Hints, Tips and Tricks for using XABEL with Xilinx M1.4 Design and Implementation Tools

XAPP109 February, 1998 Version 1.0 Application Note

Summary

This application note summarizes the issues and design techniques specific to the Xilinx ABEL Interface, version M1.4.

Xilinx Family

All

Overview

Products

XABEL included only in Foundation product

XABEL is included in all variations of the Foundation F1.4 product (Base or Standard, with or without HDL). When used in Foundation, XABEL design entry is supported by the HDL Editor, and design development is tightly integrated into the Foundation Project Manager.

XABEL is not included in any Alliance products. A separate XABEL Interface package is available for download from the Xilinx web site which can be used to develop ABEL modules to be included as macros in schematic-based design. (See ABEL modules for Alliance designs below.)

No workstation version of XABEL software

The ABEL6 compiler from Synario is not available for any UNIX workstations. ABEL modules to be included in design prepared on workstation-based schematic capture systems should be compiled on a PC. The EDIF netlist for each ABEL module can be transferred to the workstation and included in the top-level design.

The older ABEL5 compiler that was available with XACT version 5 was available in a UNIX compatible form. XNF netlists (for FPGAs) or Plusasm equation files (for CPLDs) produced by that interface are still readable by XACT M1.4 software. However, the ABEL5 software cannot be integrated into the XACT-M1 system and cannot be used to produce EDIF netlists.

Documented in Foundation on-line help

Because XABEL is sold only in the Foundation product, all documentation supporting the XABEL interface is included in the Foundation on-line help system. This documentation includes ABEL design techniques for FPGAs and CPLDs, and an ABEL-HDL Language Reference. The on-line help document can be useful for preparing ABEL modules for use in an Alliance design. The on-line help files can be downloaded separately from the Xilinx web site at www.xilinx.com → Support → File Download → Software Help → Foundation; filename is fnd_help.zip.

Capabilities

Supports all families

All XABEL designs are compiled into EDIF netlists consisting of architecture-independent Xilinx Unified Library components, which can be read and incorporated into designs targeting any Xilinx device families.

Creates macro modules and stand-alone designs

The XABEL Interface can optionally add pad components to all pins declared in the ABEL design, thus generating a top-level EDIF netlist that can be read directly by the Xilinx Design Manager. Otherwise, pad components can be omitted to generate a netlist used to define the logic of a macro symbol embedded in a schematic design. Only simple input and output buffers (including 3-state outputs) are supported by XABEL. So any FPGA designs requiring registered input or output pads should be based on top-level schematics containing the desired I/O symbols.

Based on Synario ABEL 6 with hierarchy

The ABEL compiler used in the XABEL Interface (version M1.4) is the ABEL version 6.2 engine from Synario Design Automation, formerly a subsidiary of Data I/O, and now a subsidiary of Minc.

The ABEL6 compiler has several improvements over ABEL5, including the ability to combine multiple ABEL modules into a hierarchical design.

EDIF netlists are encrypted

The XABEL Interface uses a special version of the Synario ABEL which does not require a parallel port security key. Because of this, the EDIF netlists produced by XABEL are encrypted so they cannot be used to target any non-Xilinx technologies. Xilinx cannot provide the non-encrypted version of any EDIF netlist produced by XABEL to any of our customers.

Even though the netlist is encrypted, functional simulation of the XABEL design is supported at 3 levels:

1. Test vectors embedded in the ABEL source design are automatically simulated during compilation by the built-in ABEL simulator (BLIFSIM), producing a tabular report file.
2. The Foundation gate-level simulator can directly simulate the encrypted EDIF netlists.
3. A simulation netlist can be generated by the Xilinx implementation software after the top-level design is read and translated. The simulation netlist consists of Xilinx simulation primitives (simprims) that can only be used for simulation modeling and not for design entry.

Installation requirements

Local hard-drive only

The ABEL compiler software does not run reliably when installed to a network directory. Therefore, XABEL should only be installed onto the local hard disk of the PC.

Same directory as XACT-M1

The XABEL interface is dependent on XACT-M1 implementation (core) software system. XABEL must therefore reside in the same directory as the XACT-M1 implementation (core) software. This is the directory referenced by the XILINX environment variable. This requirement applies whether XABEL is used with Foundation F1.4 software or with Alliance M1.4 software.

Because XABEL must be installed to a local hard drive, the entire XACT-M1 implementation software must also be installed on a local hard drive if XABEL is to be used.

No other ABEL products tolerated (bug)

The license mechanism used by the ABEL compiler software relies on a product entry made in the Windows registry during XABEL installation. This registry entry is adversely altered if a different version of the ABEL compiler software is installed by a different (non-Xilinx) product, including the Synario and Workview Office design systems. Similarly, the installation of XABEL may cause a non-Xilinx version of ABEL to stop running. If you have a non-Xilinx ABEL product installed on your PC, you should save its registry keys in a .reg file before installing XABEL. (See OLE server and registry issues below.)

ABEL design techniques

Name of module must match filename.

The module name specified in each ABEL file must match the name of the file. For example, if your file is named myfile.abl, it must contain "module myfile" at the top.

If you are creating a hierarchical ABEL design, each module of the design must be contained in a separate file and each filename must match the contained module name.

Pin vs internal feedback

In a top-level ABEL design, if you define an output pin and also use the name of that output pin as the name of a signal source in another equation, the interpretation of that signal name reference may be ambiguous.

For example, your ABEL design may contain:

```
myoutput pin;
equations
myoutput = a & b;
y = myoutput & x;
```

The ABEL compiler may interpret “myoutput” in the last equation as the internal feedback from the logic expression “a & b”. Alternatively, it may implement myoutput as a bidirectional pin and interpret the reference to myoutput in the last equation as the pin input. Generally, the latter interpretation will occur if you also define a 3-state output enable condition for myoutput, as in:

```
myoutput.oe = my_oe;
```

To prevent ambiguity, you should specify the “.FB” extension for internal feedback (before 3-state control), or the “.PIN” extension for bidirectional pin input, wherever an output pin name is used as an input name. The equation for “y” should therefore be written as either:

```
y = myoutput.FB & x;
```

or

```
y = myoutput.PIN & x;
```

Feedback interpretation in XABEL-M1 different than in pre-M1 versions.

The interpretation of feedback is different in XACT-M1 than in earlier versions of the XABEL Interface, such as in XACT version 6 (or earlier) or in the XABEL-CPLD product. (This only applies to top-level ABEL designs for CPLD.) In pre-M1 versions, references to a feedback signal were always interpreted as internal feedback unless the “.PIN” extension was specified. In XABEL-M1 (using the EDIF interface), references to feedbacks from outputs with 3-state enable conditions are interpreted as pin feedback unless the “.FB” extension is specified. For example, in the following case, the pre-M1 version of XABEL would interpret the feedback from myoutput in line 4 as internal feedback (before the 3-state control); XABEL-M1 interprets myoutput in line 4 as input from a bidirectional pin (after 3-state control).

```
myoutput pin;
equations
myoutput = a & b;
y = myoutput & x;  // line 4
myoutput.oe = my_oe;
```

Note: The obsolete PLUSASM flow provided with XABEL-M1 interprets feedback the same way as pre-M1 XABEL; i.e., it assumes internal feedback unless “.PIN” is specified.

Register initial states

for FPGA use .AP/.AR

In designs targeting FPGA devices, the initial (power-on) state of each flip-flop is determined by whether there is an asynchronous set or reset condition specified. The initial state always coincides with the specified asynchronous condition. Registers with an asynchronous reset (.AR equation) will initialize to 0. Registers with an asynchronous preset (.AP equation) will initialize to 1. Registers with neither asynchronous reset or preset will, by default, initialize to 0. If you just want to specify an initial state but do not want to actively reset or preset the flip-flop, just define an .AP or .AR equation with the value of zero, as in:

```
my_reg.AP = 0;
```

In FPGA designs, it is illegal to specify both asynchronous reset and preset for the same registered signal, as the FPGA architectures do not physically support that.

for CPLD use INIT

In designs targeting CPLD devices, the initial (power-on) state of each flip-flop is determined by the INIT attribute specified using a Xilinx property statement. Unlike FPGA devices, initial states in CPLDs can be set independently of any asynchronous set or reset conditions applied to the registers. Also, CPLD devices support both asynchronous reset and preset on each register, so you can specify both an .AR and .AP equation for the same registered signal.

To specify the initial state of a register in a CPLD design, use the following declaration in the ABEL design:

```
xilinx property 'INIT=state
signal_list...';
```

where state is either “R” for reset (0) or “S” for set (1). By default, all flip-flops in a CPLD design initialize to 0.

INITs reversed for registers with asynchronous preset (bug)

In the M1.3 and M1.4 software, there is a bug which causes the initial state to be reversed for some flip-flops in CPLD designs. If your design contains a register with an asynchronous preset (.AP) equation, but without an asynchronous reset (.AR) equation, the register is implemented in negative-logic form. That is, the register is implemented using a flip-flop primitive with asynchronous reset and its D-input and Q-output are inverted. This effectively reverses the initial state of the register in your design, whether specified by an INIT property or by default. Therefore, by default, any register with asynchronous preset (and no reset) will initialize to one.

To workaround this problem, you can apply an INIT property to the register in your design specifying the opposite initial state value. For example, if you have a register “myreg” with an .AP condition and no .AR condition in your design and you want that register to initialize to zero (normally the default), you need to specify the following declaration:

```
xilinx property 'INIT=S myreg';
```

FSM initial state

for FPGA designs

For FPGA designs, 1-hot is the preferred state machine encoding style due the abundance of registers. 1-hot FSMs can be expressed either symbolically or explicitly. Either way, one of the state flip-flops must initialize (power-up) to the one state so that the FSM begins in a legal state.

Symbolic 1-hot FSMs

If you are defining a symbolic 1-hot FSM for an FPGA device, always use the Async_reset statement to identify the initial state of the FSM. If you want to actively and asynchronously reset the FSM to its initial state, declare both the initial state and the asynchronous reset condition as follows:

```
asynch_reset state_name :
reset_condition;
```

If you only want to specify the initial state, but do not wish to actively reset the FSM, declare just the initial state as follows:

```
asynch_reset state_name : 0;
```

Explicit 1-hot FSMs

If you are defining a 1-hot encoded FSM explicitly using ordinary registered signals, always define an asynchronous preset condition (.AP equation) for the register that corresponds to the initial state of your FSM. If you want to actively and asynchronously reset the FSM to its initial state, declare both the initial state register and the asynchronous reset condition as follows:

```
initial_register_name.AP =
reset_condition;
```

Then also declare the same condition for resetting the remaining registers of the FSM, as follows:

```
other_register_name.AR = reset_condition;
```

If you only want to specify the initial state, but do not wish to actively reset the FSM, declare just the initial state as follows:

```
initial_register_name.AP = 0;
```

(You do not need to specify .AR equations for the remaining flops because they will, by default, initialize to zero.)

INITIALSTATE property ignored (bug)

The Xilinx property INITIALSTATE was supported in earlier (pre-M1) versions of XABEL as a way to specify the initial state of 1-hot symbolic FSMs. This property is still sup-

ported in the XABEL-M1 EDIF interface, but it currently does not work properly for FPGA designs in F1.3 and F1.4 versions of the XABEL Interface. In XABEL-M1, the INITIALSTATE property translates to an INIT=S property in the EDIF netlist, which is not supported for FPGA designs. Please use the Async_reset statement or .AP equation extension instead, as described above.

for CPLD designs

Binary encoded FSM

For CPLD designs, binary encoding is the preferred coding style because of the high logic-to-register ratio in the architecture. In a binary encoded FSM, the initial state is commonly assigned the all-zero value. Since zero is the default initial state of all flip-flops in CPLDs, there is typically no need to do anything special to define the initial (power-up) state of the FSM.

If you want to actively and asynchronously reset the binary-encoded FSM to its initial state, simply define .AR equations for all the state registers, assuming an all-zero initial state value.

Symbolic 1-hot FSM

Alternatively, 1-hot encoding is also applicable to CPLD designs and may prove to yield higher-performance results in cases where state transition logic is particularly complex. 1-hot FSMs can be expressed either symbolically or explicitly. Either way, if you are defining a 1-hot FSM, one of the state flip-flops must initialize (power-up) to the one state so that the FSM begins in a legal state.

If you are defining a symbolic 1-hot FSM for a CPLD device, and you do not need to actively and asynchronously reset the FSM to its initial state, use the Xilinx property INITIALSTATE to declare the initial (power-on) state of the FSM, as follows:

```
xilinx property 'INITIALSTATE
state_name';
```

This will automatically apply the property "INIT=S" to the flip-flop corresponding to the initial state in the EDIF netlist. The remaining state flops will, by default, initialize to zero on power-up.

If you do want to actively and asynchronously reset the symbolic 1-hot FSM to its initial state, declare the asynchronous reset condition using the Asynch_reset statement as follows:

```
asynch_reset state_name :
reset_condition;
```

This will automatically generate an asynchronous preset condition for the initial state flop. Normally, you would also specify the Xilinx property INITIALSTATE to define the power-on state, as described above. However, due to a bug in the F1.3 and F1.4 XABEL interface, the power-on state of the initial state flop will be reversed because it has an asynchronous preset condition. Therefore, omit the INITIALSTATE property from your design; the power-on state of the initial state flop will, by default, be one due to the bug.

Explicit 1-hot FSM

If you are defining a 1-hot encoded FSM explicitly using ordinary registered signals, and you do not need to actively and asynchronously reset the FSM to its initial state, use the Xilinx property INIT to set the initial state flip-flop of the FSM to one at power-on, as follows:

```
xilinx property 'INIT=S
initial_register_name';
```

The remaining registers will, by default, initialize to zero.

If you do want to actively and asynchronously reset the explicitly-defined 1-hot FSM to its initial state, declare an asynchronous preset equation for the initial state register, and declare asynchronous reset equations for the remaining state bits, as follows:

```
initial_register_name.AP =
reset_condition;
other_register_name.AR = reset_condition;
```

Normally, you would also specify the Xilinx property INIT=S for the initial state flop to define its power-on state, as described above. However, due to a bug in the F1.3 and F1.4 XABEL interface, the power-on state of the initial state flop will be reversed because it has an asynchronous preset condition. Therefore, omit the INIT property from the state registers in your design; the power-on state of the initial state flop will, by default, be one due to the bug.

Transparent latches

.LH equations use flip-flops in CPLDs

When you use the .LH equation to define a transparent latch, a Xilinx latch primitive (LD) is used in the resulting netlist. For example:

```
mylat node istype 'reg';
Equations
mylat := d_input;
mylat.LH = latch_enable;  `` transparent
high
```

For CPLD designs, the latch primitive is implemented as a flip-flop with a grounded clock and the data input gated by the latch-enable into the flop's asynchronous reset and preset inputs. Essentially, the above latch equations are implemented in a manner equivalent to the following flip-flop equations:

```
mylat node istype 'reg';
Equations
mylat := 0;
mylat.clk = 0;
mylat.AP = d_input & latch_enable;
mylat.AR = !d_input & latch_enable;
```

In CPLD devices, each macrocell flip-flop has a single p-term available for each of its reset and preset inputs. A simple latch equation in which the data and latch-enable inputs are primary inputs (from an input pin or a register output) is implemented efficiently in a single macrocell. However, if the data input is preceded by any combinatorial logic, or if the latch-enable input is any logic function other than a simple AND-gate, then that combinatorial logic cannot be optimized into the same macrocell as the flip-flop and a macrocell feedback delay will be incurred.

Avoid combinatorial feedback latches in CPLD designs (bug)

Normally, there is an alternative way to express a transparent latch is by using a combinatorial feedback equation, as follows:

```
mylat node istype 'com, retain';
mylat = d_input & latch_enable
# mylat & !latch_enable
# mylat & d_input;
```

The retain attribute turns off Boolean minimization in the ABEL compiler and CPLD fitter to retain the redundant product term required to implement a hazard free combinatorial feedback latch.

Normally, all logic in a combinatorial feedback latch equation should be implemented in the same macrocell. However, the F1.3 and F1.4 versions of the CPLD fitter have a problem recognizing the feedback loop and often implement the latch incorrectly using 2 macrocells per latch. If possible, use .LH equations (shown above) to represent transparent latches. Otherwise, the obsolete Plusasm flow provided in F1.4 can successfully implement combinatorial feedback latches without encountering the CPLD fitter problem.

Large comparator/decoder logic may cause ABEL compiler to fail (bug)

Large comparator or decoder functions may prevent compilation of an ABEL design. After invoking the ABEL compiler (abl2edif translator), you may observe that the AHDL2BLF step completes, then the BLIFOPT step begins but never completes. In other cases, the process runs all the way into the BLIF2NET step, which either never completes or produces a system error after exhausting all available virtual memory. One possible cause is that the design contains some large comparator logic (of the form Y = A == B) or decode logic (of the form Y = A == constant), especially if the comparator or decode logic is combined with any other combinatorial logic in the same equation. The suspected cause of this problem is that the ABEL compiler is trying to process the negated logic form of the expression which contains an exceedingly large number of min-terms.

You can usually workaround this problem by modifying the problematic equation in your ABEL source. Best results are usually obtained by decomposing equations containing comparator/decoder logic by separating out the comparator /decode logic into new intermediate node equations. When factoring out comparator logic, express the intermediate node equations as active-low comparators of the form NOT_Y = A != B, which is the form containing fewer min-terms. Also, prevent collapsing of the intermediate nodes by applying the attribute KEEP.

For example, if the original equation is:

```
Q = (A == B) & something # (C == D) & smore ;
```

Decompose into:

```
A_ne_B = (A != B);  "new node
C_ne_D = (C != D);  "new node
Q = (!A_ne_B) & something # (!C_ne_D) &
smore ;
```

Then declare the new nodes with the KEEP attribute, as in:

```
A_ne_B, C_ne_D node istype 'KEEP';
```

Attributes for controlling design implementation

The discussion in this section assumes you are familiar with the capabilities of the target PLD architecture and the general means for controlling the implementation software. This section focuses on the techniques and issues specific to controlling XABEL designs.

Pin assignment

Numeric pin names in ABEL "pin" declarations

You can indicate pin assignments for most packages directly in your ABEL pin declarations. For example:

```
a pin 34;
b, c pin 35, 36 istype 'reg';
```

Pin numbers are applicable only to top-level ABEL designs for either FPGA or CPLD devices.

BGA pin names in UCF file

ABEL only accepts numeric pin numbers. If you are using a BGA package (which uses alphanumeric pin designations), specify your pinout in a User Constraint file (UCF) using the LOC property. For example:

```
net a LOC=A7;
```

Output slew (FAST, SLOW)

You can control slew rate for specific output or I/O pins with the FAST and SLOW attributes. Use these attributes to selectively control whether specific pins operate in fast slew rate (FAST) or slew rate limited (SLOW) mode as follows:

```
xilinx property 'FAST | SLOW signal_list';
```

For Example:

```
xilinx property 'SLOW q1 q2 q3';
```

The FAST and SLOW properties are applicable only to top-level ABEL design for either FPGA or CPLD devices.

CPLD fitter patch for pin assignment and FAST/SLOW slew-rate (bug)

The XC9500 design implementation software, versions M1.4 and F1.4, ignores FAST, SLOW and LOC properties applied to I/O pads in top-level XABEL designs. If you need to use pin assignment or the FAST or SLOW Xilinx property in your ABEL designs, please download the latest CPLD fitter patch named "cpld_nt*.zip" from the Xilinx web site (www.xilinx.com) in the Support → File Download → Software Help → M1.4 Alliance area.

Preserving combinatorial nodes (KEEP)

If you want to preserve a combinatorial node in your ABEL design so that it remains intact throughout design implementation, apply the attribute KEEP to the node declaration in your ABEL design as follows:

```
signal_list node istype 'KEEP';
```

The KEEP attribute is recognized by the ABEL compiler so that it will not collapse the node during ABEL compilation and netlisting. The KEEP attribute is also propagated to the EDIF netlist so that the core implementation software (CPLD fitter or FPGA mapper) does not collapse the node during design optimization.

The KEEP attribute is applicable to top-level ABEL designs and modules for schematic designs.

Global buffers for CPLD (BUFG)

You can manually assign selected input pin signals in your top-level ABEL design to global nets on a CPLD using the BUFG attribute as follows:

```
xilinx property 'BUFG={CLK | OE | SR}
signal_name;
```

For example:

```
xilinx property 'BUFG=CLK my_clock';
xilinx property 'BUFG=OE my_enable';
xilinx property 'BUFG=SR my_reset';
```

Macrocell power mode for CPLD (PWR_MODE)

Use the PWR_MODE attribute to selectively control whether specified logic operates in high speed or low power mode. The default power mode for the design is controlled in the Design Manager, and is initially set to STD for new projects. Use the following syntax:

```
xilinx property 'PWR_MODE={LOW|STD}
signal_list';
```

For example, to set the functions out0 and out1 to low power mode, (the remaining functions will use the default power mode) use the following:

```
xilinx property 'PWR_MODE=LOW out0 out1';
```

Timespecs

For top-level ABEL designs, timespecs must be specified in a UCF file. For ABEL modules in a schematic design, timespecs are typically added to the schematic, but could also be specified in a UCF file.

Either way, timespecs can reference timing groups by names attached to elements in the design. You can attach a timing group name (TNM) to a signal in an ABEL design or module using the TNM property, as follows:

```
xilinx property 'TNM=group_name
signal_list';
```

XC9500 local feedback

The local feedback path will be used when you have grouped logic functions together in the same function block with a special form of the LOC attribute and the local feedback path is required to meet a timing constraint. The special form of the Xilinx LOC property required to control function block placement of internal logic is as follows:

```
xilinx property 'BLOCK signal_name
LOC=FBnn';
```

For example, assume you want to group flip-flops REG_X and REG_Y into the same function block so that you can speed up the cycle time by using the local feedback path. Also assume there is some combinatorial logic between these flip-flops that was not fully optimized, resulting in an additional macrocell feedback between the flip-flops and whose output is GATE_A. You would also need to group the combinatorial logic (GATE_A) into the same function block. To place these functions in function block 1, the required Xilinx LOC properties would be expressed as follows:

```
xilinx property 'BLOCK REG_X LOC=FB1';
xilinx property 'BLOCK REG_Y LOC=FB1';
xilinx property 'BLOCK GATE_A LOC=FB1';
```

Excessive KEEP/RETAIN/LOC causes CPLD fitter problems (bug)

In earlier (pre-M1) versions of the XABEL Interface, it was often necessary to explicitly control the mapping of large portions of a CPLD design in order to achieve satisfactory performance. Such explicit control typically included assigning function block locations and either forcing or blocking logic optimization or minimization. These controls were expressed using now-obsolete Plusasm properties such as PARTITION and LOGIC_OPT.

In XACT-M1 implementation software, the CPLD fitter has been enhanced to provide high performance results more

automatically. It is usually unnecessary to constrain the mapping or optimization for large portions of the design to achieve satisfactory results. In some cases, over-constraining a design in a manner familiar to earlier versions of the software may actually cause severe problems when using modern fitter software. These problems can range from poorer quality of results to software crashes. If you experience such problems and your design is heavily constrained, we recommend removing all but the essential constraints and trying again. Then constraints could be added gradually as needed to remedy specific needs of your design.

The constraints that have been seen to cause problems, if used excessively, are function block LOC (as in xilinx property 'BLOCK *signal_name* LOC=FB*nn*';), RETAIN and KEEP.

Mapping ABEL equations directly to CPLD macrocells

Occasionally, the CPLD fitter may implement a particular equation or set of equations less efficiently than originally expressed in ABEL. This is particularly true when your design contains combinatorial nodes that feed back into other logic equations. By default, the fitter does not give preference to the partitioning of logic as expressed in the ABEL equations. It is free to optimize combinatorial logic across equation boundaries and form intermediate macrocell outputs from fragments of an equation.

If, for certain equations in your design, you prefer that the fitter implement the equations as expressed in ABEL, you could use the KEEP attribute on combinatorial nodes to prevent the fitter from optimizing logic across equation boundaries. Then, if you notice that the logic within your equations does not get fully optimized into the same CPLD macrocell, you could increase the Collapsing Pterm Limit in the Design Manager Implementation Options template to further flatten that logic.

Processing XABEL designs

Improving Performance in CPLD designs

For some CPLD designs you may notice that the logic of some of your equations does not get fully flattened, resulting in poor performance and sometimes excessive logic utilization. Performance problems can be observed in the Timing Report summaries. Incomplete flattening is also indicated in the Fitting Report under the "Resources Used by Successfully Mapped Logic" section, by the presence of several "internal" signal names that do not match any of the signal names in your design. These internal names each represent a CPLD macrocell feedback used to implement intermediate logic nodes within your equations.

To get the fitter to further flatten your design, increase the Collapsing Pterm Limit in the Advanced Optimization tab of the XC9500 Implementation Options menu in the Design Manager GUI. First try increasing the Pterm Limit to 90 (the maximum) and rerun the fitter. For many designs, all your equation logic will become flattened and the internal signal names will disappear from the Fitting Report.

For some designs, especially those containing combinatorial node equations or complex logic functions, increasing the Pterm Limit may cause the logic utilization to explode and the design will not fit. You may be able to find a Pterm Limit less than 90 that produces satisfactory performance while successfully fitting the device. Otherwise, you may need to apply the KEEP attribute to some of the combinatorial node equations in your design to prevent these nodes from being flattened too much when you raise the Pterm Limit. If you still have some equations in your design that consume too many logic resources when flattened, you may need to decompose large equations into smaller intermediate equations and apply the KEEP property to your intermediate nodes as needed.

ABEL modules for Alliance designs

The XABEL interface is only shipped in the Xilinx Foundation product and is not included in the Alliance product. If you are using the Alliance M1.4 software with a 3rd-party schematic capture tool and you want to include XABEL macros in your schematic design, you can add the XABEL Interface to your PC to process your ABEL modules.

Your alternatives are:

1. Install all design entry tools from the Foundation F1.4 product and use the Foundation HDL Editor GUI for ABEL design entry and translation. Your Alliance M1.4 software can either be on the PC or workstation.
2. Install only the XABEL Interface from the Foundation F1.4 product and use a text editor for ABEL design entry and use commands in a DOS window for ABEL translation. Your Alliance M1.4 software can either be on the PC or workstation.
3. Download the XABEL Interface from the Xilinx web site into your existing Alliance M1.4 installation and use a text editor for ABEL design entry and use commands in a DOS window for ABEL translation. Your must have Alliance M1.4 software on your PC.

Using Foundation design entry tools for ABEL module development

Installation

If you are installing both Alliance and Foundation software onto the same PC, they can be installed in either order.

1. Insert the Foundation F1.4 Design Entry Tools CD on your PC.

2. When the Master Setup screen appears, choose "Install Design Entry Tools." If the Master Setup screen does not appear, run the Setup.exe program from the CD.
3. Follow the instructions on the screen. When asked which type of setup to run, choose Custom.

Note: If you plan to use the Foundation tools for schematic entry or simulation as well as ABEL compilation, choose "Typical Installation" instead. (See the Foundation Release Document for details.)

4. On the Select Components screen, you will have the option to install various portions of the software. To reduce the amount of disk space required, you may select to not install the sample projects, or just install those which involve Abel. You may also choose to only install those libraries for the devices you will be targeting. Below is a summary of the available components:

   Program Files - Required

   X-VHDL - Not Required. Requires separate license to run.

   Sample Projects - Not required, but you may want to install just the Abel projects.

   System Libraries - At least one required. You may choose only those families you will be targeting.

   Keylock Drivers - Not required. (Only required for X-VHDL.)

   Even with a minimal install, you will have the ability to use all features of the Foundation software, with the exception of X-VHDL or Express synthesis.

5. When Design Entry Tools installation is complete and the Master Setup screen reappears, select "Design Implementation Tools". (The XABEL Interface software resides on the Design Implementation Tools CD.)
6. Insert the Foundation Design Implementation Tools CD. (Run Setup.exe if the installer doesn't automatically start.)
7. Follow the instructions on the screen. If Alliance M1.4 software is also installed on your PC, you should specify the same directory to install the XABEL Interface. This is because the XILINX environment variable, also required by XABEL, cannot specify more than one path.
8. When asked to select the type of installation, choose "Design Entry Tool Components Only".

Design flow

The following procedure outlines the basic flow for creating a project in Foundation and compiling one or more ABEL macros for incorporation into a 3rd party schematic. For complete documentation of the Foundation Design Entry tools and the XABEL Interface, refer to the on-line help in the Foundation Project Manager (Help → Foundation Help Contents).

1. Invoke the Foundation Project Manager.
2. Create a new project by selecting File → New Project.
3. Choose the appropriate design directory and family.
4. Invoke the HDL Editor by clicking on the HDL Editor button in the Project Manager.
5. Select "Use HDL Design Wizard" or "Create Empty" to enter a new ABEL design. Otherwise, select "Existing File" and browse to find an existing.ABL file. The Design Wizard creates a template ABEL design based on names of macro pin names you enter. The HDL Editor color codes ABEL keywords and provides syntax checking capability.
6. Since the Abel file will be a module in a top-level schematic, be sure that the "Macro" compile switch is selected in the Synthesis → Options dialog.
7. To synthesize the ABEL code, select Synthesis → Synthesize. An EDIF (.EDN) netlist file will be created for the module and placed in the project directory.

Using the XABEL Interface alone

You can install the XABEL interface either from the Foundation F1.4 CDs or by downloading it from the Xilinx web site.

Installation from Foundation CDs

If you are installing both Alliance and XABEL Interface software onto the same PC, they can be installed in either order.

1. Insert the Foundation Design Implementation Tools CD and run Setup.exe.
2. Follow the instructions on the screen. If Alliance M1.4 software is also installed on your PC, you should specify the same directory to install the XABEL Interface. This is because the XILINX environment variable, also required by XABEL, cannot specify more than one path.
3. When asked to select the type of installation, choose "Design Entry Tool Components Only".

Downloading from the web

You must install Alliance M1.4 on your PC to use the XABEL Interface provided on the web. This is because the XABEL Interface relies on software libraries included in XACT-M1.

You must install both Alliance M1.4 implementation (core) software and the XABEL Interface onto a local hard disk drive on your PC. This is because the XABEL Interface does not run reliably if installed onto a network drive.

1. The XABEL Interface is located at www.xilinx.com → Support → File Download → Software Help → Foundation. The filename is xabel140.zip (or the highest numbered revision of xabel14*.zip). Download the xabel140.zip file to any location.

2. Unzip the file using PKZip or equivalent. The ZIP file contains an installer package that you can extract to any location on your PC.
3. Read the installation and usage instructions contained in the text file read_abl.txt.
4. Run the installer, Setup.exe. Follow the directions on the screen. You must set the Destination Directory to the same directory where you installed Alliance M1.4 implementation (core) software.

Note: Once you set the Destination Directory and click OK, you cannot cancel the installer.

For detailed documentation on using XABEL, including CPLD and FPGA design techniques and an ABEL language reference guide, you can also download the Foundation on-line help files from the Xilinx web site at www.xilinx.com → Support → File Download → Software Help → Foundation, filename “fnd_help.zip”.

Design flow

When using the XABEL Interface alone (without the Foundation Design Entry Tools), XABEL design entry is performed using a conventional text editor. XABEL designs are then compiled using a 1-line command entered in a DOS window as follows:

1. Open a DOS window.
2. Change directory (CD) to the directory containing your ABEL source file.
3. Execute the abl2edif command as follows:

   `abl2edif -s` *`level module_name`*

   where *level* is "top" for top-level ABEL design, or "mod" (default) for module to use in a schematic. The abl2edif program generates the following output files:

   abl2edif.log: log file of program execution

   module_name.edn: output EDIF netlist

   module_name.err: error log from program execution

   module_name.smx: simulation output file (if the design contains test vectors)

   module_name.tmv: test vector file for XC9500 functional test (if the design contains test vectors)

Instantiating ABEL macros in a schematic

Viewlogic Workview Office

Once you have created the EDIF file in the Foundation environment, you must instantiate that EDIF in your Viewlogic schematic.

Create a new symbol for the ABEL module complete with input and output pins. Make sure that all the input and output ports match the symbol by name. Use square brackets for bus notation: BUS[3:0]. If a symbol created by SYMGEN already exists, simply remove the DEF=XABEL and FILE=abelfile.ABL properties before continuing.

Two more properties must be added to complete the symbol. Right-click in the symbol window (but outside the symbol box itself) and select Properties. Under the Block tab, change the Symbol Type to Module. This will prevent the EDIF netlister from looking for an underlying schematic. Then, under the Attributes tab, add an attribute with a Name of FILE and a Value of abelfile.EDN. If the EDIF file is not located in the project directory, then the full path to the .EDN file must be specified.

If the ABEL code is modified in such a way that the input or output ports are modified, then the symbol will have to be manually updated to match the new ABEL module.

Because the ABEL module does not have a gate-level representation within the Viewlogic realm, the design will have to be compiled through NGDBUILD in order to process the ABEL portions before performing a functional simulation. Chapter 4 of the Viewlogic Interface and Tutorial Guide describes this process.

There is also a push-button solution available for these steps. Solution #1985 in the Xilinx Solutions Database contains the files and setup instructions for this flow. No changes to the timing simulation flow are required. This procedure applies to Powerview users as well, although some of the commands listed above will differ slightly for the workstation version of ViewDraw.

Mentor Graphics Design Architect

Once you have created the EDIF file in the Foundation environment, you must instantiate that EDIF in your Mentor Graphics schematic. Note that, since this EDIF file comes from a Windows 95/NT environment, you should run a DOS-to-UNIX file-conversion utility (such as dos2unix) on this EDIF file to avoid possible file-format problems.

Create a new symbol for the ABEL module complete with input and output pins. Make sure that all the input and output ports match the symbol by name. Use parentheses for bus notation: BUS(3:0). If a symbol created by SYMGEN already exists, simply remove the DEF=XABEL and FILE=abelfile.abl properties before continuing.

One more property must be added to complete the symbol. With the symbol for the ABEL module loaded into the symbol editor, select Right Mouse Button → Properties (logical) → Add Single Property. For Property Name, enter FILE. For Property Value, enter abelfile.edif, where abelfile is the name of the ABEL module represented by this symbol. If the EDIF file is not located in the project directory, then the full path to the .edif file must be specified.

If the ABEL code is modified in such a way that the input or output ports are modified, then the symbol will have to be manually updated to match the new ABEL module.

Because the ABEL module does not have a gate-level representation within the Mentor realm, the design will have to be compiled through NGDBUILD and PLD_EDIF2SIM in order to process the ABEL modules and generate a suitable simulation model for PLD_QuickSim. PLD_QuickSim must then be run on the output netlist from PLD_EDIF2SIM. (To annotate simulation values to the original schematic, you may enable cross-probing in PLD_QuickSim). Chapter 6, *Mixed Designs with Schematic on Top* in the Mentor Graphics Interface/Tutorial Guide, describes this process in the *Functional Simulation After Synthesis* section.

No changes to the timing simulation flow are required.

Optimization of XABEL logic

FPGA designs

Prior to XACT-M1, all logic from ABEL modules had to be optimized for FPGA architectures by the XABEL Interface itself before writing the XNF netlist. In M1, the XABEL Interface does not perform any of the Xilinx-specific optimization. The EDIF netlist produced by XABEL is architecturally generic. FPGA-specific optimization is performed by the OPTX program, which is part of the mapping step of the implementation software. OPTX optimizes logic on a module-by-module basis. XABEL modules are identified by the “OPTIMIZE” property which is automatically written at the root level of each XABEL module netlist. When the top-level schematic design is submitted to the implementation software for mapping, each of the XABEL netlists are, in turn, read in. The OPTIMIZE property remains associate with each XABEL module and OPTX performs logic optimization on each of the modules tagged with the OPTIMIZE property.

OPTX optimization is normally desirable for all XABEL modules. If you prefer to disable OPTX optimization on an XABEL module, you can turn off the OPTIMIZE property by specifying the following in your ABEL source declarations:

```
xilinx property ‘OPTIMIZE=OFF’;
```

CPLD designs

All logic in CPLD designs, including XABEL-generated logic, is always optimized by the CPLD fitter. The OPTIMIZE property which is automatically written into the XABEL EDIF netlist is ignored by the CPLD fitter. The only way to control logic optimization in a CPLD design is by using the KEEP, RETAIN and COLLAPSE properties. (These are described elsewhere.) You can also adjust the settings of the CPLD fitter options template in the Design Manager, as described in the Foundation on-line help document.

OLE server and registry issues

Multiple ABEL versions in the Windows registry (bug)

If any other non-Xilinx ABEL software is installed on your PC, you may get one of several error conditions including an OLE communication server error, a licensing error, an error about a corrupted encrypted file, or other installation-related error messages.

To allow Xilinx-ABEL to run, you must remove the registry keys for the non-Xilinx ABEL software from your Windows registry. To remove the non-Xilinx ABEL registry keys:

1. Run the Registry Editor (Regedit.exe under your Windows directory).
2. Use Edit → Find to search for the folder named "DIOEDA".
3. Open DIOEDA and then open the Products folder. The keys for Xilinx-ABEL are stored in the folder XABELM1; the keys for any non-Xilinx ABEL product would be stored in an adjacent folder.
4. Before deleting the non-Xilinx ABEL folder, you may want to save a copy of it in a .reg file (using Registry → Export Registry File) in case you want to later restore those settings to run the non-Xilinx ABEL software.
5. Select the non-Xilinx ABEL folder and use Edit → Delete to remove it.
6. When finished, close the Registry Editor; then reboot your PC (to terminate any resident OLE communications server used by ABEL).

If you need to remove the Xilinx-ABEL registry keys (in order to run a non-Xilinx ABEL product), you can restore XABEL by running the registry file, bin\nt\install\xabel.reg, located in your Xilinx implementation (core) software installation area; then re-boot your PC.

Windows 95 hangs due to OLE (bug)

On some Windows-95 systems, the XABEL translator (abl2edif or abl2pld) runs OK the first time it is invoked, but will not run again subsequently. This is due to a problem with the OLE communications server (ntolesrv.exe) used by the ABEL compiler. The OLE server normally remains active for a short while after the abl2edif translator completes, and then times out. A problem occurs when the OLE server fails to time out and prevents the abl2edif translator from being run subsequently. If you cannot successfully terminate the OLE server, you may need to reboot your PC to clear this problem. This problem has not been observed on Windows NT systems.

XC9500 JEDEC test vectors

In addition to programming information, the JEDEC download files produced by the CPLD fitter software for XC9500

devices can also contain functional test vectors. These test vectors can be used by the Xilinx JTAG download system as well as 3rd-party programming equipment to functionally test XC9500 devices after programming.

The CPLD fitter can automatically translate simulation test vectors embedded in ABEL designs and write them into the XC9500 JEDEC file. If your XABEL design contains a Test_vector section, then in addition to performing simulation, the XABEL translator produces a test vector interface (.TMV) file.

How to use .tmv file

When you are ready to implement your XABEL design, open the Options template in the Design Manager. Select the Programming tab and browse for the .TMV file produced by the XABEL Interface. The test vectors from the .TMV file will automatically be written into the programming JEDEC file.

Note: Only test vectors from a top-level (stand-alone) ABEL design can be used for device functional testing.

Use upper case signal names (bug)

Due to a bug in the F1.3 and F1.4 CPLD fitter, only signal names that are all upper-case will get their test vector information translated into the JEDEC file. Any signal name containing lower-case letters will result in “X” being written in the corresponding column of the test vector section of the JEDEC file. Therefore, if you plan to perform device functional test, use only all-upper-case names for all pin signals in your ABEL design.

When to use Plusasm (CPLD only)

In pre-M1 versions of the XABEL Interface, all CPLD designs were translated into the Plusasm equation language which was the file format read by the CPLD fitter. In XACT-M1, the primary design file format is the EDIF netlist. Plusasm is still supported by M1.4 software for back-compatibility of interfaces that still depend on Plusasm equation files. However, EDIF should be used for all new designs if at all possible.

PLUSASM LANGUAGE WILL NO LONGER BE ACCEPTED FOR DESIGN ENTRY IN LATER RELEASES OF XILINX DESIGN IMPLEMENTATION SOFTWARE.

One of the characteristics of Plusasm is that each equation in the Plusasm file (.pld) is automatically forced to be fully optimized and mapped into a single CPLD macrocell. While this could be beneficial if the equations are optimally designed to suit the CPLD architecture, it could otherwise prevent the fitter from finding more efficient implementations of the same logic. In contrast, when an ABEL design is translated to EDIF, the combinatorial logic in each ABEL equation is decomposed into a network of individual AND/OR gates. The fitter no longer recognizes equation boundaries and is free to optimize the logic as well as it can.

Existing design with unconverted Plusasm properties

To constrain a design when using the Plusasm flow, it was necessary to embed Plusasm language declarations in the ABEL source design. These Plusasm declarations were not in the form of conventional properties and cannot be supported, as is, via the EDIF interface. In XACT-M1, a new set of properties are provided that are compatible with EDIF as well as other forms of Xilinx design entry including schematics and HDLs.

If you have an existing XABEL CPLD design that required XEPLD PROPERTY or PLUSASM PROPERTY statements, you must either remove or replace these properties with the supported EDIF-compatible properties or you must use the Plusasm flow (see *Converting Plusasm properties into M1 attributes,* next column). If your existing design does not contain Plusasm based properties, you should be able to use the new EDIF-based flow without design modification. If you are developing a new XABEL CPLD design, DO NOT USE Plusasm based properties. If you use the Plusasm flow, you cannot use any of the XILINX PROPERTY statements supported by XACT-M1 software.

Pinlocking designs in XABEL-M1 using pre-M1 pinouts (bug)

If you have a guide file (.gyd) containing a pinout that you want to use to pinlock a design iteration using the XABEL-M1 Interface, and the guide file was created using a pre-M1 version of XABEL, you may have difficulty with case-sensitivity of pin names. In pre-M1 XABEL, which used the Plusasm flow, all signal names were automatically converted to upper case. The resulting guide files therefore contained only upper case pin names. In XABEL-M1, which uses the EDIF flow, the case of the signal names in your ABEL file is preserved throughout design implementation. When trying to pinlock using an old guide file, if the name of a pin in the guide file does not exactly match the name read in from the EDIF netlist, the pinout information will not be applied.

To workaround this problem, either edit the old guide file to restore the case of each of your pin names to match the names in your ABEL design, or modify your ABEL design to use all upper-case names for external pins.

EDIF flow bugs with no workaround

The F1.3 and F1.4 versions of the XABEL Interface and the core implementation software contain some bugs that you may not be able to workaround using the EDIF flow. If none of the workarounds suggested in this application note solve the problem, the Plusasm flow can sometimes provide an alternative solution.

Persistent fitter problems or poor results

In some cases, the XACT-M1 CPLD fitter may not be able to achieve the same quality of results as obtained in an ear-

lier version of XABEL using the Plusasm flow. This typically occurs when the distribution of logic among the equations in the ABEL design is particularly well matched to the CPLD architecture. This might also occur if the earlier design was constrained using Plusasm properties in a way that cannot be reproduced using the EDIF flow.

If none of the optimization suggestions described in this application note allow you to obtain satisfactory performance, the Plusasm flow can sometimes provide an alternative solution.

Converting Plusasm properties into M1 attributes

This is a summary of the Plusasm properties that were supported in the pre-M1 XABEL Interface and the equivalent Xilinx properties compatible with the XABEL-M1 EDIF-based interface:

```
plusasm property 'FASTCLOCK signal_list';
xilinx property 'BUFG=CLK signal_list';
plusasm property 'FOEPIN signal_list';
xilinx property 'BUFG=OE signal_list';
plusasm property 'PARTITION FBnn
signal_name...';
xilinx property 'BLOCK signal_name
LOC=FBnn';
plusasm property 'LOGIC_OPT OFF
signal_list';
signal_list {NODE | PIN} istype 'KEEP';
plusasm property 'MINIMIZE OFF
signal_list';
signal_list {NODE | PIN} istype 'RETAIN';
plusasm property 'PWR {LOW | STD}
signal_list';
xilinx property 'PWR_MODE={LOW | STD}
signal_list';
plusasm property 'FAST ON signal_list';
xilinx property 'FAST signal_list';
```

Using the Plusasm design flow

The XABEL Interface, whether installed from the Foundation F1.4 CD or by downloading from the web, provides an alternative design flow for CPLD designs using Plusasm equation files instead of EDIF netlists.

Using the Foundation Design Entry tools

If you are using Foundation F1.4 to develop a top-level ABEL design for CPLD, the Plusasm flow is enabled through the Foundation Project Manager. The Foundation tools can only use the Plusasm flow if you are developing a top-level ABEL design. If you are developing ABEL modules for use in schematic-based designs and you need to use the Plusasm flow, you must compile your ABEL modules using the command-line interface described below.

To enable the Plusasm flow:

1. In the Foundation Project Manager, select File → Configuration.
2. In the Configuration window, click on "View INI File".
3. In the Report Browser window that appears, find the lines containing

   ```
   [EXTENSIONS]
   ;XABELNETLIST=PLUSASM
   ```

4. Delete the semicolon (;) in front of XABELNETLIST to enable the feature.
5. Save the file (File → Save) and close the Report Browser.
6. Click OK in the Configuration window to close it.
7. Exit the Foundation Project Manager (File → Exit) and restart it to read the new configuration.
8. Create a project in which to develop your ABEL design.
9. Invoke the HDL Editor and use the Synthesis → Synthesize command as you would normally. Be sure the "Chip" compile switch is selected in the Synthesize → Options menu to produce a top-level Plusasm design.

After the PLUSASM flow is enabled, the Foundation software creates PLUSASM equation files (.PLD) for all top-level ABEL CPLD designs. When you first invoke the Xilinx Design Manager from the Project Manager for a new project, it will automatically read the PLUSASM (.PLD) file for design implementation instead of looking for an EDIF netlist. FPGA designs and all ABEL macros to be used in schematic designs will continue to use EDIF netlists for design implementation.

Note: After it creates the Plusasm equations file (runs the ABL2PLD translator), the Foundation system continues to run the ABL2EDIF translator to produce an EDIF netlist. The EDIF file is used only for functional simulation, which is optional; it is not used for design implementation.

Note: If you have already compiled the ABEL design using the EDIF flow, you should create a new project before re-compiling using the Plusasm flow. Otherwise, the implementation software may continue to read the existing .EDN file instead of the new .PLD file.

Using DOS command-line

If you are using the XABEL Interface installed from the Foundation CD or downloaded from the web, ABEL designs are translated to Plusasm using a 1-line command as follows:

1. Open a DOS window.
2. Change directory (CD) to the directory containing your ABEL source file.
3. Execute the abl2pld command as follows:

   abl2pld -s *level module_name*

   where *level* is "top" for top-level ABEL design, or "mod" (default) for module to use in a schematic. The abl2pld

program generates the following output files:

abl2pld.log: log file of program execution

module_name.pld: output Plusasm equation file

module_name.err: error log from program execution

module_name.smx: simulation output file (if the design contains test vectors)

module_name.tmv: test vector file for XC9500 functional test (if the design contains test vectors)

4. In the Design Manager, create a project and specify (Browse) the .PLD file as the Input Design.