A Real-Time RTL Engineering-Change Method Supporting On-Line Debugging for Logic-Emulation Applications

Wen-Jong Fang¹, Allen C.-H. Wu¹, and Ti-Yen Yen²
¹Department of Computer Science, Tsing Hua University
Hsinchu, Taiwan, 300, Republic of China
²Quickturn Design Systems, Inc., 440 Clyde Avenue
Mountain View, California, 94043-2232, U.S.A

Abstract
In recent years, logic emulation has been widely used as a key design verification methodology in many complex CPU, telecom, and multimedia design projects. When using logic emulation for design verification, designers often need to perform engineering changes as a result of design debugging of a design specification modification. One of the essential issues to engineeing changes is the turn-around time. Ideally, after designers modify their designs, they resume their debugging and verification tasks immediately. However, converting a design from its Register-Transfer-Level (RTL) description to a target emulator is a time-consuming procedure which may take hours. Such long engineering-change turn-around times are unacceptable by the designers. In this paper, we present a real-time RTL engineering-change method supporting on-line debugging for logic-emulation applications. We propose a novel design method which is able to link design data generated at different design stages in a unified way. Using this method, the users can immediately locate the portion of the circuit design affected by the design modification from its RTL specification. This feature provides users with a fast time-to-debug environment by significantly improving the efficiency of the engineering-change process. We have developed a prototype system Quick_ECO supporting interactive on-line RTL engineering changes. Experimental results on a number of industrial designs are reported to demonstrate the effectiveness of the proposed method.

1 Introduction
Because of their reprogrammability, Field Programmable Gate Arrays (FPGAs) have become the most popular Application-Specific Integrated Circuits (ASICs) for rapid system prototyping. In addition, the development of reconfigurable systems by integrating FPGAs and Field Programmable Interconnect Chips (FPICs) has become the new trend in rapid prototyping and computation-intensive applications [1, 2, 3, 4, 5].

Logic emulation is the first technique to emerge that uses dynamically reprogrammable systems for prototyping and design verification [1, 4, 6]. A logic emulator is a system, consisting of hundreds to thousands of FPGAs and FPICs, which is able to realize designs through a software configuration procedure. A configured logic emulator is equivalent to the chip under design, and can be used for real-time design verification, software development, and prototyping before chip fabrication. Using logic emulation, designers can run design verification almost six orders-of-magnitude faster than classical software-based simulation [7]. In addition, designers are also able to execute complex system-level verification tasks such as running correctness tests, booting an operating system, and running application programs before tapeout. In recent years, many studies [4, 7, 8] have demonstrated that logic emulation is a very effective methodology for shortening the time-to-market of the design. Because of these advantages, logic emulation has been widely used as a key design verification methodology in many complex CPU, telecom, multimedia, and system design projects.

In general, a typical logic-emulation design process consists of two phases: (1) pre-configuration prepara-
ion and (2) full-chip configuration [8], as depicted in Figure 1. In the pre-configuration preparation phase, designers usually work on the RTL descriptions of designs. Gate-level descriptions of designs may not be available at this time. For emulation purposes, designers need to create a functionally-equivalent gate-level description, which will be replaced later by the actual gate-level designs from the design team. A common practice to convert an RTL description into a gate-level description is applying High-level Descriptive Language (HDL) synthesis tasks. In such a practice, designers first describe a design as a set of interconnected modules or sub-designs at the Register Transfer Level (RTL) in HDLs such as VHDL and Verilog. Then, designers can use a commercial or an in-house synthesis tool to perform a series of synthesis tasks, including RTL synthesis, logic synthesis, and FPGA-based mapping, to convert the HDL description of a design into a flattened CLB-based netlist.

In the full-chip configuration phase, designers convert the CLB-based netlist of the design into a set of bit-stream files which can be downloaded into the target emulator. The full-chip configuration phase consists of four steps [8, 9]: (1) partitioning, (2) system mapping, (3) FPGA placement-and-routing, and (4) design downloading (Figure 1). In the first step, a partitioner decomposes the CLB-level netlist into a set of subnetlists such that each subnetlist can be realized using an FPGA chip. In order to resolve the timing violations – especially the hold-time violations [9] – a very complex timing analysis procedure must be applied for path-delay analysis of the design before performing the partitioning. In the second step, a system mapper first assigns the subnetlists into the FPGAs on the system board and then performs system routing to realize the interconnections between the FPGAs. In the third step, a placer-and-router performs FPGA-based placement-and-routing tasks for each subnetlist. Finally, the design is converted into a set of bit-stream files and downloaded into the target emulator. Considering that a typical medium-sized design ranges between a hundred thousand to a half-million gates and a large design may have over 1 million gates, converting designs in such a large scale into a logic emulator is an extremely complex problem requiring a tremendous computational effort. Even for a medium-sized design, several hours are needed to convert the design into a logic emulator. As a result, shortening the Time-To-Emulation (TTE) is always the main concern in the logic-emulation design process.

When using logic emulation for design verification, designers often need to perform engineering changes as a result of design debugging of a design specification modification. One of the essential issues for engineering changes is the turn-around time. Ideally, after designers modify their designs, they resume their debugging and verification tasks immediately. However, a logic-emulation design process is a multi-phase design process which involves a series of design optimization and transformation tasks. A minor change on the design may require re-running a series of design tasks or even re-running the entire design process. For a medium-sized design, it may take several hours to complete a design update. Such a long Time-To-Debug (TTD) process is unacceptable to designers.

In this paper, we present a real-time RTL engineering-change method supporting on-line debugging for logic-emulation applications. We propose a novel design method which is able to link design data generated at pre-configuration preparation and full-chip configuration stages in a unified way. Using this method, the users can immediately identify the portion of the circuit design affected by the design modification. This feature provides users with a fast time-to-debug environment by significantly improving the efficiency of the engineering-change process. We have developed a prototype system Quick_ECO supporting interactive on-line RTL engineering changes. Experimental results on a number of industrial designs are reported to demonstrate the effectiveness of the proposed method.

The rest of the paper is organized as follows. Section 2 describes the motivation. Section 3 presents the design method supporting real-time RTL engineering changes. Section 4 describes the Quick_ECO system. Section 5 presents an evaluation of the proposed method. Finally, section 6 gives concluding remarks.

2 Motivation

As indicated in [8], at the early design emulation stage, designers are usually working with RTL descriptions which can be used for design simulation directly, or synthesized into functionally-equivalent gate-level descriptions for emulation purposes. Hence, engineering changes often take place at the RTL descriptions of designs in the early design emulation stage.

Engineering changes on the RTL descriptions often change the functionality and structure of the designs which may require re-running the logic emulation design process. We should first investigate the effects on the logic-emulation design process when applying engineering changes to RTL descriptions. Figure 2 depicts the topological relationships among the design data at different logic-emulation design stages. After the pre-configuration preparation phase, the HDL description of a design is converted to a CLB netlist Netlist1. In the partitioning stage, the netlist is partitioned into three clusters Cluster1, Cluster2, and Cluster3. In the system mapping stage, each cluster is mapped to an FPGA chip. For example, in Figure 2, Cluster1, Cluster2, and Cluster3 are assigned to Chip1, Chip2, and Chip3, respectively.

When an engineering change is applied to the HDL description in Figure 2, the following steps are performed. In the first step, a multi-phase re-synthesis procedure, including RTL synthesis, logic synthesis, and
FPGA-based technology mapping, needs to be applied to convert the updated HDL description into a CLB-level design Netlist'. In the second step, the differences between the original netlist Netlist1 and the updated netlist Netlist1* need to be extracted. For example, in Figure 2, the two actions required to make the two netlists identical are: (1) deleting node n1 and (2) adding node n2. In the third step, a re-mapping procedure is applied to re-arrange the configuration of clusters according to the differences between the two netlists. For example, node n1 should be deleted from Cluster2 and n2 needs to be assigned to one of the clusters. Finally, the FPGA place-and-route procedure is invoked to re-run the chips affected by the engineering changes.

One of the crucial tasks in the above engineering-change process is to differentiate the original and updated netlists. In general, there are two possible ways to extract the differences between these two netlists: (1) name matching and (2) graph matching. The name matching method compares the names of nodes and nets in the netlist to differentiate the two netlists. However, modifying RTL descriptions often changes the functionality as well as the structure of the designs. It is very difficult for a synthesis system to generate a consistent naming convention for both the original and its updated design. Hence, applying the name matching method often results in perceived large differences between the original and updated netlists. This will increase the complexity of the re-mapping process to the point where it may be equivalent to re-running the entire full-chip configuration process. On the other hand, the graph matching method is a non-trivial problem which needs to search the entire netlist. It is also an extremely time-consuming procedure for designs containing hundreds of thousand-s of components. Consequently, when using the above engineering-change procedure, designers often suffer a very long-waiting period to resume their design debugging and verification tasks.

In order to facilitate the design debugging process, a more efficient and more effective real-time RTL engineering-change methodology is required to support on-line debugging. Using such a debugging method, designers should be able to perform on-line hardware debugging in much the same way a C-code debugging tool is used for software debugging: the designer makes a change, recompiles the design, and resumes the debugging task in real time. This motivates us to investigate a high-efficiency RTL engineering-change method which can provide an on-line debugging environment for logic-emulation applications.

3 A real-time RTL engineering-change method

3.1 Considerations

There are two essential considerations to supporting real-time RTL engineering-change methods: (1) data linkage and (2) design granularity. The first one is the capability of maintaining the links between design data generated at different design stages. As described in the previous section, a logic-emulation design process is a multi-phase design process which performs a series of optimization and transformation tasks to convert an HDL description into the target emulator. If we can maintain the direct linkage between an HDL description, its corresponding netlist, and the chip configuration, the portions of the netlist and chips affected by an engineering change can be identified immediately without running a time-consuming procedure to differentiate the original and updated netlists. For example, Figure 3 depicts the topological relationships between the design data at different logic-emulation design stages. After synthesis, each module has a corresponding CLB netlist. For example, in Figure 3, modules Mod1, Mod2, and Mod3 correspond to Netlist1, Netlist2, and Netlist3, respectively. In the partitioning stage, the flattened netlist is partitioned into a set of clusters such that each cluster is realized using a single FPGA chip. Thus, each chip corresponds to a cluster or a subnetlist. For example, in Figure 3, Chip1, Chip2, and Chip3 correspond to Cluster1, Cluster2, and Cluster3, respectively.

When an engineering change is applied to the HDL description in Mod1, the following observations can be made. Only the portions of the design affected by Mod1 need to be updated. From the data linkages, we can identify that only Netlist1 is affected by the change. Since Netlist1 is divided into two clusters Cluster1 and Cluster2, two chips Chip1 and Chip2 are affected. To update the design, we first need to re-synthesize Mod1 to generate an updated netlist Netlist1* followed by deleting the old netlist Netlist1 from the clusters. As shown in Figure 3, n1, n2, and n3 are deleted from Cluster1, while n4 is deleted from Cluster2. Finally, a re-partitioning procedure is applied to partition the updated netlist Netlist1* into clusters. This engineering-change procedure demonstrates that by establishing the data relationships between different design stages, we are able to localize the circuit affected by the changes.

The second consideration is the granularity of the design. The complexity of the engineering-change process is proportional to the size of the design affected by engineering changes. For example, the above engineering-change procedure is localized by a module. When a change occurs, only the netlist and chips corresponding to the modified module will be affected by the change. In other words, the complexity of the engineering changes is proportional to the size of the module. A module may be a simple component such as an adder or a multiplexer, consisting of only tens to hundreds of gates. In this case, the module-based engineering changes will be very efficient and effective. However, in many industrial designs a module often contains an arbitrary complex function,
such as an MPEG algorithm or a floating-point multiplier, which may contain tens thousands of gates. In this case, performing engineering changes at the module level becomes a very complex and time-consuming process. Thus, a fine-grained design smaller than modules would be beneficial for speeding up the engineering-change process.

3.2 Overview of the design method

The main purpose of the proposed design method is to bridge the gap between the pre-configuration preparation and the full-chip configuration design stages (Figure 1). The proposed design method establishes direct links between HDL descriptions, the netlists, and the chips to provide an environment for real-time RTL engineering changes.

Figure 4 depicts the design flow. The input to the system is an RTL description described in Verilog. A fine-grained synthesis method is used to convert the input HDL description into a CLB netlist. In addition, the synthesizer also generates a structural tree containing the linkage information between the HDL description and its corresponding netlist. After the synthesis stage, the system performs partitioning, system mapping, FPGA place-and-route, and design downloading the same as the full-chip configuration process described in the previous section.

3.3 A hierarchical fine-grained synthesis method

The main objectives of the fine-grained synthesis method are twofold: (1) provide a fine-grained design for engineering changes and (2) establish the linkages between the constructs of an HDL description and their corresponding circuit designs. We use an HDL structural tree to represent the structural hierarchy of the Verilog description of a design. In an HDL structural tree, the root node represents the design, each intermediate node represents a higher level construct such as a module, a process, and a task, each leaf node represents an independent functional-block. An independent functional-block contains a set of statements having the same output signal and can be synthesized into an independent CLB design without affecting any other statements. We use the fine-grained synthesis method [10] to synthesize the design in a fine-grained way and generates an HDL structural tree. For example, Figure 5 shows a design example and its corresponding structural tree.

Using the hierarchical fine-grained synthesis method, we are able to establish direct links between the constructs in an HDL description and their corresponding netlists and chips. This feature provides us with a fast way to distinguish the design affected by the engineering changes. Furthermore, the fine-grained synthesis method also provides a fine-grained design environment which allows the user to perform engineering changes with different granularities of the design; i.e., at module, process, and statement levels.

3.4 Required features supporting interactive engineering changes

From a user's point of view, an interactive engineering-change system should provide two main features: (1) the user should be able to modify an HDL description on-line and (2) the user should have control over re-synthesizing and re-mapping any portions of the design. Using the system, the user can first use an editor to change an HDL description. Then, the user can evaluate the change to determine the portions of the design affected by the change. Finally, the user can re-synthesize and re-map the portions of the design.

To support on-line RTL engineering changes, an interactive system must also provide graphical views into the design, visually describing the potentially complex relationships between an HDL description and its corresponding netlists, and allowing the user to modify the design on-line and immediately perceive the consequences of his or her decisions. From a user's perspective, the system should provide users four design views: (1) an HDL view, (2) a structural view, (3) a netlist view, and (4) a CDFG view. The HDL view captures the RTL description. The structural-tree view shows the hierarchical relationship between the HDL description and its structure. The netlist view displays the top structural-view and its CLB netlist. The CDFG view shows the control/data flow of the description which is very useful for design analysis during the debugging process. It is also necessary that all of the views required for a particular task coordinate closely together such that changes...
The Quick_ECO system

The Quick_ECO system needs an underlying data model which can combine all of the design data produced at multiple levels of abstraction into a single unified view. The data model must also provide a fast updating capability to support real-time engineering changes. We use a hybrid hierarchical graph data structure [12] to represent HDL structural trees, netlists, and chips. The relationships between data at different design stages are initiated by a grouping process. For example, leaf nodes and clusters (chip nodes) in the structural-tree, contain a set of components in the netlist, as shown in Figure 4.

4 The Quick_ECO system

We have developed an interactive synthesis system Quick_ECO for on-line RTL engineering changes. Figure 6 shows the structure of Quick_ECO consisting of four parts: the design manager, the database, the tool set, and the Graphical User Interface (GUI). The design manager is the kernel of the design system, which provides services to user requests, via the graphical user interface. The services include invoking a specific design task and tool and providing a specific design view.

The tool set contains a set of tools including a Verilog compiler (V Compiler), an RTL synthesizer (EmSyn), and a partitioner (EmPar). EmSyn interfaces to a set of logic minimization and technology mapping procedures and a component library. The input to the system is a Verilog description of the design. V Compiler performs HDL compilation and converts the Verilog design description into an intermediate design format. EmSyn performs RTL synthesis including unit selection, unit/storage/interconnect binding, and construction of an HDL structural tree. The component library provides the synthesizer with a set of generic RTL components to support the RTL synthesis. EmSyn invokes logic minimization and technology mapping procedures to convert a structural design into a CLB-based design in XNF and Verilog formats. EmPar is a partitioner which contains a set of partitioning algorithms [13, 14] for circuit partitioning.

The GUI is implemented in C with the OSF Motif graphics library [15, 16] and the X-Windows system running on SUN and HP workstations. The GUI consists of an on-line editor, a composite design view, and a CDFG view, as shown in Figure 7. The composite design view consists of four parts: a structural-tree view, a schematic view, an HDL view, and a top-level structural view. The user can modify an HDL description using an on-line editor and then analyze and resynthesize the nodes in one view will be reflected in others.

To provide for these requirements, an interactive system needs an underlying data model which can combine all of the design data produced at multiple levels of abstraction into a single unified view. The data model must also provide a fast updating capability to support real-time engineering changes. We use a hybrid hierarchical graph data structure [12] to represent HDL structural trees, netlists, and chips. The relationships between data at different design stages are initiated by a grouping process. For example, leaf nodes and clusters (chip nodes) in the structural-tree, contain a set of components in the netlist, as shown in Figure 4.

5 Experiments

We have tested our proposed method on four industrial designs, as shown in Table 1. The first design is a 32-bit floating-point multiplier. The second is a DSP core. The third design is a controller which contains a large control circuit and a small data path. The fourth is a communication chip. The number of lines of Verilog codes describing the designs range from 253 to 6,298. We have performed two synthesis methods, module-based and fine-grained methods on each design. For circuits produced using both synthesis methods, we have the same test-vector set to fully verify the correctness of the RTL spec and synthesized circuits. The equivalent gate-count of the designs ranges from 10,000 to 30,000 gates. Table 1 shows the results comparing module-based to fine-grained synthesis methods.

From the experiments, the following observations can be made. First, the fine-grained synthesis method produced designs with 7 to 167 process nodes and 44 to 5,148 leaf nodes. On the other hand, using the module-based synthesis method, the synthesized designs contain 1 to 36 module nodes. This demonstrates that the fine-grained synthesis method provides a much finer granularity of designs for engineering changes compared to that using the module-based synthesis method.

Second, the fine-grained synthesis method produced designs using more CLBs than that produced by the module-based synthesis method. Using the fine-grained synthesis method, the quality of designs is usually proportional to the number of the leaf nodes in the designs. Since each leaf node is synthesized into an independent circuit, a global optimization procedure will not be very effective on a design containing a large number of leaf nodes. In fact, it may result in a larger CLB design. For example, the design of Design 3 produced using the fine-grained synthesis method is 87% larger than that pro-
duced using the module-based synthesis method. However, despite these shortcomings, the design with a finer granularity provides a much greater degree of flexibility and efficiency for real-time engineering changes.

Third, as stated in [11], engineering changes should have the following characteristics: (1) locality and (2) small size. First, engineering changes should not completely change the functionality or structure of the design. Second, it should only affect small portions of the design. Under these assumptions, a minor change to an RTI design will only affect one leaf-node of the structural-tree and its corresponding netlist, which should drastically shorten the processing time for engineering changes. However, as we described earlier, engineering changes on the RTL descriptions often change the functionality and structure of the design. We found that only some simple modifications in an HDL description, such as changing the assigning values in a statement, will affect only one leaf-node in the structural tree. In many cases, adding a new statement into a process will change the functionality of the process. For example, designers may change the I/O ports of a process after adding some statements. This requires resynthesizing the process. Using the proposed method, the user is able to perform engineering changes at different design levels, either at module, process, or statement level.

6 Conclusions

In this paper, we have presented a real-time RTL engineering-change method and a prototyping system Quick_ECO supporting on-line debugging for logic-emulation applications.

The experiments on several industrial designs show that when using the fine-grained synthesis method a high degree of granularity in a design can be achieved. It provides a much greater degree of flexibility and efficiency for RTL engineering changes. However, it must also pay a penalty of design overhead in the size of the synthesized circuits. Nevertheless, it can significantly improve the efficiency of the engineering-change process which is especially beneficial to logic-emulation applications due to their extremely long design-processing time.

This is the first attempt to tackle the RTL engineering-change problem. Many open problems need to be studied further which include trading off the design overhead and the design granularity and developing a real-time engineering changing methodology by combining module-based and fine-grained synthesis methods for large logic-emulation applications.

References


