Chapter 8

Conclusion

This dissertation began with four objectives:

- investigate incremental placement algorithms to improve the design development cycle,
- develop and demonstrate a prototype of an incremental FPGA design tool that can be used to shorten the FPGA design cycle for million-gate devices,
- explore a background refinement mechanism to preserve design fidelity, and
- develop large designs to evaluate the incremental placement algorithm and the design tool.

With the end as a beginning, this chapter concludes this dissertation and reexamines all of the goals. Future directions are identified as well.

8.1 Goals Reexamined

8.1.1 Incremental placement algorithms

Focusing on the goal to improve the FPGA design and debug cycle, an incremental placement algorithm was implemented in Chapter 3. Considering the importance of design reuse in large-scale FPGA designs, pre-defined cores have been employed as the

design library in this placement algorithm. Those cores are processed as the functional unit and are placed incrementally at their relatively desired position with the minimal possible shifts of the existing designs; thus speeding up the FPGA design cycle especially as the gate counts increase to multi-millions.

A detailed processing flow has been presented. Three methods, namely the nearest position, recursive search, and force-directed position, have been examined and compared to find desired positions for shifted cores. The terminating condition processing techniques have also been discussed to ensure placement success rate.

The guided placement methodology as well as its exception handling techniques have been investigated to find changed parts in a design and to take advantage of the optimized design from previous iterations. Finally, cluster merge strategies have been discussed in Chapter 3 to complete this core-based guided incremental placement algorithm.

Different from other incremental placement algorithms, this algorithm uses pre-defined cores as the design elements, and places and moves a core without breaking its original shape. In addition, this algorithm can be employed not only for processing minor changes in a design, it can also be used to place a million-gate design from scratch. Aimed at significantly reducing the processing time, this core-based incremental placement algorithm is expected to accelerate the FPGA design-and-debug cycle with a speed orders-of-magnitude faster than the traditional FPGA design flow; thus providing the possibility of developing a more user-interactive integrated FPGA design-and-debug environment.

8.1.2 Incremental design IDE

To demonstrate the incremental placement algorithms implemented in Chapter 3, a prototype of an incremental integrated design environment was developed in Chapter 4. Following the procedure of object-oriented analysis and design, requirements of the system were analyzed first. Based on the obligation of low coupling and high cohesion in software engineering, the incremental design IDE was developed using three packages:

`DesignIDE`, `DesignArea` and `DesignTool`. The program structure, data structure and their implementations were discussed in Chapter 4. Dynamic linking techniques were also investigated to allow designers to build a design using the Java Language and compile the design with the standard Java compiler.

The developed design IDE reads a user-designed Java class as the input and creates a bitstream as the output. It then employs the incremental placement algorithm and the guided placement methodology to represent the design. JBits RTPCores are instantiated as the functional units, and JRoute is employed to route the designs after placement using the incremental techniques. The placed and routed design is then displayed on the simulated device platform and can be browsed from different views and scales. The functionality of the created bitstreams can be tested using Xilinx BoardScope software. This tool provides a design environment that simplifies the FPGA design-and-debug cycle, and allows designers to build their design using unmodified Java languages and Java compiler. This tool also presents JBits designers a platform that can be used to place, route, and generate the executable bitstreams automatically, therefore helping to exploit the potential of JBits toolkits.

8.1.3 Garbage collection mechanism

Garbage collection and background refinement techniques were investigated in Chapter 5 to overcome the native shortcomings of the incremental algorithm. A core-based simulated annealing placer was implemented as the background refiner of the incremental placer developed in Chapter 3. The properties of the simulated annealing placer and its advantages used as the background refinement thread have been analyzed. The long processing time and the high placement performance make the simulated annealing placer well suited as the background refiner to maintain the performance of the incremental placer. Running as a lower priority thread, the simulated annealing placer does not compete for CPU time with the incremental placer; thus ensuring the fast placement speed of the foreground incremental placer. More importantly, when the incremental placer finishes its work, it can still take advantage of the better solutions that the background refiner provides, and can use that as a guide template when more

elements are added and modifications are made. Local minima are thus avoided and the performance and robustness of the incremental placer are continued by employing the core-based simulated annealing placer as the background refiner.

8.1.4 Design evaluation

To achieve the last goal of this dissertation, a large number of design applications with logical gate sizes varying from tens of thousands to approximately one million were built in Chapter 6. The incremental placement algorithms, the design IDE, and the background refinement techniques were evaluated using these designs and the experimental results have been presented in Chapter 7.

Two sets of design examples were developed in Chapter 6. A random circuit generator was built to simulate a large number of real design circuits with a random number of randomly connected and randomly sized cores. A design class derived from real applications that compute large polynomials for data compression was also developed. All the design examples had been implemented on the Xilinx Virtex XCV300 and the XCV1000 FPGAs.

The performance of the incremental placement algorithms was assessed using different sizes of random circuits implemented on different devices in Chapter 7. The performance statistics demonstrate that the incremental design tool operates with 100% place-and-route success rate when the device utilization is below 80% for small block size circuits, and 65% for large circuits. It then decreases as the device utilization increases, and the performance is much better when the core sizes are small. The speed of the incremental placement algorithm was calculated and presented in Chapter 7. This placement algorithm can place a random circuit with about 100 randomly sized cores (with a mean core height of 10 and a mean core width of 5) in 1.25 seconds at the speed of 700k gates per second on a 1GHz PC. If routing is attempted for those placed circuits, they are all routed successfully using JRoute. Design sets derived from an actual application were also tested in Chapter 7. The experimental data demonstrates that all the polynomial computation circuits with degrees ranging from 3 to 23 have been successfully placed

using the incremental placement algorithms within 1 second and all the placed circuits have been successfully routed using the JRoute API.

The performance of the three incremental methods that were employed was also analyzed and compared using the place-and-route success rate, placement speed, and wire-length as metrics. The performance analysis showed that the nearest position method provided better performance than the recursive search and the force-directed method. Designers have the option to select any of the three methods during placement.

The functionality of the terminating refinement strategy was also evaluated in Chapter 7. It was shown from the statistics that the refined terminating condition improved the place-and-route success rate from 1% to 7.6% depending on the device and the mean size of the random cores.

The performance of the guided placement methodology was examined. It was found from the comparison that by applying the guided placement methodology, one could save placement time when the input design was not updated or only minor changes were found in the input design. When a design obtained from a traditional placer such as the simulated annealing placer was employed as the guide template, the guided placement methodology provided much better performance than using the guide design obtained from the incremental placement algorithms at almost the same processing speed. In addition, this methodology presented comparable placement performance with the traditional placer but orders-of-magnitude faster processing time if the guided template was chosen properly.

The performance of the methods that handle large-percentage changes and the exceptions occurring during the execution of the guided placement methodology were evaluated in Chapter 7. The experiment demonstrated that these methods had efficiently maintained the functionality and the performance of the guided placement methodology.

The functionality of the cluster merge mechanism was investigated. The experimental data showed that this mechanism reduced the effect of the order in which cores are added; thus preserving the performance of the incremental placement algorithm.

The polynomial computation design was implemented using both the incremental design tool and JHDL, and was synthesized by the incremental design techniques, the core-based simulated annealing placer, and the standard Xilinx M3 tool. The comparison showed that the core-based incremental placement algorithm was orders-of-magnitude faster than the simulated annealing and the Xilinx placer. It takes only 560 milliseconds to place a polynomial with degree 23 that uses about 74 percent of the resources on the Virtex XCV1000, while it requires 40 seconds for the simulated annealing placer and 92 seconds for the Xilinx placer. Based on the fast placement, the incremental FPGA design tool is more user-interactive than the Xilinx Foundation series 3.3i. To generate a circuit that computes a polynomial with degree 23, the entire procedure from loading the design to generating the bitstreams can be finished within 76 seconds in the incremental FPGA design tool where JRoute consumes up to 71 seconds of the total time. Compared to the traditional design flow, it takes about 25 minutes to load and netlist the design using JHDL, and another 40 minutes to complete the entire design procedure using the Xilinx M3 tool chain.

The incremental placement algorithm accelerates the placement procedure at a slight cost in placement fidelity. It requires about 1.3 times the resources of the Xilinx M3 tool. Several reasons that lead to the resource utilization discrepancy were explained in Chapter 7. The analysis showed that employing cores as the functional unit, different ways to interpret empty resource, and the density of the pre-defined cores were the main reasons that affect the resource inconsistency.

The incremental placer was also shown to provide placements with longer wire-lengths when compared to the simulated annealing placer, which was caused by the inherent disadvantages of the incremental algorithms. Thus, implementing a background refinement mechanism was shown to be essential. The advantages of the background

refiner were also indicated in Chapter 7. The experimental results proved that integrating the background refinement mechanism with the incremental placement algorithm improved the design performance by significantly decreasing the wire length.

In conclusion, incremental design techniques for improving the performance of million-gate FPGA designs have been presented in this dissertation. The results have shown that the incremental design techniques are orders-of-magnitude faster than traditional approaches such as techniques used in the standard Xilinx M3 tool chain. The algorithm's fast processing speed and user-interactive property make it potentially useful for prototype developing, system debugging, and modular testing in million-gate FPGA designs with little sacrificing in design quality. When combined with a background refiner, the incremental design tool offers the instant gratification that designers expect, while preserving the fidelity attained through batch-oriented programs.

8.2 Future Directions

8.2.1 Using standard design circuits

A large number of test circuits were generated in Chapter 6 to evaluate the performance of the algorithms and the design tool developed in this dissertation. Widely used standard test circuits, such as the MCNC standard circuits, cannot be directly employed as the test circuits in this dissertation because of their format incompatibility and small circuit sizes. Thus, the testing circuits used in this dissertation were hand-crafted. Tested with these hand-generated circuits, the incremental design tool presented orders-of-magnitude faster speed than other approaches, such as the Xilinx placer. Employing standard testing circuits is more convincing to other tool developers, and should be considered to further evaluate the performance of the incremental placement algorithms and to explore the potential use of this design tool.

With the rapid development of large scale FPGA design, widely used large-scale test circuits have been and will be developed. Future work is necessary to convert these circuits into an incremental design tool compatible format.

8.2.2. More investigation in resource utilization comparison

As exhibited in Chapter 7, the incremental design tool uses more resources than the traditional tool such as the Xilinx M3 tool chain. Three reasons were given to explain the discrepancy. The density of the JBits RTPCore was one of the reasons. Developing high-density functional units is one method that could improve the efficiency of the incremental design tool. Because core development is beyond the goals of this dissertation, this analysis is expected to help the JBits core developers to fine-tune their designs. Comparing the resource utilization using designs built with high-density cores is another task for further investigation.

8.2.3 Improving the core-based simulated annealing placer

Because the focus of this dissertation has been on the incremental placement methodology, the core-based simulated annealing placer implemented in this dissertation is only a prototype and the choice of the cost function and the cooling schedule can be improved to make the performance better. Implementing a core-based simulated annealing placer with better performance is another future direction suggested by this dissertation. We can predict from the current results that the incremental design tool will gain more benefits from the background refiner if a fine-tuned core-based simulated annealing placer is employed.

8.2.4 Future expectations

Improving the design-and-debug cycle is a consistent goal in design automation. It becomes more and more important as gate counts and design complexity increase. Incremental design is an efficient way to speed up FPGA processing time especially as the gate sizes increase to multi-millions. The incremental placement algorithms and the incremental design tool developed in this dissertation provides orders-of-magnitude faster speed than competing approaches, making the user-interactive design environment possible for million-gate FPGA designs. The incremental design techniques developed in this dissertation pave the way for a new design process in large-scale FPGA designs. This technique is expected to add value to real-world applications.

It is anticipated that the incremental technique developed in this dissertation will benefit problem domains other than FPGA design. Obviously, the technique can be efficiently adopted to improve the ASIC design and development cycle. Using the pre-optimized ASIC components as the design library, the concept of the incremental placement algorithm developed in this dissertation is also useful to accelerate ASIC synthesis and layout procedures. Besides the hardware design domain, other fields including computer graphics, high performance computing, and software engineering can also take advantage of this incremental technique. For example, an incremental polygonization technique could accelerate the overall polygonization process for complex objects, exploiting the potential for providing high quality video effects in motion pictures. Similarly, this technique can be extended to fields where a huge amount of data needs to be handled to reduce a processing cycle.