An Automatic Design and Implementation Framework for Reconfigurable Logic IP Core

Qian Zhao, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga and Toshinori Sueyoshi
Graduate School of Science and Technology, Kumamoto University

Abstract—Conventional full-custom reconfigurable logic device design and implementation are time consuming processes. In this research, we propose a design framework in order to improve FPGA IP core design efficiency by link academic FPGA design flow and commercial VLSI CADs based on the synthesizable method. A novel FPGA routing tool is developed in this framework, namely the EasyRouter. By using simple templates, EasyRouter can automatically generate the HDL codes and the configuration bitstream for an FPGA. With this design flow, accurate physical information can be reported when a new FPGA architecture is evaluated with reliable commercial VLSI CADs. For FPGA architectures that cannot be easily implemented with present VLSI process, EasyRouter provides a fast performance analysis flow, which improved delay accuracy 5.1 times than VPR on average.

1. Introduction

Embedded systems play an increasingly important part in electronic products. In particular, system-on-a-chip (SoC) technology has developed rapidly. A variety of functions can be implemented by embedding various hard intellectual property (IP) cores in a single silicon die. However, a new product must be fabricated with an entirely new mask. Even if only small changes are made to a product to improve functionality, a huge cost is incurred. The embedded field-programmable gate array (FPGA) IPs can be used to solve this problem because of their programmability after manufacture.

There are two FPGA IP implement methods. The full-custom FPGA IP is designed in time-consuming manually process. On the other hand, the synthesizable FPGA IP is designed with automatic application specified integrated circuit (ASIC) flow. In traditional designs, the synthesizable method had much worse area, delay and power performances than the full-custom. However, the performance gaps had been improved significantly in researches such as [1]. Therefore, synthesizable design method is suitable for design efficiency sensitive customizable FPGA IP implementation.

Xilinx and Altera have released their programmable SoC products [2] [3]. A powerful ARM-based processor and universal FPGA fabrics are integrated into one chip to reduce power, cost, and board size. However, the FPGA IP cores from these companies are not customizable and not provided to other SoC designers. Menta is providing domain-specific synthesizable and hard macro eFPGA core IPs [4]. However, Menta’s CAD tools are only designed for their commercial eFPGA IPs. Therefore, CAD tools and a design flow for FPGA IP research and design are necessary.

The contribution of this paper is to propose an FPGA design framework that specifically improves the design efficiency of FPGA IP for SoC. We have developed a simple and automatic FPGA IP design framework that combines FPGA design tools with commercial very-large-scale integration (VLSI) CADs. The FPGA IP that produced by the proposed framework can be directly adopted in SoC design flow as an IP core.

The remainder of this paper is organized as follows. Section 2 introduces related FPGA design flows and issues of traditional design flows. The novel router tool EasyRouter is introduced in Section 3. Section 4 describes the proposed FPGA IP design flow. In Section 5, we first introduce evaluation conditions. Then we compare the performance of EasyRouter with the conventional VPR and then discuss evaluation results for the proposed flow. Finally we show the simplicity and expandability of EasyRouter with a three-dimensional (3D) FPGA case study. Conclusions are given in Section 6.

2. Related Works

2.1 FPGA design CAD tools

Xilinx ISE and Altera Quartus are commercial CAD tools used to implement circuits on their FPGAs. On the other hand, open source design flows like Verilog-to-Routing (VTR) project [5] are used for academic FPGA researches. The VTR project consists of the placement and routing tool Versatile Packing, Placement and Routing (VPR) [11], the synthesis tool ODIN II [6], and technology mapping tool ABC [7]. VPR [11] is the CAD tool that directly related to the FPGA physical architecture.

Because VPR cannot be used for unsupported architectures, many other FPGA design frameworks have been developed for various devices. Grant et al. [8] employed a typical FPGA design flow together with a new placing, routing, and scheduling tool for their coarse-grained architecture. Ababei et al. [9] and Miyamoto et al. [10] proposed design flows for a 3D-FPGA. The authors of [9] developed their TPR on
the basis of VPR 4.0, while those of [10] used a modified VPR for 3D-FPGA.

2.2 Issues of traditional design flows

We now discuss two issues of VPR since it is directly related to the physical architecture of the FPGA.

First, the architecture-description-file based architecture definition method provides flexibility for various logic block structures. However, the flexibility of routing structure is still limited to the supported island style architectures. For much of our research, such as on a 3D-FPGA, we have to modify the VPR to implement various routing architectures. It consumes considerable development time to master, modify, and debug the C-coded VPR.

Second, the VPR is integrated with a simple delay model to facilitate timing-driven routing and post-routing timing analysis. The final timing report consists of the logic and routing delays, which are calculated in different ways. Therefore, although the relative values of VPR delay results can fairly evaluate FPGA architectures, the absolute value has low accuracy for synthesizable FPGA IP design, which requires an accurate entire chip static timing analysis (STA) with a standard cell library. Further, VPR does not provide any function that links FPGA design flow with commercial VLSI CADs.

3. EasyRouter

In this section, we introduce the proposed routing tool EasyRouter. Based on the similar routing and reporting functions of VPR, EasyRouter has some improved features. First, because we developed EasyRouter in C# language with full object-oriented programming coding style, the amount of code and complexity was reduced, making it easier to understand and modify. Owing to the benefits of the open-source Mono runtime environment, EasyRouter can be executed in most operating systems. Second, we developed a script-based architecture definition mechanism by considering the code file itself to be the architecture definition file. This mechanism offers users maximum flexibility in implementing new architectures. Finally, we developed HDL codes and bitstream generation functions to facilitate the evaluation of the designed FPGA using commercial VLSI CADs. The block diagram of EasyRouter is shown in Fig. 1. We now describe each of the blocks in detail.

3.1 RRGraph building block

The RRGraph describes the target FPGA architecture with routing resources (nodes) and their connection relationships [11]. We describe the RRGraph with a graph data structure, which is independent with any FPGA architecture. Each routing resource in the RRGraph is called an RRNode. The RRGraph is a collection of all necessary RRNodes.

As Fig. 1 shows, the RRGraph building block of EasyRouter reads the C# coded FPGA architecture script file to generate an RRGraph. The actual architectural dependent codes such as architecture and physical parameters setup, netlist and placement files import, and the RRGraph building are implemented in the RRGraph generation script files. The architecture and physical parameters setup block sets parameters of one FPGA architecture like the VPR architecture file does. New FPGA architecture can be implemented by modifying the RRGraph building codes of the script. The architecture script only returns architectural independent RRGraph to the routing block. The dynamic script support is implemented with the Dynamic Language Runtime (DLR) of the .net framework. With this feature, the FPGA architecture to be evaluated by EasyRouter can be changed by switching the RRGraph generation script input file. Therefore, new FPGA architecture can be implemented easily using the EasyRouter. And the architecture script is generic to implement various FPGA architectures. When evaluating many architectures, it is easy to switch between them without recompiling the main EasyRouter program.

3.2 Routing block

EasyRouter implements conventional breadth-first and timing-driven pathfinder routing algorithms [11]. Note that the timing-driven algorithm can improve delay of routing result when implementing customer circuits, however, it is not employed during the FPGA scale exploration phase because accurate physical delay information is unknown before the architecture implementation.

3.3 HDL codes and bitstream generation block

We developed EasyRouter using FPGA HDL codes and the user circuit configuration bitstream generation functions to link the academic FPGA design flow with the commercial VLSI CAD tools, since the routing algorithm stores a large amount of architecture information that can be used to generate HDL codes and bitstreams. As Fig. 3 shows, when
EasyRouter operates in the evaluation mode, the channel width (CW) and array size, which are input parameters, are fixed. Using the netlist file, placement result file, HDL codes templates, and architecture parameters, EasyRouter can generate all the FPGA HDL codes and an application bitstream.

First, we introduce HDL code generation. The logic part contains three levels of codes: the logic cell, basic logic element (BLE), and logic cluster (with a local connection block). For most FPGA architectures, these structures are homogeneous for all reconfigurable tiles. Therefore, the logic components of HDL codes can easily be prepared manually. The routing components of HDL codes are generated automatically with simple templates. The template consists of the structure of the switch box (SB), connection block (CB), and I/O block (IOB). The final routing HDL codes are generated according to the channel width and other routing parameters such as \( F_{c, in} \), \( F_{c, out} \) and \( F_s \) [11]. Routing resources and their connections can be generated automatically according to the information maintained in the RRGraph of the router.

Next, we discuss bitstream generation. The logic element bitstream consists of the logic cell lookup table (LUT) and the configuration memory bit of the output multiplexer. The output multiplexer selects the output of the BLE directly from the LUT or through a register [11]. The logic element bitstream is generated according to the netlist after technology mapping. The routing bitstream contains configuration memory values of the SB, CB, local connection block (LCB), and IOB, which are generated according to the actual routing results.

3.4 Report generation block

The report generation block exports routed circuit information on the target device as the final execution stage of EasyRouter. The device array size, minimum channel width, the quantity of all routing resources, and the number of used routing resources are included in this exported report. These data are derived directly from a routed RRGraph, and are useful for device performance analysis.

In order to evaluating large devices efficiently or special VLSI technology (such as 3D-VLSI) that cannot be implemented easily, a fast performance analysis method of EasyRouter can be used. Because common FPGAs are composed of tiles of the same structure, area and delay performance can be calculated from the physical information of one FPGA tile. We first finish the layout of a tile structure with VLSI design flow and obtain its area. Then the device area can be obtained from the product of the tile area and \( \text{ArraySize} \times \text{ArraySize} \). We then perform timing analysis using a simplified tile delay model, which extracts some representative paths such as SB to SB, Channel to LB, and BLE input to output, and set their delay to values according to tile STA results. The critical path and its delay are obtained from the timing analysis using the routed RRGraph and these represent delays of the paths. The area and delay performance analysis at this stage is less accurate. However, it is fast and has sufficient precision for architecture exploration. We will prove this in Section 5.3.

4. Proposed FPGA IP Design Flow

Conventional FPGA architecture exploration and implementation processes involve two separate flows. The FPGA architecture is determined by academic FPGA design flow. However, in the implementation phase, commercial VLSI design flow are used which gives rise to two problems. One is that the academic design flow cannot provide high accuracy area, delay and power estimates. The other is that if design defects are found in the VLSI design phase, then it is necessary to restart from the FPGA design flow and a large number of HDL codes needs to be revised.

We propose an FPGA IP design flow that combines the FPGA and VLSI design flows, to solve the above problems. The proposed FPGA IP design flow consists of three parts: the conventional FPGA design flow, VLSI back-end design and analysis flow, and the novel tool EasyRouter which can bridge the two flows. By employing the proposed IP design flow, architecture exploration and implementation can be performed with high accuracy and within a reasonable execution time.

4.1 FPGA scale exploration

Since the FPGA IP core has limited on-chip area, FPGA scale exploration is necessary. The objective of FPGA scale exploration is to find a rational FPGA tile array size and routing channel width by implementing target application circuits.

Figure 2 shows how we link EasyRouter with VTR to perform FPGA scale exploration. The synthesis tool ODIN II reads and optimizes an HDL-described application circuit. The output of ODIN II is a Blif netlist as it is the standard format used to pass circuit information between academic FPGA tools. Blif format circuits (ex. MCNC benchmarks)
can be directly inputted into ABC. The technology mapping tool ABC maps the netlist logic circuits into FPGA logic elements, which are typically \( k \)-input LUTs. In the case of VPR 6.0, the logic elements are first packed into clusters. The clustered logic blocks are then placed in an \( n \times n \) tile array. Finally, we use EasyRouter to make the connections for the I/O pins of all logic blocks and I/O ports of the FPGA IP. Placement and routing are performed ten times for each circuit since different seeds (from 0 to 9) of the simulated annealing based placement algorithm generate different placement solutions. The routing result for each circuit is the average of the results of ten placement seeds.

### 4.2 FPGA IP implementation and performance analysis with commercial VLSI CADs

After the architecture is determined, we run EasyRouter in the evaluation mode to generate the FPGA HDL codes and each circuit’s bitstream, which is shown in Fig. 3. When all the FPGA HDL codes and an application bitstream are generated, we can start the back-end design with commercial VLSI design CAD tools. Back-end design flows differ according to the technique used and the researcher’s design experience. However, in general, the steps shown in Fig. 3 are necessary, which are the same with common ASIC design flow.

### 4.3 Fast performance analysis with EasyRouter

The full back-end design of a large scale FPGA device is an intensely time consuming process. On the other hand, special VLSI process devices such as the 3D-FPGA cannot presently be implemented easily because of the lack of available CADs support and process technology. For these reasons, the evaluation flow presented in Fig. 3 is sometimes not efficient or not applicable. Therefore, we developed a fast performance analysis function for EasyRouter to evaluate these devices.

Fig. 4 shows the flow when using EasyRouter for fast performance analysis. When the target device architecture is determined with the method described in Section 4.1, we can make HDL code for one tile of the target device. We then implement the one tile HDL code with VLSI design flow and obtain the physical information such as area and delays of representative paths, as shown in Fig. 4 (a). Finally, as shown in Fig. 4 (b), in the fast performance analysis mode with this physical information, EasyRouter executes the area reports and timing results.

### 5. Evaluation

In this section, we first introduce the evaluation conditions. Second, we report the performance of EasyRouter, which include the execution time and minimum channel width for each benchmark. We then evaluate the proposed post-routing performance evaluation flow with a homogeneous FPGA IP. Finally, we show the expandability of EasyRouter with a 3D-FPGA case study.

#### 5.1 Evaluation conditions

During the EasyRouter performance evaluations, we used conventional island style FPGA that supported by VPR
For post-routing performance evaluation and 3D-FPGA case study, we employed a novel homogeneous FPGA architecture [12], as shown in Fig. 5. In this device, all tiles have the same structure, unlike the island-style FPGA architecture, which is composed of several types of different tiles. Therefore, the homogeneous FPGA architecture can be easily produced and tested. The details and performance of this architecture have been described in a previous paper [12]. In this evaluation we employed 4-LUT with cluster size of four. The number of inputs of LB was ten. The SB was wilton type. The $F_s$ value was 3 and the $F_c$ value was 0.5.

Circuits from the largest 20 MCNC benchmark were used for evaluation. The device was designed using e-Shuttle 65 nm CMOS technology. The functional simulation tool was ModelSim 6.5b. The design was synthesized with Synopsys Design Compiler F-2011.09-SP2. The layout was performed using Cadence EDI system 10.13. We checked the gate level netlists outputted from the Design Compiler and EDI with Formality A-2008.03-SP3. Finally, the STA was performed with PrimeTime F-2011.12-SP1.

For the comparison, the area and delay physical parameters of VPR were derived in the same flow and technology process. A tile of the target FPGA was synthesized and laid out with the same back-end design flow. The tile area was derived from the GDS after layout. Delays within the LB were extracted with the STA. The wire RC model was analyzed with the HSpice. All physical parameters were written into the architecture file in the VPR format. Note that our evaluation targets of this evaluation were synthesizable FPGAs. The evaluation result of VPR may be different for full-custom designed FPGA.

### 5.2 EasyRouter performance evaluation

As we talked, the most time-consuming function of a router is the heap sort. We tested the same heap sort algorithm in C and C#. The basic test operation involves adding numbers from 0 to 999,999 to a min-heap and then deleting it to empty from the top. The basic test operation was repeated for 30 times. Then we compared the execution time for the two implementations. The results showed that the C# implementation was around 5.0 times slower than the C implementation, because of the performance difference of C# and C language. This implies that when implementing a given routing algorithm, the C# program will be at least 5.0 times slower than the C program.

We evaluated the execution time of 17 benchmarks. According to the results, EasyRouter was 8.4 times slower than VPR on average. However, for large circuits like frisk, pdc, and clma, EasyRouter was near to 5.0 times slower. This is because for large circuits, the heap sort operations dominate the execution time to a greater extent. We examined the s298, alu4, and pdc circuits, and the cpu instruction sampling results showed that the execution time ratio of the heap function were 65.8%, 76.1%, and 83.2%. Therefore, for large circuits, the execution time overhead of EasyRouter was close to the performance difference between the C and C# implementations.

Fig. 6 shows the minimum channel widths of EasyRouter and VPR. We can see that the routing performance of both tools were similar. A reason the channel width of both differ in some circuits, is that during the RRGraph searching step, the expansion order of the RRNode with the same cost value will influence the routing results. However, because of this, the influence of the minimum channel width was only about a factor of two (the minimum change step for unidirectional routing architecture). Therefore, EasyRouter has a capability that is almost identical to that of VPR.

### 5.3 Post-routing performance evaluation

Because the FPGA IP designs have limited die size, we used a device array size of $15 \times 15$ to introduce the generation of HDL codes and bitstreams, and post-routing evaluation methods. The CW was fixed to 50. We selected the six circuits from the 20 largest MCNC benchmarks to evaluate the target device, because they can be implemented with a
target device of array size of $15 \times 15$.

The area calculation model of VPR multiplies the area of one tile by the number of tiles in the array. With an accurate tile area after layout, this module is reliable. Therefore, we only provided the physical area information of the designed target device, which is presented in Fig. 7.

Fig. 8 shows the critical path delay calculated by the flow of EasyRouter with full FPGA VLSI back-end design and STA (Full FPGA STA), EasyRouter fast performance analysis (EasyRouter), and VPR. We believe the critical path delay of the full FPGA STA was an accurate delay value because the evaluation of commercial VLSI design flow with a standard cell library has the highest simulation accuracy in industry. Note that we used the breadth-first router of EasyRouter and VPR for pure delay accuracy comparison.

The delay value accuracy calculated by VPR was 8.9 times lower than that obtained from the full FPGA STA on average. This was because the delay model of VPR was pessimistic and had low accuracy. For example, all routing segment delays were calculated with the same wire RC model. In an actual final layout, the placement was optimized and the physical delays were different. However, we can see that VPR correctly reflected the performance relationship between the circuits. This shows the reliability of VPR as a fast architecture exploration tool.

The result accuracy calculated by EasyRouter fast performance analysis was 1.7 times lower than that obtained from the full FPGA STA on average. This resulted showed that EasyRouter improved delay accuracy 5.1 times than VPR on average. This was because, although EasyRouter used a similar pessimistic model as VPR, all representative path delays were calculated with the high accuracy STA process. On the other hand, the routing delay and logic delay of VPR was calculated with different models. Therefore, we conclude that the EasyRouter fast performance analysis method is reliable for fast high accuracy device evaluation.

5.4 3D-FPGA case study

EasyRouter is designed to implement new FPGA architectures easily. In this section, we show the expandability of EasyRouter by evaluating a novel 3D-FPGA architecture that was developed in a previous work [13]. The area and critical path delay performance of the homogeneous 2D-FPGA and the novel 3D-FPGA were compared. The new 3D-FPGA architecture script file was modified from a conventional 2D-FPGA architecture script file by adding only few codes for vertical connections of 3D-VLSI technology.

5.4.1 Target 3D-FPGA architecture

Fig. 9(a) and (b) shows the tile image and the detail of the proposed 3D routing architectures. The two layers in the proposed 3D-FPGA were the logic and routing layers. We employed the face-down 3D stacking technique to connect two dies with micro bumps. The tiles on the logic layer had a LB and a small part of the routing resources, while the tiles on the routing layer had only routing resources. The tiles for the two layers were designed within approximately the same area. Different from conventional 3D routing architectures with 3D-SBs, we made the 3D connections on the input and output pins of the LB, which we named 3D-CB structure. The router chose one net to be routed on either the logic layer or the routing layer.

By dividing routing resources into two layers, we achieved a smaller tile. A smaller tile means a higher logic density, shorter routing wire, and faster signal transportation. Therefore, the routing performance could be improved. Moreover, the proposed 3D-FPGA was realistic, because the number of inter-layer connections within one tile was equal to the number of input and output pins of the LB. Compared to conventional the 3D-FPGA based on the 3D-SB, which required two times the number of channel width inter-layer connections, the proposed architecture significantly reduced the requirement for inter-layer connections.

5.4.2 Evaluation conditions and results

We successfully implemented the 3D-FPGA architecture on EasyRouter in a relatively short development time. The FPGA scale exploration was performed with the flow that we introduced in Section 4.1. The performance analysis was performed using the method that we described in Section
4.3. We simply define the delay of one vertical connection between logic layer and routing layer as the same delay of one segment wire.

Fig. 10 shows the evaluation results for the area. We can see that the proposed 3D-FPGA used half the package area of 2D-FPGA by allocating nets on two layers. This means the logic density had improved by about a factor of two. The critical path delay also improved about 4% on average. This is because the increased channel width has better routability, and the smaller tile has shorter routing wire length.

With this 3D-FPGA case study, we can say various architectures can be implemented on the EasyRouter framework within a relatively short development time. High accuracy area and delay performance analysis can also be performed with the proposed framework.

6. Conclusions

In this paper, we proposed a novel FPGA routing tool, EasyRouter, and an FPGA IP design flow that combines conventional FPGA design tools with VLSI CADs. EasyRouter facilitates easy modeling of new FPGA architectures without any limitations, which can significantly shorten the development cycle. EasyRouter can also automatically generate device HDL codes and configuration bitstream files of the implemented circuits that can be processed by VLSI CADs. With this design flow, accurate physical information STA can be reported when a new FPGA IP architecture is evaluated with reliable commercial VLSI CADs. For FPGA architectures that cannot be easily implemented with present VLSI process, EasyRouter provides a fast performance analysis flow, which improved delay accuracy 5.1 times than VPR on average. We have also evaluated the proposed FPGA design flow with three different devices to show its performance and expandability.

References


