

Introduction

A major benefit of programmable logic is that it accommodates changes to the system specification late in the design cycle. In a typical engineering project development cycle, the specification for the programmable logic portion is likely to change when engineering development begins or when all system elements are being integrated.

These last-minute design changes are commonly referred to as engineering change orders (ECOs). ECOs are defined as small changes to the functionality of a design, after the design has been fully compiled, i.e., synthesis and place-and-route are completed.

ECOs are usually intended to correct errors found in the programmable logic design during debugging, or after changes that are made to the design specification to compensate for design problems in other areas of the system design. The operation of the system design cannot easily be changed in these areas.

As the project nears completion, a significant amount of time and effort has been invested in achieving timing closure in the programmable logic device (PLD). It is crucial that the programmable logic design flow is optimized to support ECOs in an efficient manner.

Impact of Last Minute Design Changes

ECOs have an impact on the following areas of a system design:

- Performance
- Compile time
- Verification
- Documentation

Performance

When a small change is made to the design functionality, it can result in previous design optimizations being lost. Typical examples of design optimizations are floorplan optimizations and physical synthesis. Ideally, there should be a means to preserve the design optimizations that has already been made. This will focus future optimizations that might be made to the design on the areas of the design to which the ECO changes were applied.

Compile Time

In the traditional programmable logic design flow, a small change in the design results in a complete recompilation of the design, i.e., synthesis and place-and-route. Thus, the process of making small changes to the design in order to reach the final implementation on a board can be a very long process. Ideally, in order to reach the desired functionality and to reach timing closure, a small change in functionality should result in a reduced compilation time. This can be achieved using incremental compilation technology that uses the previous fit information on the areas of the design that have not been affected by the ECOs.

Verification

After any design change, the impact of the change on the design must be verified. This verification is achieved through timing analysis and simulation. You can choose to limit the verification to the area of the design that is impacted by the ECOs. This is accomplished by running timing analyses on select paths and having the option to perform simulation on gate level and timing simulation netlists.

Documentation

Changes to the project files must be tracked. This helps other users reproduce the results at a later date. Ideally, you should be able to have multiple compilation revisions, so that others can try out changes without corrupting the results that have been previously obtained.

ECO Support

ECOs can be applied at either of two stages of a typical design flow:

- HDL level
- Netlist level

Traditionally in programmable logic design, ECOs have been applied at the HDL level. This is because the tools to easily create ECOs and to enable design sign-off at the netlist level have generally not been available for PLDs.

ECO Support at the HDL Level

An ECO at the HDL level is a small incremental change to the design's Verilog or VHDL source. This change may range from a single line to several lines of code modified within a module or entity. Typical examples of such modifications are:

- Changes to the state encoding of a finite state machine
- Addition of pipeline registers to improve design performance
- Signal duplication to reduce fan-out
- Adding a term to a conditional expression
- Changing the polarity of register control signals

A few changes to the source code can produce many changes to the netlist produced by other EDA synthesis or tools the Quartus® II software's integrated synthesis. During the synthesis process, the synthesis tools generally preserve the names of registers from the HDL source code, but automatically generate names for the combinational (look-up table level) nodes. This automatic name generation is necessary to accommodate the synthesis optimization performed on the HDL source to use the target device resources more efficiently.

Thus, a minor source code change can result in a many changes to the names in the synthesis netlist. The changes in the synthesis netlist can be due to either of two reasons:

- The node names in the new netlist implement different functionality than in the previous netlist
- The node names in the new netlist implement the same functionality as in the previous netlist, but have different names

In order to leverage the previous design optimizations and to reduce the compilation time, there must be a means of performing an incremental compilation on the nodes with the new functionality and preserving the previous optimizations on the nodes that have not changed. Thus, a means of identifying nodes that maintain the same functionality but have different names is essential for providing an ECO flow that truly works. Such a solution is provided with the incremental fitting feature available in the Quartus II software version 4.0.

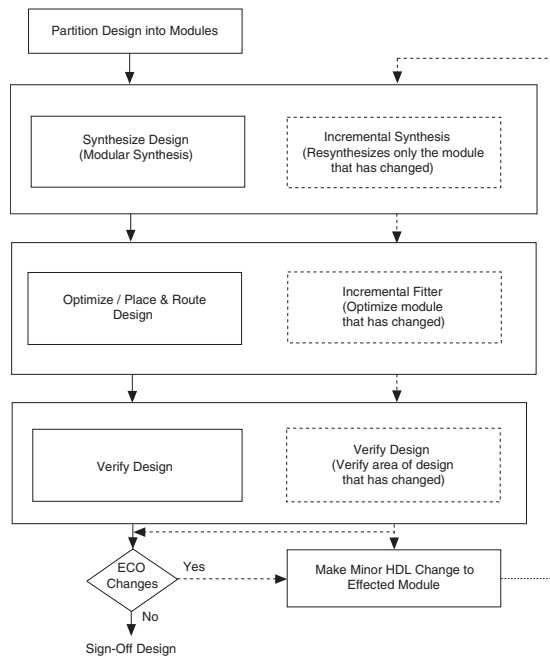
The Quartus II software version 4.0 incremental fitting feature performs a comparison between the original synthesis netlist and the new netlist containing the ECO changes. It matches nodes based upon names, functionality, fan-in, and fan-out. Those nodes that can be matched inherit the assignments from the previous fit.

Thus, the incremental fitting feature can preserve existing fitting information and timing. This feature limits any timing and fitting changes to the logic that has changed in functionality and reduces the compilation time.

In order to limit the changes caused by ECOs, it is recommended that users adopt a modular design flow. A modular design flow combined with the incremental compilation features mentioned previously

minimizes the changes in performance caused by ECOs and also reduces compilation time. Partitioning the design to adopt a modular design flow facilitates the placing of each module in the floorplan for performance. The Quartus II software provides the LogicLock™ feature to optimize the floorplan of modular designs. The *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook* describes how to apply the LogicLock methodology to a modular design flow. Figure 5–1 details the recommended design flow to support ECO changes at the HDL level.

Figure 5–1. Design Flow to Support ECO Changes



ECO Support at the Netlist Level

For certain ECO changes, it can be quicker to make changes at the netlist level rather than at the HDL level. This happens when you are debugging the design on silicon and need a very fast turnaround in generating a programming file for debugging the system.

A typical application occurs when you uncover a problem on the board and isolate the problem to the appropriate nodes or I/O cells on the PLD. You then need to be able to quickly correct the functionality of the

offending logic cell or the properties of the I/O cell and generate a new programming file in minutes. In doing this, you can verify the operation of the change without having to modify the HDL, and performing a synthesis and place-and-route operation. This minimizes the disruption to the board verification procedure.

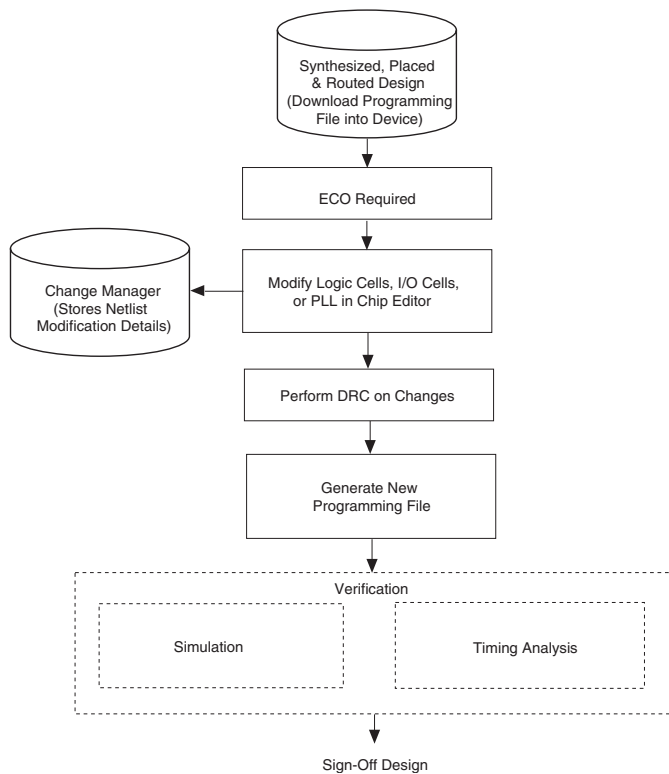
If this quick fix works, you do not need to change the HDL source code and rerun place-and-route. You should have the option to:

- Document the change that has been made
- Easily recreate the steps taken to produce the changes to the design
- Generate EDA simulation netlists for verification of the design
- Perform timing analysis on the design

These capabilities are provided in the Chip Editor feature of the Quartus II version 4.0 software.

The Quartus II Chip Editor allows you to make functional changes to individual logic cells and to the I/O cell and phase-locked loop (PLL) parameters. These changes are stored in the Quartus II Change Manager log. This allows you to control the application of the changes, and more importantly, to generate a tool command language (Tcl) file. This Tcl commands file recreates the changes on the original netlist. This Tcl file documents the changes made to the project and enables you to recreate the changes on the original design files at a later date, without having to change the HDL source. You can regenerate an EDA simulation netlist for the modified design, if it is necessary to perform a gate-level simulation of the modified design. If the designer needs to rerun timing analysis to sign-off the design, timing analysis can be rerun on the netlist containing the ECO changes. [Figure 5-2](#) shows the flow for ECO changes at the netlist level.

Figure 5–2. Design Flow for ECO Changes at the NetList Level



Conclusion

Support for ECOs requires a combination of a modular design methodology and the appropriate software design tools.

The Quartus II software version 4.0 provides you with the software tools and the design methodology to successfully perform ECOs at both the HDL and netlist level for programmable logic designs. This reduces the design cycle time and provides faster timing closure on designs that require last minute changes.