

# Compiling VHDL Designs Using Foundation XVHDL

rev. 1.2  
February 1997

This application note outlines recommended software flows for compiling VHDL designs using the Xilinx Foundation software tools. The overall Foundation XVHDL flow is described, and three compilation methods are described. VHDL partitioning and other design issues are discussed. The software flow for each compilation method is outlined in detail. Finally, an example design is presented along with step-by-step instructions for using each compilation method.

The methods described in this application note apply to top-level VHDL designs targeting Xilinx FPGAs, not VHDL macros used in a top-level schematic design. Xilinx recommends splitting large VHDL macros into smaller macros, and connecting these using a schematic.

**NOTE:** This application note is intended to be used with Xilinx Foundation Series software, version 6.0.2.

## Table of Contents

FOUNDATION XVHDL DESIGN FLOW .....	1
COMPILATION METHODS .....	2
PARTITIONING GUIDELINES .....	4
DESIGN ISSUES .....	5
DESIGN FLOWS .....	6
EXAMPLE DESIGN .....	8

## Foundation XVHDL Design Flow

Figure 1 illustrates the Foundation XVHDL design flow, starting with VHDL files and ending with one or more optimized Xilinx Netlist Format (XNF) files.



**Figure 1: XVHDL Design Flow**

The VHDL compilation takes place in two steps. First, the VHDL is converted to an XNF netlist. Then, the IMPROVEX program further optimizes the combinational logic and maps the design into the target FPGA architecture.

## Compilation Methods

There are three basic methods to use when compiling a VHDL design using the Foundation tools: **Fully Grouped**, **Fully Ungrouped**, or **Selectively Grouped**. The following sections describe each method, the advantages and disadvantages of using each method, and when each method should be used.

Refer to the *Design Flows* section to learn how to use each compilation method. The *Example Design* section contains a sample design and step-by-step examples of using each compilation method.

### Fully Grouped

In a Fully Grouped compile, all VHDL files are compiled at the same time. The compiler output is a single XNF file which contains all the logic described in the design (Figure 2).



**Figure 2: Fully Grouped Compile Flow**

The advantages of the Fully Grouped compilation method are:

1. Software has maximum opportunity to optimize combinational logic.
2. Easiest method to implement, requiring only a single “Synthesize” command.

The disadvantages of this method are:

1. Long IMPROVEX run-times.
2. Possible loss of quality in the results (the optimization program IMPROVEX works best on files containing less than 3,000 gates).

Use the Fully Grouped compilation method for small designs (3,000 gates or less).

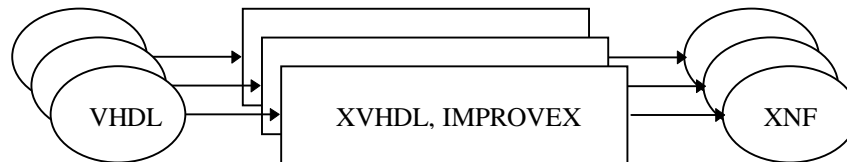
You can also perform a Fully Grouped compile if you’re not sure how large the design is. After IMPROVEX optimizes the design, the report will state approximately how many Configurable Logic Blocks, or **CLBs**, are used. Use this number to determine whether one of the other compilation methods should be used (3,000 gates is about 150 XC3000 or XC4000 CLBs, or about 80 XC5200 CLBs)..

**NOTE:** Designs consisting of a single VHDL file can only be compiled using the Fully Grouped compilation method. If another method needs to be used, the design must be manually split into multiple VHDL files.

### Fully Ungrouped

In a Fully Ungrouped compile, each VHDL file is compiled separately. Some VHDL files may contain no logic, simply forming connections between other VHDL files.

The final result is one XNF file for each VHDL file (Figure 3). These files are merged together using the Translate command in the Xilinx Design Manager, or by running the XNFMERGE program.



**Figure 3: Fully Ungrouped Compile Flow**

The advantages of the Fully Ungrouped compilation method are:

1. Shortest IMPROVEX run-times for each compile.

The disadvantages of this method are:

1. Multiple “Synthesize” commands must be executed.
2. Quality of results can suffer if the VHDL files are too small (refer to the *Partitioning Guidelines* section for tips to maximize logic optimization).
3. All of the XNF files must be merged before functional simulation can be performed (refer to the *Design Issues: Simulation* section).

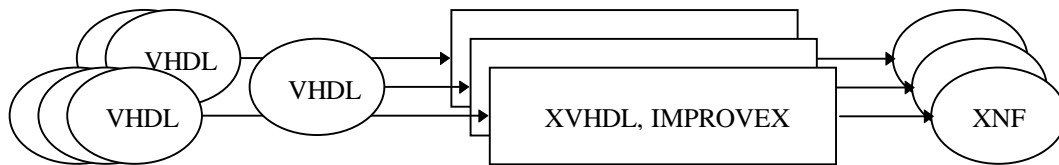
Use the Fully Ungrouped compilation method for medium or large designs comprised of a few VHDL files.

## Selectively Grouped

In a Selectively Grouped compile, groups of VHDL files, or **blocks**, are compiled separately. Some blocks may contain no logic, simply forming connections between other blocks.

The final result is one XNF file for each block (Figure 4). These files are merged together using the Translate command in the Xilinx Design Manager, or by running the XNFMERGE program.

The Fully Ungrouped compilation method is a special case of this method, where each block is a single VHDL file.



**Figure 4: Selectively Grouped Compile Flow**

The advantages of the Selectively Grouped compilation method are:

1. Shorter IMPROVEX run-times than a Fully Grouped compile.
2. Best quality of results for large designs, if the design is partitioned well (refer to the *Partitioning Guidelines* section for partitioning tips).

The disadvantages of this method are:

1. Multiple “Synthesize” commands must be executed.
2. All of the XNF files must be merged before functional simulation can be performed (refer to the *Design Issues: Simulation* section).

Use the Selectively Grouped compilation method for medium or large designs comprised of many VHDL files.

## Partitioning Guidelines

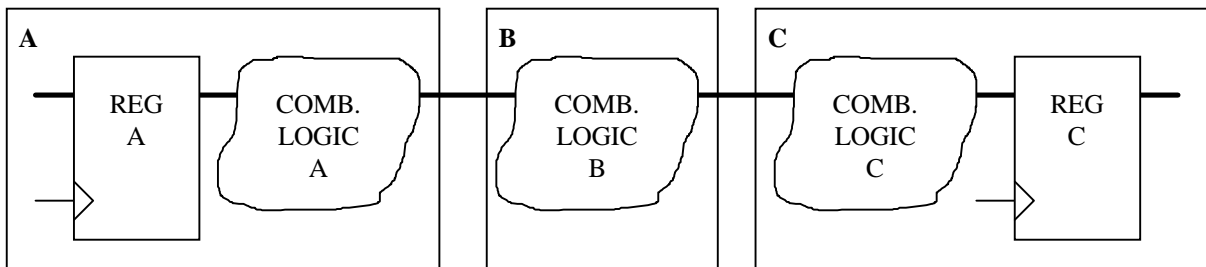
The degree to which IMPROVEX can optimize an XNF file depends on the size of the file. The compilation methods described above are provided as guidelines to avoid creating XNF files that are too large to be optimized well.

Another factor that affects optimization results is how the design is partitioned into entities, and how those entities are grouped into blocks. If a design is poorly partitioned, logic optimization can suffer.

Here are some VHDL coding and partitioning guidelines that will help improve logic optimization.

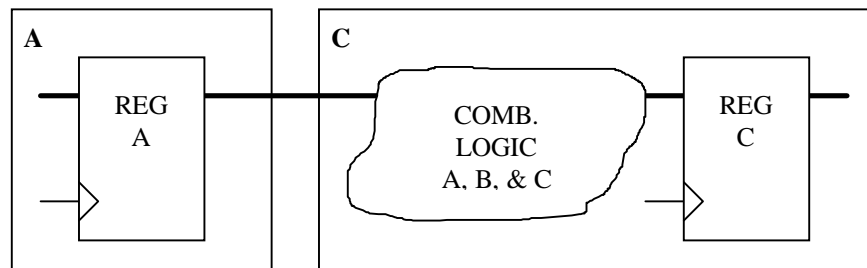
### Avoid Imposing Boundaries on Combinational Paths

If parts of a combinational logic path are compiled in separate blocks (Figure 5), no logic optimization can be performed across the block boundaries.



**Figure 5: Combinational Logic Path Split Across Boundary - BAD**

Partition the design so that combinational paths are not split across multiple blocks (Figure 6). This gives the software the most opportunities to optimize combinational logic on the path.



**Figure 6: Combinational Logic Path Grouped Into One Block - GOOD**

### Register all Block Outputs

Partition the design into blocks in such a way that all block outputs are registered. This guarantees that no boundaries are being imposed on any combinational paths. See Figure 6 for an example.

If different compilation methods or partitioning schemes are going to be used on a single design, try to register all outputs of the major entities in the design (this may require merging some entities or changing the way the design is divided into entities). This will ensure that all block outputs are registered, regardless of the compilation method used or how the design is partitioned.

# Design Issues

## Entities

When using the Fully Ungrouped or Selectively Grouped compilation methods, each block is composed of a **parent entity** and zero or more **child entities**. Child entities are instantiated as components in the architecture of the parent. This is analogous to a schematic: a parent entity is like a schematic sheet, and a child entity is a symbol on that sheet. A child entity may have children of its own, just as a symbol on a schematic may actually be another schematic sheet.

In order to successfully compile a design using the Fully Ungrouped or Selectively Grouped methods, each VHDL file that contains the parent entity of a block must conform to a set of rules. Other VHDL files do not need to follow these rules.

**NOTE:** In a Fully Ungrouped compile, every VHDL file is also a block, and therefore contains the parent entity of a block. This is not the case in a Selectively Grouped compile, where some VHDL files may not contain the parent entity of any block.

**Rule 1:** Except for the top-level VHDL file, the parent entity of each block must have the same name as the VHDL file that contains it. This is required in order for the Xilinx Design Manager or the XNFMERGE program to correctly merge the design files.

**Rule 2:** A single VHDL file may not contain the parent entity of more than one block.

**Rule 3:** When using the Fully Ungrouped method, a VHDL file may not contain child entities required by another VHDL file (the exception is that the parent entity of one VHDL file can be instantiated as a child in another file). Use the Selectively Grouped method to group these child entities with the file containing the parent.

## Libraries

### What is a User Library?

In the Foundation XVHDL environment, a user library is a VHDL file that is referenced by another file in a LIBRARY statement. A user library can contain packages and/or entities. User libraries can be located in either the current project directory, or in <ACTIVE>\VHDL\VHDL\_LIB, where <ACTIVE> is the directory where the Foundation tools have been installed (default is C:\ACTIVE).

User libraries are declared and used just like system libraries such as IEEE. For example, to access the entities defined in the library MYLIB.VHD use the following syntax:

```
LIBRARY mylib;  
USE mylib.ALL;
```

### What is a Library Alias?

Normally, when a library has been included in a VHDL file, XVHDL looks for a single file with the same name as the library. For example, if a library called MYLIB is declared in TOP.VHD, the compiler looks for a file called MYLIB.VHD when TOP.VHD is synthesized.

A library alias allows users to map a library name to one or more VHDL files that may have different names.

To create a library alias in the Foundation XVHDL environment, create an empty VHDL file in the <ACTIVE>\VHDL\VHDL\_LIB directory. Use the Synthesis→Options→Library Alias dialog box to assign VHDL files to the alias.

## Simulation

When using the Fully Ungrouped or Selectively Grouped compilation methods, the final result is multiple files. The Foundation Logic Simulator expects XVHDL to produce a single file for the entire VHDL design. Therefore, only the top-level file is loaded for functional simulation, resulting in an incomplete simulation netlist.

To create a complete functional simulation netlist from multiple XNF files, perform the following steps:

1. If the design is targeting an XC3000 family device, select Synthesis→Options and uncheck the IMPROVEX checkbox before synthesizing each block.
2. Open a DOS session and CD into the project directory.
3. Type the following command: `XNFMERGE <project>.J`, where `<project>` is the project name (always use the project name, even if the top-level VHDL file has a different name).
4. Type the following command: `COPY <project>.XFF <project>.XAS.J`. When prompted if you want to overwrite the existing file, answer `Y` (the XAS file is only used during functional simulation, so overwriting it will not affect place&route or timing simulation).
5. Exit the DOS session.
6. In the Foundation Project Manager, click on the Functional Simulation button (Figure 7).
7. If the design is targeting an XC3000 family device, remember to re-synthesize the design with the IMPROVEX option turned on before proceeding to design implementation.



Figure 7: Functional Simulation Button

## Design Flows

### Fully Grouped Compile

To prepare a Foundation XVHDL design for a Fully Grouped compile, use the Document→Add command in the Project Manager to add all of the VHDL files to the project.

Open the top-level VHDL file. Select Synthesis→Options. In the General tab, select Chip. In the Advanced tab, select the top-level entity and architecture and click OK. Select Synthesis→Synthesize to compile the design and create an XNF file.

### Fully Ungrouped Compile

To use the Fully Ungrouped compilation method, the VHDL files must follow the rules outlined in the *Design Issues: Entities* section.

To prepare a Foundation XVHDL design for a Fully Ungrouped compile, use the Document→Add command in the Project Manager to add only the top-level VHDL file to the project.

Open the top-level VHDL file. Select Synthesis→Options. In the General tab, select Chip. In the Advanced tab, select the top-level entity and architecture. Select Synthesis→Synthesize to compile this file as the top-level. Ignore any warnings about “No Entity bound to this instance”.

Select File→Open. Open one of the lower-level VHDL files. Select Synthesis→Options. In the General tab, select Macro. In the Advanced tab, select the highest level entity and architecture and click OK. Select Synthesis→Synthesize to compile this file as a macro. Repeat this procedure for each lower-level VHDL file in the design.

## Selectively Grouped Compile

There are two methods that can be used to perform a Selectively Grouped compile. The first method uses user-defined libraries, and the second uses library aliases. Refer to the *Design Issues: Libraries* section for information on user libraries and library aliases.

To use the Selectively Grouped compilation method, the VHDL files must follow the rules outlined in the *Design Issues: Entities* section. Use the tips in the *Partitioning Guidelines* section to decide how you want to group VHDL files into blocks.

To prepare a Foundation XVHDL design for a Selectively Grouped compile, use the Document→Add command in the Project Manager to add only the top-level VHDL file to the project.

### Selectively Grouped Compilation Method 1

For each block, open the VHDL file that contains the parent entity of the block. Declare and use all of the other VHDL files in the block as user libraries. Select Synthesis→Options. In the General tab, select Chip if this is the top-level block, otherwise select Macro. In the Advanced tab, select the parent entity and architecture and click OK. Select Synthesis→Synthesize to compile the block.

### Selectively Grouped Compilation Method 2

Create an file called CHILDREN.VHD in the <ACTIVE>\VHDL\VHDL\_LIB directory. This file can be empty, or it can contain text.

For each block, open the VHDL file that contain the parent entity of the block. Declare and use the library CHILDREN. Select Synthesis→Options. In the General tab, select Chip if this is the top-level block, otherwise select Macro. In the Library Alias tab, select CHILDREN and click Add. Browse to the project directory and use CTRL-click to select all of the other VHDL files in the block. Click OK to add the files to the library alias. In the Advanced tab, select the parent entity and architecture and click OK. Select Synthesis→Synthesize to compile the block.

## Example Design

The example design contains 4 VHDL source files: TOP.VHD, MID1.VHD, MID2.VHD, and BOTTOM.VHD. Each file is listed below (Figures 8-11).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY top IS
    PORT (sysclk: IN std_logic;
          enables: IN std_logic_vector (4 DOWNTO 0);
          mux_sel: IN std_logic_vector (1 DOWNTO 0);
          muxout: OUT std_logic_vector (3 DOWNTO 0);
          match: OUT std_logic );
END top;

ARCHITECTURE top_arch OF top IS
    COMPONENT mid1
        PORT (clk, cen1, cen2: IN std_logic;
              match: OUT std_logic);
    END COMPONENT;
    COMPONENT mid2
        PORT (clk, cen1, cen2, cen3, cen4: IN std_logic;
              sel: IN std_logic_vector (1 DOWNTO 0);
              muxout: OUT std_logic_vector (3 DOWNTO 0) );
    END COMPONENT;
    SIGNAL match_int: std_logic;
BEGIN
    U1: mid1 PORT MAP (clk => sysclk, cen1 => enables(0), cen2 => enables(1),
                      match => match_int);
    U2: mid2 PORT MAP (clk => sysclk, cen1 => enables(2), cen2 => enables(3),
                      cen3 => enables(4), cen4 => match_int, sel => mux_sel,
                      muxout => muxout);
    match <= match_int;
END top_arch;
```

**Figure 8: TOP.VHD File**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY mid1 IS
    PORT (clk, cen1, cen2: IN std_logic;
          match: OUT std_logic );
END mid1;

ARCHITECTURE mid1_arch OF mid1 IS
    COMPONENT bottom
        PORT (clk, ce: IN std_logic;
              q: INOUT std_logic_vector (3 DOWNTO 0)) ;
    END COMPONENT ;
    SIGNAL a, b: std_logic_vector (3 DOWNTO 0);
BEGIN
    U1: bottom PORT MAP (clk => clk, ce => cen1, q => a);
    U2: bottom PORT MAP (clk => clk, ce => cen2, q => b);

    -- Assert match when both counters are at F
    match <= '1' WHEN (a = "1111") AND (b = "1111") ELSE '0';
END mid1_arch;
```

**Figure 9: MID1.VHD File**



```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY mid2 IS
    PORT (clk, cen1, cen2, cen3, cen4: IN std_logic;
          sel: IN std_logic_vector (1 DOWNTO 0);
          muxout: OUT std_logic_vector (3 DOWNTO 0) );
END mid2;

ARCHITECTURE mid2_arch OF mid2 IS
    COMPONENT bottom
        PORT (clk, ce: IN std_logic;
              q: INOUT std_logic_vector (3 DOWNTO 0)) ;
    END COMPONENT ;
    SIGNAL a, b, c, d: std_logic_vector (3 DOWNTO 0);
BEGIN
    U1: bottom PORT MAP (clk => clk, ce => cen1, q => a);
    U2: bottom PORT MAP (clk => clk, ce => cen2, q => b);
    U3: bottom PORT MAP (clk => clk, ce => cen3, q => c);
    U4: bottom PORT MAP (clk => clk, ce => cen4, q => d);

    PROCESS (sel, a, b, c, d)    -- 4:1 mux
    BEGIN
        CASE sel IS
            WHEN "00" => muxout <= a;
            WHEN "01" => muxout <= b;
            WHEN "10" => muxout <= c;
            WHEN "11" => muxout <= d;
        END CASE;
    END PROCESS;
END mid2_arch;

```

**Figure 10: MID2.VHD File**

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY synopsys;
USE synopsys.std_logic_unsigned.ALL;

ENTITY bottom IS
    PORT (clk, ce: IN STD_LOGIC;
          q: INOUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
END bottom;

ARCHITECTURE bottom_arch OF bottom IS
BEGIN
    PROCESS (clk)    -- 4 bit counter with clock enable
    BEGIN
        IF clk'event AND clk='1' THEN
            IF ce='1' THEN
                q <= q + 1;
            END IF;
        END IF;
    END PROCESS;
END bottom_arch;

```

**Figure 11: BOTTOM.VHD File**

## Fully Grouped Example

To prepare the project for a Fully Grouped compile:

1. In the Project Manager, select Document→Add.
2. Use the list box to list files of type HDE.
3. Select all of the .VHD files and click OK.

To compile the design:

1. Double-click on TOP.VHD to start the HDL Editor.
2. Select Synthesis→Options.
3. In the General tab, select Chip.
4. In the Advanced tab, use the list boxes to select TOP as the top-level entity and TOP\_ARCH as the top-level architecture.

5. Click OK.
6. Select Synthesis→Synthesize.

This will produce a file named EXAMPLE.XNF, which contains all the logic described in TOP.VHD, MID1.VHD, MID2.VHD, and BOTTOM.VHD.

## Fully Ungrouped Example

To prepare the project for a Fully Ungrouped compile:

1. In the Project Manager, select Document→Add.
2. Use the list box to list files of type HDE.
3. Select TOP.VHD and click OK.

To compile the top-level file:

1. Double-click on TOP.VHD to start the HDL Editor.
2. Select Synthesis→Options.
3. In the General tab, select Chip.
4. In the Advanced tab, select TOP as the parent entity and TOP\_ARCH as the architecture.
5. Click OK.
6. Select Synthesis→Synthesize.

This will produce a file named EXAMPLE.XNF, which contains only the logic described in TOP.VHD (the I/O pins and a description of the connections between the MID1 and MID2 blocks).

During synthesis, messages saying “No Entity bound to this instance” will appear. These warnings can be ignored, because the lower-level entities will be compiled separately, and merged into the top level with XNFMERGE.

To compile the lower-level files:

1. Select File→Open and open MID1.VHD.
2. Select Synthesis→Options.
3. In the General tab, select Macro.
4. In the Advanced tab, use the list boxes to select MID1 as the parent entity and MID1\_ARCH as the architecture.
5. Click OK.
6. Select Synthesis→Synthesize.
7. Repeat steps 1-6 for MID2 and BOTTOM.

The final result is 4 files: EXAMPLE.XNF, MID1.XNF, MID2.XNF, and BOTTOM.XNF. The XACTstep Design Manager will merge these files together when Design→Translate is run.

## Selectively Grouped Examples

To prepare the project for a Selectively Grouped compile:

1. In the Project Manager, select Document→Add.
2. Use the list box to list files of type HDE.
3. Select TOP.VHD and click OK.

### Method 1 Example

The design will be partitioned as follows:

Block 1: TOP.VHD and MID1.VHD and BOTTOM.VHD

Block 2: MID2.VHD and BOTTOM.VHD

To compile Block 1:

1. Double-click on TOP.VHD to start the HDL Editor.
2. After the IEEE library declaration, add the following lines:  

```
LIBRARY mid1;  
USE mid1.ALL;  
LIBRARY bottom;  
USE bottom.ALL;
```
3. Select Synthesis→Options.
4. In the General tab, select Chip.
5. In the Advanced tab, select TOP as the parent entity and TOP\_ARCH as the architecture.
6. Click OK.
7. Select Synthesis→Synthesize.

To compile Block 2:

1. Select File→Open and open MID2.VHD.
2. After the IEEE library declaration, add the following lines:  

```
LIBRARY bottom;  
USE bottom.ALL;
```
3. Select Synthesis→Options.
4. In the General tab, select Macro.
5. In the Advanced tab, select MID2 as the parent entity and MID2\_ARCH as the architecture.
6. Click OK.
7. Select Synthesis→Synthesize.

The final result is 2 files: EXAMPLE.XNF and MID2.XNF. The XACTstep Design Manager will merge these files together when Design→Translate is run.

### Method 2 Example

The design will be partitioned as follows:

Block 1: TOP.VHD

Block 2: MID1.VHD and BOTTOM.VHD

Block 3: MID2.VHD and BOTTOM.VHD

To create a library alias called CHILDREN:

1. Open a text editor such as Notepad or Edit.
2. Type a space, or some text like:  

```
-- This file is used to create the library alias CHILDREN.
```
3. Save the file as <ACTIVE>\VHDL\VHDL\_LIB\CHILDREN.VHD.

To compile Block 1 (note that this is the same as a Fully Ungrouped compile):

1. Double-click on TOP.VHD to start the HDL Editor.
2. Select Synthesis→Options.
3. In the General tab, select Chip.
4. In the Advanced tab, select TOP as the parent entity and TOP\_ARCH as the architecture.
5. Click OK.
6. Select Synthesis→Synthesize.

To compile Blocks 2 and 3:

1. Select File→Open and open MID1.VHD.
2. After the IEEE library declaration, add the following lines:  

```
LIBRARY children;  
USE children.ALL;
```
3. Select Synthesis→Options.
4. In the General tab, select Macro.
5. In the Advanced tab, select MID1 as the parent entity and MID1\_ARCH as the architecture.
6. In the Library Alias tab, select CHILDREN from the list of libraries and click Add.
7. Browse to the project directory and select BOTTOM.VHD. Click OK to add the file to the library alias.
8. Click OK.
9. Select Synthesis→Synthesize.
10. Repeat steps 1-9 for MID2.VHD.

The final result is 3 files: EXAMPLE.XNF, MID1.XNF, and MID2.XNF. The XACTstep Design Manager will merge these files together when Design→Translate is run.