h1)
          #  H_FB1.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h1)
          # !H_FB1.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h9)
          #  H_FB1.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^h9));


H_FB2.T = (!H_FB2.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h2)
          #  H_FB2.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h2)
          # !H_FB2.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^ha)
          #  H_FB2.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^ha));

H_FB3.T = (!H_FB3.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h3)
          #  H_FB3.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h3)
          # !H_FB3.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^hb)
          #  H_FB3.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^hb));


H_AB0.T = (!H_AB0.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h8)
          #  H_AB0.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h8)
          # !H_AB0.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h8)
          #  H_AB0.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h8));

H_AB1.T = (!H_AB1.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h9)
          #  H_AB1.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h9)
          # !H_AB1.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h9)
          #  H_AB1.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^h9));


H_AB2.T = (!H_AB2.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^ha)
          #  H_AB2.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^ha)
          # !H_AB2.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^ha)
```

Appendix A. Real-World Converted Designs (continued)

```
            #  H_AB2.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^ha));

  H_AB3.T = (!H_AB3.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^hb)
            #  H_AB3.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^hb)
            # !H_AB3.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^hb)
            #  H_AB3.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^hb));


  H_FB4.T = (!H_FB4.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h4)
            #  H_FB4.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^h4)
            # !H_FB4.Q &  H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^hc)
            #  H_FB4.Q & !H_FEP_SET & !L_FEP_WE & (FEP_SEL == ^hc));


  H_AB4.T = (!H_AB4.Q &  H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^hc)
            #  H_AB4.Q & !H_PPB_SET & !L_PPB_WE & (PPB_SEL == ^hc)
            # !H_AB4.Q &  H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^hc)
            #  H_AB4.Q & !H_PPA_SET & !L_PPA_WE & (PPA_SEL == ^hc));

test_vectors ([R_40MHZ,L_RESET,
               L_FEP_WE, FEP_SEL, H_FEP_SET,
               L_PPA_WE, PPA_SEL, H_PPA_SET,
               L_PPB_WE, PPB_SEL, H_PPB_SET]
            -> [H_FA7, H_FA6, H_FA5, H_FA4, H_FA3, H_FA2, H_FA1, H_FA0,
                H_FB4, H_FB3, H_FB2, H_FB1, H_FB0,
                H_AB4, H_AB3, H_AB2, H_AB1, H_AB0])
[C,1,1,^h0,0,1,^h1,0,1,^h0,0]->[X,X,X,X,X,X,X,X, X,X,X,X,X, X,X,X,X,X];
[C,1,0,^h0,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h1,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h2,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h3,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h4,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h5,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h6,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h7,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h8,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h9,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^hA,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^hB,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^hC,0,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h0,1,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,0,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h1,1,1,^h1,0,1,^h0,0]->[0,0,0,0,0,0,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h2,1,1,^h1,0,1,^h0,0]->[0,0,0,0,0,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h3,1,1,^h1,0,1,^h0,0]->[0,0,0,0,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h4,1,1,^h1,0,1,^h0,0]->[0,0,0,1,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h5,1,1,^h1,0,1,^h0,0]->[0,0,1,1,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h6,1,1,^h1,0,1,^h0,0]->[0,1,1,1,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h7,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 0,0,0,0,0, 0,0,0,0,0];
[C,1,0,^h8,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 0,0,0,0,1, 0,0,0,0,0];
[C,1,0,^h9,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 0,0,0,1,1, 0,0,0,0,0];
[C,1,0,^hA,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 0,0,1,1,1, 0,0,0,0,0];
[C,1,0,^hB,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 0,1,1,1,1, 0,0,0,0,0];
[C,1,0,^hC,1,1,^h1,0,1,^h0,0]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 0,0,0,0,0];
[C,1,1,^h7,1,0,^h0,0,1,^h0,0]->[1,1,1,1,1,1,1,0, 1,1,1,1,1, 0,0,0,0,0];
```

Appendix A. Real-World Converted Designs (continued)

```
  [C,1,1,^h7,1,0,^h1,0,1,^h0,0]->[1,1,1,1,1,1,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h2,0,1,^h0,0]->[1,1,1,1,1,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h3,0,1,^h0,0]->[1,1,1,1,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h4,0,1,^h0,0]->[1,1,1,0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h5,0,1,^h0,0]->[1,1,0,0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h6,0,1,^h0,0]->[1,0,0,0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^h7,0,1,^h0,0]->[0,0,0,0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h0,0]->[0,0,0,0,0,0,0,0, 1,1,1,1,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h1,0]->[0,0,0,0,0,0,0,0, 1,1,1,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h2,0]->[0,0,0,0,0,0,0,0, 1,1,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h3,0]->[0,0,0,0,0,0,0,0, 1,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h4,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h8,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h9,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hA,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hB,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hC,0]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hC,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 1,0,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hB,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 1,1,0,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^hA,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 1,1,1,0,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h9,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 1,1,1,1,0];
  [C,1,1,^h7,1,1,^h7,0,0,^h8,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,0, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h7,1,0,^h4,1]->[0,0,0,0,0,0,0,0, 1,0,0,0,0, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h7,1,0,^h3,1]->[0,0,0,0,0,0,0,0, 1,1,0,0,0, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h7,1,0,^h2,1]->[0,0,0,0,0,0,0,0, 1,1,1,0,0, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h7,1,0,^h1,1]->[0,0,0,0,0,0,0,0, 1,1,1,1,0, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h7,1,0,^h0,1]->[0,0,0,0,0,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h7,1,1,^h0,1]->[1,0,0,0,0,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h6,1,1,^h0,1]->[1,1,0,0,0,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h5,1,1,^h0,1]->[1,1,1,0,0,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h4,1,1,^h0,1]->[1,1,1,1,0,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h3,1,1,^h0,1]->[1,1,1,1,1,0,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h2,1,1,^h0,1]->[1,1,1,1,1,1,0,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h1,1,1,^h0,1]->[1,1,1,1,1,1,1,0, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h0,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,0,^h8,0,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,0];
  [C,1,1,^h7,1,0,^h9,0,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,0,0];
  [C,1,1,^h7,1,0,^hA,0,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,0,0,0];
  [C,1,1,^h7,1,0,^hB,0,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,0,0,0,0];
  [C,1,1,^h7,1,0,^hC,0,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 0,0,0,0,0];
  [C,1,1,^h7,1,0,^hC,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,0,0,0,0];
  [C,1,1,^h7,1,0,^hB,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,0,0,0];
  [C,1,1,^h7,1,0,^hA,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,0,0];
  [C,1,1,^h7,1,0,^h9,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,0];
  [C,1,1,^h7,1,0,^h8,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1];
  [C,1,1,^h7,1,1,^h0,1,1,^h0,1]->[1,1,1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1];

END FLAGCTLR;

----------------------------------- cut here ---------------------------------

--  Converted to IEEE 1076 VHDL

--  Module FLAGCTLR
```

Appendix A. Real-World Converted Designs (continued)

```
--  Title 'Flag Controller 1 - Uxx_xx
--  Revision 01'

use work.cypress.all;
use work.rtlpkg.all;
use work.int_math.all;

ENTITY FLAGCTLR IS PORT(
  R_40MHZ,H_FEP_SET,L_FEP_WE,H_PPA_SET,
  L_PPA_WE,H_PPB_SET,L_PPB_WE,L_RESET   : IN BIT;
  PPA_SEL,PPB_SEL,FEP_SEL               : IN BIT_VECTOR (3 downto 0);
  FA                                    : INOUT BIT_VECTOR (7 downto 0);
  FB,AB                                 : INOUT BIT_VECTOR (4 downto 0));
attribute part_name of eventflg: entity is "c371";
END FLAGCTLR;

ARCHITECTURE CONVERTED_ABL OF FLAGCTLR IS

FUNCTION frbl_to_b(in1:Boolean)     RETURN bit IS
BEGIN
IF (in1=true) THEN
     RETURN '1';
ELSE
     RETURN '0';
END IF;
END frbl_to_b;
--  This type conversion function converts a signal or relational operation
--  result from type BOOLEAN to type BIT.  A Boolean can have a value of either
-- 'TRUE' or'FALSE'.  A bit can have a value of either '0' or '1'.

BEGIN
PROCESS (R_40MHZ, L_RESET)
BEGIN

IF (L_RESET ='0') THEN

      FOR i IN 0 TO 4 LOOP
           FA(i) <= '0'; FB(i) <= '0'; AB(i) <= '0';
      END LOOP;

      FOR i IN 5 TO 7 LOOP
           FA(i) <= '0';
      END LOOP;

ELSIF (R_40MHZ'EVENT AND R_40MHZ ='1') THEN

  FA(0) <= FA(0) XOR
         ((NOT FA(0) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"0"))
      OR (FA(0) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"0"))
      OR (NOT FA(0) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"0"))
      OR (FA(0) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"0")));

  FA(1) <= FA(1) XOR
         ((NOT FA(1) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"1"))
      OR (FA(1) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"1"))
```

Appendix A. Real-World Converted Designs (continued)

```
    OR (NOT FA(1) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"1"))
    OR (FA(1) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"1")));

FA(2) <= FA(2) XOR
       ((NOT FA(2) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"2"))
    OR (FA(2) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"2"))
    OR (NOT FA(2) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"2"))
    OR (FA(2) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"2")));

FA(3) <= FA(3) XOR
       ((NOT FA(3) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"3"))
    OR (FA(3) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"3"))
    OR (NOT FA(3) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"3"))
    OR (FA(3) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"3")));

FA(4) <= FA(4) XOR
       ((NOT FA(4) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"4"))
    OR (FA(4) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"4"))
    OR (NOT FA(4) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"4"))
    OR (FA(4) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"4")));

FA(5) <= FA(5) XOR
       ((NOT FA(5) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"5"))
    OR (FA(5) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"5"))
    OR (NOT FA(5) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"5"))
    OR (FA(5) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"5")));

FA(6) <= FA(6) XOR
       ((NOT FA(6) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"6"))
    OR (FA(6) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"6"))
    OR (NOT FA(6) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"6"))
    OR (FA(6) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"6")));

FA(7) <= FA(7) XOR
       ((NOT FA(7) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"7"))
    OR (FA(7) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"7"))
    OR (NOT FA(7) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"7"))
    OR (FA(7) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"7")));

FB(0) <= FB(0) XOR
       ((NOT FB(0) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"0"))
    OR (FB(0) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"0"))
    OR (NOT FB(0) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"8"))
    OR (FB(0) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"8")));

FB(1) <= FB(1) XOR
       ((NOT FB(1) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"1"))
    OR (FB(1) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"1"))
    OR (NOT FB(1) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"9"))
    OR (FB(1) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"9")));

FB(2) <= FB(2) XOR
       ((NOT FB(2) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"2"))
    OR (FB(2) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"2"))
    OR (NOT FB(2) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"A"))
```

Appendix A. Real-World Converted Designs (continued)

```
      OR (FB(2) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"A")));

  FB(3) <= FB(3) XOR
         ((NOT FB(3) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"3"))
      OR (FB(3) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"3"))
      OR (NOT FB(3) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"B"))
      OR (FB(3) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"B")));

  AB(0) <= AB(0) XOR
         ((NOT AB(0) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"8"))
      OR (AB(0) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"8"))
      OR (NOT AB(0) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"8"))
      OR (AB(0) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"8")));

  AB(1) <= AB(1) XOR
         ((NOT AB(1) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"9"))
      OR (AB(1) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"9"))
      OR (NOT AB(1) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"9"))
      OR (AB(1) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"9")));

  AB(2) <= AB(2) XOR
         ((NOT AB(2) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"A"))
      OR (AB(2) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"A"))
      OR (NOT AB(2) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"A"))
      OR (AB(2) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"A")));

  AB(3) <= AB(3) XOR
         ((NOT AB(3) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"B"))
      OR (AB(3) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"B"))
      OR (NOT AB(3) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"B"))
      OR (AB(3) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"B")));

  FB(4) <= FB(4) XOR
         ((NOT FB(4) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"4"))
      OR (FB(4) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"4"))
      OR (NOT FB(4) AND H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"C"))
      OR (FB(4) AND NOT H_FEP_SET AND NOT L_FEP_WE AND frbl_to_b(FEP_SEL=x"C")));

  AB(4) <= AB(4) XOR
         ((NOT AB(4) AND H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"C"))
      OR (AB(4) AND NOT H_PPB_SET AND NOT L_PPB_WE AND frbl_to_b(PPB_SEL=x"C"))
      OR (NOT AB(4) AND H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"C"))
      OR (AB(4) AND NOT H_PPA_SET AND NOT L_PPA_WE AND frbl_to_b(PPA_SEL=x"C")));

END IF;
END PROCESS;
END CONVERTED_ABL;

------------------------------------ cut here ------------------------------------

Module CONVERTER
Title 'Converter
Revision 01'

" This device converts a 32-bit floating point word from one format to another.
```

Appendix A. Real-World Converted Designs (continued)

```
CONVERTER                    device       'MACH210A';

"Control Inputs:

CLK           PIN 35;" Clock
OE            PIN 10;" Low Active Output Enable
WE            PIN 11;" Low Active Write Enable
MODE          PIN 13;" Shift Mode

"Data I/O BITS:

D31           PIN 43   ISTYPE    'REG,BUFFER';
D30           PIN 42   ISTYPE    'REG,BUFFER';
D29           PIN 41   ISTYPE    'REG,BUFFER';
D28           PIN 40   ISTYPE    'REG,BUFFER';
D27           PIN 39   ISTYPE    'REG,BUFFER';
D26           PIN 38   ISTYPE    'REG,BUFFER';
D25           PIN 37   ISTYPE    'REG,BUFFER';
D24           PIN 36   ISTYPE    'REG,BUFFER';
D23           PIN 31   ISTYPE    'REG,BUFFER';
D22           PIN 30   ISTYPE    'REG,BUFFER';
D21           PIN 29   ISTYPE    'REG,BUFFER';
D20           PIN 28   ISTYPE    'REG,BUFFER';
D19           PIN 27   ISTYPE    'REG,BUFFER';
D18           PIN 26   ISTYPE    'REG,BUFFER';
D17           PIN 25   ISTYPE    'REG,BUFFER';
D16           PIN 24   ISTYPE    'REG,BUFFER';
D15           PIN 21   ISTYPE    'REG,BUFFER';
D14           PIN 20   ISTYPE    'REG,BUFFER';
D13           PIN 19   ISTYPE    'REG,BUFFER';
D12           PIN 18   ISTYPE    'REG,BUFFER';
D11           PIN 17   ISTYPE    'REG,BUFFER';
D10           PIN 16   ISTYPE    'REG,BUFFER';
D09           PIN 15   ISTYPE    'REG,BUFFER';
D08           PIN 14   ISTYPE    'REG,BUFFER';
D07           PIN  9   ISTYPE    'REG,BUFFER';
D06           PIN  8   ISTYPE    'REG,BUFFER';
D05           PIN  7   ISTYPE    'REG,BUFFER';
D04           PIN  6   ISTYPE    'REG,BUFFER';
D03           PIN  5   ISTYPE    'REG,BUFFER';
D02           PIN  4   ISTYPE    'REG,BUFFER';
D01           PIN  3   ISTYPE    'REG,BUFFER';
D00           PIN  2   ISTYPE    'REG,BUFFER';

H,L,C,Z,X  =  1,0,.C.,.Z.,.X.;

DIN        =  [D19.PIN,D18.PIN,D17.PIN,D16.PIN,
               D15.PIN,D14.PIN,D13.PIN,D12.PIN,
               D11.PIN,D10.PIN,D09.PIN,D08.PIN,
               D07.PIN,D06.PIN,D05.PIN,D04.PIN,
               D03.PIN,D02.PIN,D01.PIN,D00.PIN];

DOUT       =  [D31,D30,D29,D28,D27,D26,D25,D24,
               D23,D22,D21,D20,D19,D18,D17,D16,
```

Appendix A. Real-World Converted Designs (continued)

```
                D15,D14,D13,D12,D11,D10,D09,D08,
                D07,D06,D05,D04,D03,D02,D01,D00];

DBIDI       =   [D19,D18,D17,D16,D15,D14,D13,D12,
                D11,D10,D09,D08,D07,D06,D05,D04,
                D03,D02,D01,D00];

DOUTFB      =   [L,L,L,L,L,L,L,L,L,L,L,L,
                D19.FB,D18.FB,D17.FB,D16.FB,
                D15.FB,D14.FB,D13.FB,D12.FB,
                D11.FB,D10.FB,D09.FB,D08.FB,
                D07.FB,D06.FB,D05.FB,D04.FB,
                D03.FB,D02.FB,D01.FB,D00.FB];

M0_SHIFT6   =   !WE & !MODE &
                ((D19.PIN == H) # (D18.PIN == H) # (D17.PIN == H));

M0_SHIFT3   =   !WE & !MODE &
                (((D19.PIN == L) & (D18.PIN == L) & (D17.PIN == L)) &
                ((D16.PIN == H) # (D15.PIN == H) # (D14.PIN == H)));

SHIFT0      =   !WE &
                (((D19.PIN == L) & (D18.PIN == L) & (D17.PIN == L)) &
                ((D16.PIN == L) & (D15.PIN == L) & (D14.PIN == L)));

M1_SHIFT6   =   !WE & MODE &
                ((D19.PIN == H) # (D18.PIN == H));

M1_SHIFT4   =   !WE & MODE &
                (((D19.PIN == L) & (D18.PIN == L)) &
                ((D17.PIN == H) # (D16.PIN == H)));

M1_SHIFT2   =   !WE & MODE &
                (((D19.PIN == L) & (D18.PIN == L) &
                (D17.PIN == L) & (D16.PIN == L)) &
                ((D15.PIN == H) # (D14.PIN == H)));

EQUATIONS

DOUT.CLK    =   CLK;

DOUT.OE     =   !OE;
DOUT        :=  M0_SHIFT6  &
                          [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,H,L,D19.PIN,D18.PIN,
                               D17.PIN,D16.PIN,D15.PIN,D14.PIN,D13.PIN,D12.PIN,
                               D11.PIN,D10.PIN,D09.PIN,D08.PIN,D07.PIN,D06.PIN]

             #

                M0_SHIFT3  &
                          [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,H,D16.PIN,D15.PIN,
                               D14.PIN,D13.PIN,D12.PIN,D11.PIN,D10.PIN,D09.PIN,
                               D08.PIN,D07.PIN,D06.PIN,D05.PIN,D04.PIN,D03.PIN]

             #

                SHIFT0     &
                          [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,D13.PIN,D12.PIN,
```

Appendix A. Real-World Converted Designs (continued)

```
                        D11.PIN,D10.PIN,D09.PIN,D08.PIN,D07.PIN,D06.PIN,
                        D05.PIN,D04.PIN,D03.PIN,D02.PIN,D01.PIN,D00.PIN]

            #

              M1_SHIFT6  &
                        [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,H,H,D19.PIN,D18.PIN,
                        D17.PIN,D16.PIN,D15.PIN,D14.PIN,D13.PIN,D12.PIN,
                        D11.PIN,D10.PIN,D09.PIN,D08.PIN,D07.PIN,D06.PIN]

            #

              M1_SHIFT4  &
                        [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,H,L,D17.PIN,D16.PIN,
                        D15.PIN,D14.PIN,D13.PIN,D12.PIN,D11.PIN,D10.PIN,
                        D09.PIN,D08.PIN,D07.PIN,D06.PIN,D05.PIN,D04.PIN]

            #

              M1_SHIFT2  &
                        [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,H,D15.PIN,D14.PIN,
                        D13.PIN,D12.PIN,D11.PIN,D10.PIN,D09.PIN,D08.PIN,
                        D07.PIN,D06.PIN,D05.PIN,D04.PIN,D03.PIN,D02.PIN]

            #

              WE & DOUTFB;

Test_Vectors

([CLK,OE,WE,MODE,              DBIDI]          ->      DOUT)
 [C,H,L,X,                    ^b00000000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000000101101110110;"Read,Shift0
 [C,H,L,L,                    ^b00000110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000100110101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00001010101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010100101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00010010101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011001010101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00011010101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011101010101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00010110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011010110101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00001110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010111010101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00011110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011111010101101011;"Read,Shift3/Mode0
 [C,H,L,L,                    ^b00111110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010001111110101101;"Read,Shift6/Mode0
 [C,H,L,L,                    ^b01011110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010010111110101101;"Read,Shift6/Mode0
 [C,H,L,L,                    ^b10011110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010100111110101101;"Read,Shift6/Mode0
 [C,H,L,L,                    ^b10100110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010101001101101101;"Read,Shift6/Mode0
```

Appendix A. Real-World Converted Designs (continued)

```
 [C,H,L,L,                      ^b01100110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000100110011010110 1;"Read,Shift6/Mode0
 [C,H,L,L,                      ^b11000110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000101100011010110 1;"Read,Shift6/Mode0
 [C,H,L,L,                      ^b11111110101101011011] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000101111111010110 1;"Read,Shift6/Mode0
 [C,H,L,H,                      ^b00000100101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000010100101101110 1;"Read,Shift2/Mode1
 [C,H,L,H,                      ^b00001000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011000101101110 1;"Read,Shift2/Mode1
 [C,H,L,H,                      ^b00001100101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011100101101110 1;"Read,Shift2/Mode1
 [C,H,L,H,                      ^b00010000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000100100001011011 1;"Read,Shift4/Mode1
 [C,H,L,H,                      ^b00100000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000101000001011011 1;"Read,Shift4/Mode1
 [C,H,L,H,                      ^b00110000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000101100001011011 1;"Read,Shift4/Mode1
 [C,H,L,H,                      ^b00111100101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000101111001011011 1;"Read,Shift4/Mode1
 [C,H,L,H,                      ^b10000000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011100000000101101;"Read,Shift6/Mode1
 [C,H,L,H,                      ^b01000000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011010000000101101;"Read,Shift6/Mode1
 [C,H,L,H,                      ^b11000000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011110000000101101;"Read,Shift6/Mode1
 [C,H,L,H,                      ^b11010100101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011110101000101101;"Read,Shift6/Mode1
 [C,H,L,H,                      ^b11101000101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011111010000101101;"Read,Shift6/Mode1
 [C,H,L,H,                      ^b11111100101101110110] -> Z;"Write
 [C,L,H,X,Z]    -> ^b00000000000000000011111111000101101;"Read,Shift6/Mode1

End CONVERTER;

----------------------------------- cut here ---------------------------------

-- CONVERTED TO IEEE 1076 VHDL

-- Module CONVERTER
-- Title 'Converter
-- Revision 01'

-- This device converts a 32-bit floating point word from one format to another.

-- Control Inputs

use work.cypress.all;
use work.rtlpkg.all;
use work.int_math.all;

ENTITY CONVERTER IS PORT(
CLK,OE,WE,MODE                                  : IN BIT;
D                                               : INOUT X01Z_VECTOR (0 TO 31));
```

Appendix A. Real-World Converted Designs (continued)

```
attribute part_name of CONVERTER: entity is "c371";
END CONVERTER;

ARCHITECTURE CONVERTED_ABL OF CONVERTER IS
SIGNAL SHIFT2_TMP, SHIFT1_TMP, SHIFT0_TMP, SHIFT2,
       SHIFT1, SHIFT0, M0_SHIFT6, M0_SHIFT3,
       M_SHIFT0, M1_SHIFT6, M1_SHIFT4, M1_SHIFT2     : BIT;
SIGNAL D_TMP, D_FB                                   : BIT_VECTOR (0 TO 31);
BEGIN

P1: PROCESS
 BEGIN
  WAIT UNTIL CLK ='1';

   FOR i IN 0 TO 13 LOOP
   D_TMP(i) <=(D_FB(i+6) AND     SHIFT2 AND     SHIFT1 AND NOT SHIFT0)
           OR (D_FB(i+4) AND     SHIFT2 AND NOT SHIFT1 AND NOT SHIFT0)
           OR (D_FB(i+3) AND NOT SHIFT2 AND     SHIFT1 AND     SHIFT0)
           OR (D_FB(i+2) AND NOT SHIFT2 AND     SHIFT1 AND NOT SHIFT0)
           OR (D_FB(i+0) AND NOT SHIFT2 AND NOT SHIFT1 AND NOT SHIFT0)
           OR (WE AND D_TMP(i));
   END LOOP;

   D_TMP(14) <= ('0' AND M0_SHIFT6) OR ('1' AND M0_SHIFT3)
           OR ('0' AND  M_SHIFT0) OR ('1' AND M1_SHIFT6)
           OR ('0' AND M1_SHIFT4) OR ('1' AND M1_SHIFT2)
           OR (WE AND D_TMP(14));

   D_TMP(15) <= ('1' AND M0_SHIFT6) OR ('0' AND M0_SHIFT3)
           OR ('0' AND  M_SHIFT0) OR ('1' AND M1_SHIFT6)
           OR ('1' AND M1_SHIFT4) OR ('0' AND M1_SHIFT2)
           OR (WE AND D_TMP(15));

   FOR i IN 16 TO 28 LOOP
   D_TMP(i) <='0';
   END LOOP;

END PROCESS P1;

M0_SHIFT6  <= (NOT WE AND MODE) AND
            (D_FB(19) OR D_FB(18) OR D_FB(17));

M0_SHIFT3  <= (NOT WE AND NOT MODE AND NOT D_FB(19) AND NOT D_FB(18)
             AND NOT D_FB(17)) AND (D_FB(16) OR D_FB(15) OR D_FB(14));

M_SHIFT0   <= NOT WE AND NOT D_FB(19) AND NOT D_FB(18) AND NOT
            D_FB(17) AND NOT D_FB(16) AND NOT D_FB(15) AND NOT D_FB(14);

M1_SHIFT6  <= (NOT WE AND MODE) AND (D_FB(19) OR D_FB(18));

M1_SHIFT4  <= (NOT WE AND MODE AND NOT D_FB(19) AND NOT D_FB(18))
             AND (D_FB(17) OR D_FB(16));

M1_SHIFT2  <= (NOT WE AND MODE AND NOT D_FB(19) AND NOT D_FB(18) AND
             NOT D_FB(17) AND NOT D_FB(16)) AND (D_FB(15) OR D_FB(14));
```

Appendix A. Real-World Converted Designs (continued)

```
SHIFT2_TMP <= M0_SHIFT6
              OR M1_SHIFT6
              OR M1_SHIFT4;

SHIFT1_TMP <= M0_SHIFT6
              OR M0_SHIFT3
              OR M1_SHIFT6
              OR M1_SHIFT2;

SHIFT0_TMP <= M0_SHIFT3;

-- Mapping for the Bidirectional buffers
-- D_TMP is the internal signal which drives the output buffer
-- OE    is the signal for output enable (active high)
-- D     is the pin name, matches name in port assignment
-- D_FB  is the signal from pin that feeds back and drives the internal
--       structure
G1: FOR i IN 0 TO 28 GENERATE
      B1:BUFOE PORT MAP(D_TMP(i), OE,D(i), D_FB(i));
END GENERATE;

B2: BUF PORT MAP(SHIFT0_TMP,SHIFT0); -- Forces logic synthesis to ”split
B3: BUF PORT MAP(SHIFT1_TMP,SHIFT1); -- sums” into SHIFT codes that are
B4: BUF PORT MAP(SHIFT2_TMP,SHIFT2); -- encoded and placed on outputs D29–31

B5: BUFOE PORT MAP(SHIFT0, OE, D(29), open);
B6: BUFOE PORT MAP(SHIFT1, OE, D(30), open);
B7: BUFOE PORT MAP(SHIFT2, OE, D(31), open);

END CONVERTED_ABL;

---------------------------------- cut here ----------------------------------
```