UNIVERSITY OF CALIFORNIA,
IRVINE

Heterogeneous FPGA Place and Route

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Information and Computer Science

by

Simin Dai

Thesis Committee:
Professor Eli Bozorgzadeh, Chair
Professor Tony Givargis
Professor Ian Harris

2006
The thesis of Simin Dai is approved:

___________________________
___________________________
___________________________

Committee Chair

University of California, Irvine
2006
DEDICATION

To

my parents, my family and friends
# TABLE OF CONTENTS

<p>| LIST OF FIGURES | v |
| LIST OF TABLES | viii |
| ACKNOWLEDGMENTS | ix |
| ABSTRACT OF THE THESIS | x |
| CHAPTER 1: Introduction | 1 |
| Evolution of Programming Logic Devices | 1 |
| Modern FPGA Design Flow | 4 |
| Heterogeneous FPGA Placement and Routing Challenges | 5 |
| New Design Flow for Heterogeneous FPGA Devices | 6 |
| CHAPTER 2: FPGA Place and Route Related Work | 9 |
| VPR Tool | 9 |
| Directional Bias and non-uniformity in FPGA Routing | 11 |
| Embedded Memory Block within FPGA | 13 |
| Heterogeneous FPGA Placement and Route Challenges | 14 |
| CHAPTER 3: H-VPR Enhancements | 15 |
| Netlist Generation for Designs using embedded Cores in FPGAs | 15 |
| Place and Route tools for FPGAs with embedded Cores | 26 |
| CHAPTER 4: H-VPR Benchmarks | 46 |
| Altera Benchmark Circuits | 46 |
| Embed Blackbox Circuit using Synopsys Flow | 48 |
| H-VPR placement and routing | 49 |
| CHAPTER 5: Design Space Exploration | 51 |
| Uniform channel routing | 51 |
| Non-uniform channel routing | 51 |
| Experimental Results | 54 |
| CHAPTER 6: Summery and Conclusion | 66 |
| REFERENCES | 67 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Generic FPGA Floorplan</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Generic CAD Flow from Function Description to FPGA Programming</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Hetero-FPGA Design Flow</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>VPR CAD Flow</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Based FPGA Logic Block Consist of LUT and FF</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Directional Bias Routing in FPGA</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Non-Uniformity Routing in FPGA</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>FPGA with Embedded Memory Blocks</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Circuit Block Diagram</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Hierarchy chart</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Input and Output Node Changes as Result of Black Box</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Circuit Placement and Route</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Black Box IO Node Placements</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Hierarchy chart</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Timing graph</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Block Connections</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Connections between Wires</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Routing with Black Boxes Fully Connected</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Detailed View of Area near Black Box</td>
<td>34</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>Single Segment Routing</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.13</td>
<td>Wire Split at Black Box</td>
<td>36</td>
</tr>
<tr>
<td>Figure 5.15</td>
<td>Final Circuit Area for Oc_cordic_r2p Benchmark Circuit</td>
<td>62</td>
</tr>
<tr>
<td>Figure 5.16</td>
<td>Final Circuit Area for Fip_cordia_cla Benchmark Circuit</td>
<td>62</td>
</tr>
<tr>
<td>Figure 5.17</td>
<td>Final Circuit Area for Oc_des_perf_opt Benchmark Circuit</td>
<td>63</td>
</tr>
<tr>
<td>Figure 5.18</td>
<td>Final Circuit Area for Oc_fpu Benchmark Circuit</td>
<td>63</td>
</tr>
<tr>
<td>Figure 5.19</td>
<td>Circuit Area Summary for all Benchmark Circuits</td>
<td>64</td>
</tr>
<tr>
<td>Figure 5.20</td>
<td>Key parameter Summary for all Benchmark Circuits</td>
<td>65</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>VPR Benchmark Comparison</td>
<td>11</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Summary of H-VPR Results on Benchmark Circuit</td>
<td>47</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Summary of Black Box Circuit Attributes</td>
<td>48</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Synopsys Benchmark Circuit Attributes</td>
<td>49</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Channel Width Comparison</td>
<td>49</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to thank Professor Bozorgzadeh for the guidance and advice throughout the entire process. I really appreciate all the informative discussion.

I would also like to thank Professor Givargis and Professor Harris for taking valuable time in their busy schedule to serve on my thesis committee.
ABSTRACT OF THE THESIS

Heterogeneous FPGA Place and Route

By

Simin Dai

Master of Science in Information and Computer Science

University of California, Irvine, 2006

Professor Eli Bozorgzadeh, Chair

Programmable logic devices offer significant advantages in design flexibility, in field program ability and time to market. Traditional FPGA devices were consisted of IO blocks on the periphery and logic blocks in the middle connected by wires. The entire device appears homogeneous. For performance and functionality enhancement, latest generation of FPGA devices incorporate embedded hard macros. With the embedded IP blocks, the device is classified as a heterogeneous FPGA.

FPGA placement and route is a very important process as it directly affects the size, speed and power dissipation of the realized circuit. The versatile placement and route (VPR) [12] tool was developed to study the architecture tradeoffs for FPGA place and route. But VPR was originally conceived for homogeneous architectures and does not support heterogeneous FPGA with embedded hard cores. There is a need for CAD tool that supports multiple architectures and allows parameterization of attributes that affect the efficiency of placement and routing operation. Such tool would enable the researcher to conduct design space exploration and evaluate the effectiveness of algorithms for placement and route within a heterogeneous FPGA.
The purpose of this project is to develop a CAD flow called H-FPGA from RTL down to bit stream level such that the user can define any modules to be embedded as hard macros within the FPGA and the software outputs a placed and routed implementation using selected architecture. We have developed H-VPR tool on top of current state-of-the-art VPR. The application of H-FPGA flow is demonstrated successfully on multiple benchmark circuits.
Chapter 1 – Introduction

1.1 Evolution of Programming Logic Devices

Recently, the uses of programmable logic devices have seen tremendous growth, particular in the areas of networking and signal processing. FPGA is a class of integrated circuits which could perform different logic functions by changing the internal connection between logic elements. The modern programmable logic devices such as FPGA now utilizes the most advanced IC manufacturing process and have enough processing power that could compete with microprocessors. Compared to traditional Application Specific Integrated Circuits (ASIC), FPGA offers several advantages and disadvantages:

Advantages:

- Reduced design cycle time. The silicon device is already in hand. No need to wait for mask generation and fabrication of devices.
- Re-programmable, device functionality can be upgraded in the field.
- Flexibility, additional functions can be incorporated into the device by re-programming the device.
- Potential lower cost for low to medium volume applications where ASIC foundry charges could be very high.

Disadvantages:

- FPGA uses larger circuit area to implement a particular function, as a result could have higher power dissipation compared to ASIC device.
• FPGA has slower speed compared to ASIC due to tradeoff between optimized routing and re-programmability.

• For large volumes, FPGA devices have higher unit cost compared to ASIC device. Currently in market, the programmable logic device can be classified as PLA, CPLD, and FPGA devices [1] [2] [4].

1.1.1 PLA and PAL Devices

PAL is the first programmable device developed by MMI [3]. PAL devices has a programmable AND array and a fixed set of OR array. PLA programmable logic array has programmable AND array and programmable OR array. These two devices usually are used to implement combinational circuits such as addressing decoders. TIBPAL22V10-7C from Texas Instrument is an example of PAL [5].

1.1.2 CPLD Devices

When we put several PLA/PALs on one silicon chip, it becomes a complex programmable logic device (CPLD). CPLD is composed of several logic blocks. Within each logic block there are multiple set of macro cells. Logic blocks are connected by a programmable interconnect matrix. Compared to other programmable logic devices, CPLD have good performance but less flexibility. Usually we use CPLD devices to implement control logic or complex state machines which does not require as much flip flops. An example CPLD device is Cypress ultra37000.

1.1.3 FPGA Devices

FPGA stands for Field Programmable Gate Array. FPGA devices also contain multiple logic blocks. Unlike CPLD devices, the logic block within FPGA contain both LUT and flip flops. Figure 1.1 illustrates a generic two-dimensional FPGA layout (e.g.
older generation Xilinx devices). A typical FPGA is consists of IO block, logic block and routing resources in both X and Y directions.

![Figure 1.1 Generic FPGA Floorplan](image)

On the perimeter of the FPGA are the IO blocks that contain the input and output buffers to interface to the outside world. There are logic blocks in the middle of the device. Those logic blocks are typically composed of look up tables (LUTs) that perform simple logics functions. Between the logic blocks are routing resources that can be used to connect logic blocks together. More complex logical functions can be accomplished by forming connections between logical elements. The routing resources consist of wires running in X and Y directions between the logic blocks on different layers of silicon. Switches are used to change routing direction from X to Y and vice versa. Currently the companies that provide FPGA devices are Actel [6], Altera [8], Xilinx [7] and several others.
1.2 Modern FPGA Design Flow

In modern FPGA design, circuit functionality is typically defined using hardware description languages (HDL) such as VHDL and Verilog. The circuit functionality defined by HDL must then be translated into silicon physical properties such as number of IO, gate and wires required in order to implement the function. The general CAD flows is shown in Figure 1.2:

![Diagram of CAD Flow](image)

Figure 1.2 Generic CAD Flow from Function Description to FPGA Programming

Once the circuit functionality is defined in HDL, an electronic data interchange Format (EDIF) [9] file is used to represent the Nelist, which defines the functions of logic
circuit blocks and input and output connection between the blocks. Circuit synthesis tools are then used to translate the hardware description into number of more basic logical elements and connections between them required within FPGA in order to implement the logic function. The logic synthesis may be iterated several times to ensure timing closure and optimization in terms of power and area. Currently there are several popular commercially available logic synthesis tools such as Synopsys Design Compiler and Synplify from Synplicity in use in the industry [10] [11]. Synopsys Design Compiler is used throughout the work of this project. After logic synthesis is complete, another set of tool are used to perform place and route. Place and Route (P&R) operation designates the location of the logic blocks and designate which wires are used to connect the logic blocks. Finally a FPGA programming file is generate to load into FPGA device to program the circuit functionality. This thesis focuses on the placement and route aspect of design flow. In addition to many commercially available FPGA placement and routing tools, currently there are one tool used in academic called VPR [12] (Versatile Packing, Placement and Routing for FPGAs) that can be used to perform placement and routing for FPGA devices.

1.3 Heterogeneous FPGA Placement and Routing Challenges

Traditional FPGA architecture is considered homogeneous if it is composed of regular array of logic elements (LE) and macro cells. Recently for system on a chip (SOC) developments, FPGA manufactures like Altera [8] and Xilinx [7] are fabricating heterogeneous FPGA with embedded memory blocks, phase lock loop (PLL), processor core and other intellectual property (IP blocks) in additional to programmable logic elements. For example, latest Virtex-4 FPGA from Xilinx [7] can have IBM PowerPC™
processor core, Ethernet MAC and high speed RocketIO transceivers embedded within the FPGA. With an embedded custom IP circuit block, the user doesn’t have to define and optimize the critical circuit functions. With embedded black box as part of FPGA fabric, higher performance, greater functionality and reduction of silicon area can be achieved without sacrificing the flexibility of FPGA architecture.

For FPGA routing with the black boxes in place, there is typically more congestion around black box compared to other areas inside the FGPA. One of the reasons is the black box contains predefined circuitry so very limited or non-existent routing wires can go through the black box. They must be routed around the black box. There has been intense research interest on optimal use of routing resources within the FPGA. A number of research proposals such as “If we increase the number of tracks around the black box, will the final routing resources be more optimal compared to having a uniform number of tracks throughout the FPGA?” are being proposed. Current VPR tool is targeted for homogeneous FPGA architectures and does not support black boxes. So we are not able to use VPR to evaluate different types of embedded black boxes in terms of routing congestions. So to support researchers on new algorithms and proposals for modern FPGA architectures that support embedded black boxes, new capabilities must be incorporated into VPR.

1.4 New Design Flow for Heterogeneous FPGA Devices

Currently VPR assumes a global route structure and does not have the capability of perform routing around embedded IP core. We propose a modified CAD flow called Hetero-FPGA (H-FPGA) to support the embedded black blocks. The Hetero-FPGA flow is outlined in Figure 1.3. The Synopsys Design Compiler is used to translate the HDL
netlist into a .net file, which is a netlist that defines how the circuit elements should be connected between logic blocks. The .net file is then used as input into VPR program. The VPR flow differs from commercial flow starting with translation of EDIF file into BLIF file. An .arch (architecture) file [27] is then used to describe the basic logic elements within the target FPGA as well as wiring resources. The .arch file may contain information such as number of logic element, type of logic elements such LUT and number of routing tracks between logic elements. The .arch file will be discussed in more detail in the subsequent sections.

Figure 1.3 Hetero-FPGA Design Flow
The LUT plus a D-FF can be considered as a basic logic element (BLE). Another tool called T-Vpack is then used to pack BLEs into logic clusters. A logic cluster can contain a number of BLEs as well as clock inputs. Finally, the modified VPR tool can be used to perform placement and route operation. The placement operation can be tuned to achieve minimum wiring density (routability-driven) or maximum circuit speed (time-driven).

Within the Hetero-FPGA flow is H-VPR tool, which is developed to incorporate additional capabilities to existing VPR tool to allow routing of heterogeneous FPGA devices [13][14]. The .arch file has to be modified to include additional descriptors such as size, location, number of input and outputs of black box. Additional code modification allows the H-VPR to route around the black box. With these features, H-VPR would enable researchers to explore the design space of routing channel configuration around hard black box in FPGA devices to achieve better routing resource usage.
Chapter 2 – FPGA Place and Route Related Work

2.1 VPR Tool

The Versatile Place and Route (VPR) [12] tool was developed in the late nineties by researchers from University of Toronto. The focus of VPR tool is to evaluate the effectiveness of placement and route using new FPGA architecture. Several benchmark circuits are technology mapped and routed. The effectiveness of the new architecture can be measured in terms of speed and area of the routed FPGA circuit. Figure 2.1 illustrates the input and output information for the VPR tool flow:

![VPR CAD Flow Diagram]

Figure 2.1 VPR CAD Flow

The .arch [27] file describes the FPGA architecture. It is a text based file that could contain the following information about the FPGA:

- The number of logic block inputs and outputs,
- The side(s) of the logic block from which each input and output is accessible,
- The logical equivalence between various input and output pins
- The number of I/O pads that fit into one row or one column of the FPGA, and the dimensions of the logic block array (e.g. 23 x 30 logic blocks).

VPR tool supports both global routing and detailed routing. For global routing, the following information is specified in .arch file.

- The relative widths of horizontal and vertical channels and relative widths of the channels in different regions of the FPGA.

Finally, if combined global and detailed routing is to be performed, the following are specified:

- The switch block architecture (i.e. how the routing tracks are interconnected)
- The number of tracks to which each logic block input pin connects
- The $F_c$ [17] value for logic block outputs and $F_c$ value for I/O pads.

T-VPack [16] is used to pack the technology mapped netlist composed of LUT and FF into a FPGA logic block. The FPGA logic block shown in Fig. 2.2 [17] is composed of one LUT and one FF. Logic block composed of multiple LUTs and FFs is also supported by T-Vpack.

![Figure 2.2 Based FPGA Logic Block Consist of LUT and FF](image)

The VPR tool was able to outperform other placement and routing tools available at the time. In both big and small benchmark circuits, VPR utilizes less routing resources.
An example of performance comparison using large benchmark circuits is show in Table 2.1.

Table 2.1 VPR Benchmark Comparison [12]

<table>
<thead>
<tr>
<th>Benchmark Circuit</th>
<th>Number of logic blocks</th>
<th>SEGA channel width required</th>
<th>VPR channel width required</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>1522</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>apex2</td>
<td>1878</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>apex4</td>
<td>1262</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>bigkey</td>
<td>1707</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>clma</td>
<td>8383</td>
<td>&gt;24</td>
<td>12</td>
</tr>
<tr>
<td>des</td>
<td>1591</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>diffeq</td>
<td>1497</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>dsip</td>
<td>1370</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>elliptic</td>
<td>3604</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>ex1010</td>
<td>4598</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td>ex5p</td>
<td>1064</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>frisc</td>
<td>3556</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>misex3</td>
<td>1397</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>pdc</td>
<td>4575</td>
<td>&gt;31</td>
<td>16</td>
</tr>
<tr>
<td>s298</td>
<td>1931</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>s38417</td>
<td>6406</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>s38584.1</td>
<td>6447</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>seq</td>
<td>1750</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>spla</td>
<td>3690</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>tseng</td>
<td>1047</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>&gt;331</td>
<td>197</td>
</tr>
</tbody>
</table>

With its flexibility and performance, the VPR tool can be used to study interesting FPGA architecture proposals that will be discussed in the next section.

2.2 Directional Bias and non-uniformity in FPGA Routing

The problem of direction bias and non-uniformity in FPGA routing was investigated by Betz and Rose [18]. The concepts of direction bias and non-uniformity are illustrated in Figure 2.2 and 2.3, respectively. In the case of directional bias, a FPGA may have
more routing channels in the X direction compared to Y direction. Would this architecture be more efficient and how much bias would be needed? In the case of non-uniformity architecture, the promise is to include extra routing resources for the likely area of circuit congestion such as center of FPGA for example.

Figure 2.3 Directional Bias Routing in FPGA

Figure 2.4 Non-Uniformity Routing in FPGA
Research results showed that in most benchmark circuits, the most efficient architecture was one with completed uniformity across the entire FPGA. The efficiency improved marginally with non-uniform architectures if the connection pins are strategically located.

### 2.3 Embedded Memory Block within FPGA

Wilton et al. [19] investigated FPGA architecture with embedded memory blocks, as shown in Figure 2.5. The challenge is to provide a good memory/logic interface since the memory access time can often be the performance bottleneck of the system.

![Figure 2.5 FPGA with Embedded Memory Blocks](image)

The VPR tool was enhanced to incorporate memory-to-memory switches. Each wire incident to the memory blocks can also be programmed to connect to two neighboring memory blocks via pass transistors. These connections provided very low delay path between memory blocks and also allowed sharing of address and data signal lines among
all the memory blocks. The experimental evaluation showed improvement as much as 25% in memory read time and improved area usage [19].

2.4 Heterogeneous FPGA Placement and Route Challenges

The embedded memory architecture discussed previously is one instance of using VPR to accommodate IP cores. The enhancement to VPR is specifically designed to improve routing to memory blocks by adding memory to memory switches. Also the previous investigation makes the assumption that the memory blocks must be arranged in a single row across the width of the FPGA. The focus of this thesis is on physical design exploration of FPGAs with embedded blocks. We assume the black boxes are generic in nature and their specifications are given by designers. This enables us to investigate the impact of different type of embedded blocks on physical placement and routing congestion within the FPGA fabric. Hence we start the design flow from VHDL with highlighted embedded cores, exclude those during FPGA logic synthesis and integrate them together in place and route (locating the embedded blocks as defined by users). This flow required significant enhancement to VPR capability to accommodate multiple types of black boxes with arbitrary placement within the FPGA. The next several chapters will discuss the new capabilities and modifications in detail.
Chapter 3 - H-VPR Enhancements

This chapter focuses on development of CAD tool for FPGAs with embedded hard cores such as embedded multipliers, memory blocks, etc. The CAD flow is developed from VHDL description of the design to generating bit stream for device. The tool is partly developed on top of state-of-the-art FPGA CAD tool, VPR [12]. We refer to that as H-VPR. In this chapter I will explain how the net file for H-VPR is created. I will also explain how the H-VPR performs placement and routing for netlists which have embedded hard cores.

3.1 Netlist Generation for Designs using Embedded Cores in FPGAs

The first part of the flow generates a netlist from structural VHDL description of a design. There are certain macros in design that are explicitly specified to be mapped to embedded hard macros on FPGAs. Hence, in this stage, the entire design is synthesized by synthesis tools except the parts mapped to hard cores on device. Then, the netlist is passed to place-and-route tools for physical implementation which is described in details in Section 3.2. The netlist format is in net format (VPR netlist format after technology mapping and packing to CLBs). Here I introduce two ways to generate .net file from VHDL. One method is using commercial Synthesis tools from Synopsys, converting EDIF format to BLIF format “edif2blif”, and FPGA logic synthesis tool [21] called RASP_SYN FPGA MAP. The other method is Altera Quip[22] method. In this project we use both methods to generate net file.
3.1.1 BLIF File Generation Using Synopsys

The example circuit below is taken from Synopsys Design Compiler User Guide Appendix A. With this example, we describe the process of generating the netlist. The example circuit block diagram is show in Figure 3.1 [23]:

![Circuit Block Diagram](image)

Figure 3.1 Circuit Block Diagram

The circuit hierarchy is as follows:

![Hierarchy Chart](image)

Figure 3.2 Hierarchy chart
3.1.1.1 Specify FPGA library

Link library and target library are technology libraries provided by the FPGA vendor. The libraries provide cells specifications for the logic elements within a target FPGA device. Since the final netlist file will be input file for edif2blif and it uses Altera max7000 library, so the link library, target library will be max7000.db. All these are specified in Synopsys Design Compiler set up file.

3.1.1.2 Read design

Use either read_file command or analyze/elaborate command to read VHDL/Verilog file and generate technology-independent design. If the circuits are hierarchical, start from the lowest sub-design and then goes to top circuit block. In our project, we want our final net file to act as a black box with no visibility to the circuits inside. So we didn’t read multiplier 8 x 8 and multiplier 16 x 16. Instead, we read these two source file individually in order to get their circuit area. Because in max7000.db [24], area unit is not in terms of gates, so when we want to get area of design, we shall change library to Synopsys class.db.

Since my projects involved physical instead of functional implementation of FPGA device, I didn’t specify any design environment such as drive length on input ports. Neither did I specify any design constraints such as minimum area or maximum speed.

3.1.1.3 Compile design

Use compile command to compile the current top design.

Use report_area command to get total number of gates of the design. We don’t count wires area in our project. Synopsys class.db use NOR2 as unit area. Here is example output for multiplier 8x8.
Library(s) Used:

    class (File: /opt/synopsys-U-2003.06/libraries/syn/class.db)

Number of ports:               33
Number of nets:                356
Number of cells:               339
Number of references:          19
Combinational area:            639.000000
Noncombinational area:         112.000000
Net Interconnect area:         undefined (Wire load has zero net area)
Total cell area:               751.000000
Total area:                    undefined

We could see the total gates for multiplier 8x8 is 751. Use the same method I determined the area for multiplier 16x16 is 3425 gates. We know NOR2 gates consists 4 transistors, so multiplier 8x8 has 3004 transistors and multiplier 16x16 has 13700 transistors.

3.1.1.4 Save the design database

The final design will be written in EDIF format. This design will consist of multiple black boxes, since we don’t read/compile multiplier 8x8 and multiplier 16x16. The EDIF file is composed of cell definition, instances, and net description. Their examples are listed below:
a. Cell definition

(cell INV (cellType GENERIC)

(view Netlist_representation (viewType NETLIST)

(interface (port A_OUT (direction OUTPUT)) (port A_IN (direction INPUT)))

)

)

This cell describes an INV cell which has one input \textit{A\_IN} and one output \textit{A\_OUT}.

b. Instances

(instance (rename u7\_U100\_FS\_1\_U30 "u7/U100/FS\_1/U30")

(viewRef Netlist_representation (cellRef INV (libraryRef max7000)))

)

\textit{U7\_U100\_FS\_1\_U30} is an Inverter.

c. Net description.

(net (rename u7\_U100\_FS\_1\_n54 "u7/U100/FS\_1/n54")

(joined (portRef IN2 (instanceRef u7\_U100\_FS\_1\_U7))

(portRef IN1 (instanceRef u7\_U100\_FS\_1\_U25))

(portRef A\_IN (instanceRef u7\_U100\_FS\_1\_U30))

(portRef IN1 (instanceRef u7\_U100\_FS\_1\_U31))

(portRef A\_OUT (instanceRef u7\_U100\_FS\_1\_U21))

)

)

Net name is \textit{u7\_U100\_FS\_1\_n54}. It is driven by \textit{A\_OUT} of \textit{u7\_U100\_FS\_1\_U21} instance. It has three receiver “\textit{IN1 instanceRef u7\_U100\_FS\_1\_U25}”, “\textit{A\_IN instanceRef u7\_U100\_FS\_1\_U30}” and “\textit{IN1 instanceRef u7\_U100\_FS\_1\_U31}”.
3.1.1.5 Edif2blif

Edif2blif [20] is a software tool that translates EDIF into BLIF [15] format. The BLIF stands for Berkeley Logic Interchange format. FPGA map application requires input file in BLIF format. So we need edif2blif to translate EDIF file into BLIF file. In order to make edif2blif work, we need two input files. One is EDIF file describing the circuit connection, the other is translation table. The translation table defines all cells in the EDIF file. Edif2blif application provide translation table which supports Altera Max7000 device, but it doesn’t define all the cells available in Max7000 such as OR11. We would need to add definition into translation table. Our EDIF file is consist of two black boxes which are not defined in translation table. So in order to make edif2blif ignore these two black boxes and produce BLIF file, we removed black box instances from EDIF file and make the input pin of black boxes to be whole module output pins, make the output pins of black boxes to be input of whole module. This is done manually. We haven’t developed any automated program yet. But in future, we could have a program which could remove black box automatically.

There are some limitations in edif2blif. For example, for a latch cell, it only support D type flip flop, it doesn’t support preset, enable, clear inputs. All those inputs will be ignored by edif2blif.

The final BLIF file is composed of modules, inputs, outputs, latches and logic gates.
Following is an example of output BLIF file.

a. Model description, input and output pins

```
.model ChipLevel
.inputs data16_a_15_ ....
.outputs mux_out_15_ ....
```

Model name is *ChipLevel*, it has inputs *data16_a_15*, outputs *mux_out_15*.

b. Flip flops and latches

```
.latch u1_add_20_2_U4 u1_cout_reg re clk 2
```

Latch input data is *u1_add_20_2_U4*, output is *u1_count_reg*, type is rising edge, initial value don’t care.

c. Logic gates

```
.names u7_U78 u7_U79 u7_U16
01 1
10 1
11 1
```

Logic gates demonstrate logic functions. Above means

\[
U7_U16 = u7_U78'u7_U79 + u7_U78u7_U79' + u7_U78u7U79.
\]

3.1.1.6 Logic synthesis and optimization

After we generate the BLIF file we need a tool to perform LUT mapping. RASP_SYN [21] is a general logic synthesis tool for SRAM based FPGA. This synthesis tool was developed by researchers at UCLA. It is integrated with SIS [25] developed by UC Berkeley. Its process flow is as follows:

The tool works as follows: First the gates are decomposed into \( k \) input LUTs (i.e., decompose a gate network into \( k \) bounded network), from [34][35] we know LUT input
pins 4 to 6 will be most area efficient. In our project, we will use 4-input lookup tables ($k=4$). The problem of generic LUT mapping is to map a $k$-bounded network into $k$-input LUTs. There are several algorithms we could choose, such as FlowMap[26], FlowMap-R, Cutmap, FlowSyn and others. Each has different optimization objectives. I used FlowMap in the project. FlowMap will generate a depth-optimal solution for the whole Boolean network.

In this tool there is feature for final architecture specific mapping. In our project we don’t need to group CLBs for some specific FPGA. So we did not apply this feature. In the final BLIF file, the circuit is consists of only latches, 4 input LUTS, inputs and outputs.

3.1.2 Generate BLIF file using Altera Quip

Altera Quartus Integrated Synthesis [22] can convert VHDL or Verilog designs into BLIF file for input of VPR. But how could we generate a BLIF file which can contain logic cores? Unlike previous Synopsys flow, the Quartus software will not compile the whole design if some component is missing. So process starts with removing the component from design in the original VHDL code. Then we add input signal to logic core output assign output signal to be logic core input signal. For example in Figure 3.3, we will make component Adder as black box. Net A is the input of Adder and net B is the output of Adder. We need to modify the whole design in VHDL format to remove adder block and make net A be top level output net and net B to be top level input net.
3.1.3 T-vpack

Logic synthesis produces a technology mapped netlist of look up tables and flip flops in BLIF format. T-vpack [16], a timing driven logic blocking tool will pack look-up table (LUTs) and flip flops into more complex logic blocks. The basic logic blocks will be composed of at minimum of one LUTs and FFs.

A logical cluster could contain N LUTS and N FFs. In our project we packed LUTS and FF into basic logic block which contains one LUTs and one FF. The final result is saved in net file. An example of net file is shown below.

```plaintext
.globa1 clk
.input data16_a_15_
.pinlist: data16_a_15_
.output out: cout1
.pinlist: cout1
.clb u2_u10_sout_reg_2_
.pinlist: u1_ain_tmp_reg_1 u2_u10_sout_reg_2 u2_u10_add_20_2_U28 \ u2_u10_add_20_2_U41 u2_u10_sout_reg_2 clk
.subblock: u2_u10_sout_reg_2_ 0 1 2 3 4 5
```
clk is a global signal. \textit{data16\_a\_15} is IO input pin. \textit{cout1} is IO output pin.

CLB name is \textit{u2\_u10\_sout\_reg\_2\_}. There are 6 pins on CLB. 4 of them are input pins, 1 is output pin, and one is global pin.

Inputs of CLB are \textit{u1\_ain\_tmp\_reg\_1\_ u1\_bin\_tmp\_reg\_1\_ u2\_u10\_add\_20\_2\_U28}. Output of CLB is \textit{u2\_u10\_sout\_reg\_2\_}.

Subblock name is \textit{u2\_u10\_sout\_reg\_2\_}, it contains one BLE (basic logic elements), the inputs is from CLB input pins, its output is CLB output pin. The block also has a clock global pin which is 6th pin in CLB.

\subsection{3.1.4 VPR}

The versatile Place and Route (VPR) CAD tool could perform placement and routing for FPGAs. It requires .net file and .arch file. Net file is generated from T-vpack. The .arch file is a file used to describe the FPGA architecture such as number of LUTs and FFs in one logic block, switch block type, etc. After getting the input net file, it will start placement operation which will determine where to place the logic blocks. It could minimize the total required wires or maximize circuit speed as required. After placement, switches will be turned on to connect input/output pins in logic block. A router will ensure which switch is on in order to minimize the distance between connected pins. Following is snapshot taken from VPR in this project (the net file without black box). Figure 3.4 shows an example of a benchmark circuit after VPR placement and routing.
Since we removed black box in our net file and changed the input/output of black box into FPGA input/output, we would need to put the black box back into the net file and modify VPR source code so that it could place and route a net file with black box inside. In the next section we will discuss VPR modifications in more details.
3.2 Place and Route tools for FPGAs with embedded Cores

3.2.1 Net file and Architecture File

In T-vpack, a netlist file is generated without black boxes. We need to add black box back into the netlist file. The original .arch file does not support heterogeneous FPGA architecture, so we needed to specify the embedded IP core size, location, and the number of input/output pins. But how do we determine the size of the black boxes? In Synopsys synthesis flow, we can determine the number of transistors for each cell or black box. The author of VPR also develops a tool called TransCount [17] which could calculate the number of transistors in one logic cell. In our project, one logic cell has one basic logic elements, one clock, 4 input pins, LUT size is 4. After we entered those values, the TransCount print out 213. This means one logic cells has 213 transistors. So for multiplier 8x8 it will occupy 3004/213 logic cells which is 14. For multiplier 16x16, it will occupy 13700/213 logic cells which is 64.

In .arch file, we will assign size of black box. We could assign any width or height, provided the final number of logic cell matches the one we calculated above. For multiplier 8x8 it occupies 14 logic cells. We can choose that width is 7 and height is 2, for multiplier 16x16, we choose width 8 and height 8. For the location of black boxes, we could assign the blocks at any location provide that these two logic cores does not overlap. We will assign the left bottom of black box as the start position. Here for multiplier 8x8 I assigned x = 0.2 and y = 0.3, for the multiplier 16x16, its x position is 0.5 and its y position is 0.5.
3.2.2 Read .arch and .net File

In `read_arch` step, I added new data structures for black box. The data structure stores information such as number of pins in the black box, the location of each pin in black box and the size of black boxes, etc. This information is extremely useful when we build the routing graph (*rr graph*). Each black box will consist of $n$ logic cells, those logic cells will be marked as black boxes. Since in .arch file we only know which pin is on which side of black box, it does not specify the location relative to logic cell. So if we have $n$ pins on one side of black box, we will distribute all the pins evenly in each logic cell. For example, we have 5 pins in following black box top side, so the fifth pin will be put on the right top logic cell.

![Figure 3.5 Black Box IO Node Placements](image.png)
In `read_net` step, one black box will be treated as one block, so when we read net file and found black box, we will add net information of this black box one time.

### 3.2.3 Allocate and Load Timing Graph

Timing graph provides delay between each timing node. The timing graph will determine the final speed of circuit after it was completely placed and routed. It is also useful to determine the net slack during routing. The timing node could be input or output pin of a logic cell, IO, or black box. For the black box, even if it covers several logic cells, we will treat it as one single block. The edges of input pins in black box will be all output pins on this black box. For example, in following graph, the orange color indicates the location of the black box, the green color boxes represent normal logic cell. Pins with red color are input pins and pins with yellow color are output pins. Pin 5 is CLB input pin on logic cell and connected with pin 6 on the black box. Pin 10 is circuit pad output connected to pin 0 of CLB. For black box, it doesn’t have any sub-block. But for normal logic cell it will have subblocks. One subblock is consisting of one LUT and one FF. In our case, because we have one LUT and FF, so we have one sub-block in the logic cell. The sub-block input and output pins are also nodes in the timing graph. We would like to focus on logic cores, so the following timing graph does not show the sub-blocks input and output pins.
Figure 3.6 Hierarchy chart

Timing graph is as follows:

Figure 3.7 Timing graph

In above timing graph critical path is $10 \rightarrow 0 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 8$

$T_{critical} = T_0 + T_1 + T_5 + T_6 + T_7$.

3.2.4 Build Routing Resource Graph

Routing resource graph is a graph which describes FPGA connectivity. The nodes inside the graph could be logic cell pins and wires. The edges of the graph could
represent switches. The resource graph connects all nodes that could be connected. Following graph is FPGA with all pins wires connected. The yellow boxes are switches, white block are logic cells, orange block represents the black box and black lines are wires. The channel width defines how many wires are inside the routing channel. Here the channel width is 3. The wire length is how many logic cell wires could pass. Here we define the wire length is 1. We also call this is single segment, If wire length is greater than 1, then we call it is multiple segments. The fanout is how many wires a pin could be connected to. Here the fanout is 1, which means the pin could connect all wires in its channel. The fan-in attribute is how many incoming wires that the switch block can connect. So the fan-in is 1 which means it could connect to all incoming wires. We could see there are no wires inside black box and in switches. If in black box side there are no wires, then switches will not have connection as well. The logic cells are normal and all have 4 input pins, one on each side. But for the black box, it could have 3 pins on one side.
Now let take a close look at see what the resource graph will look. In final routing part, start node is not pin instead it is logic block itself, target node is also the logic cell. This is because pins on FPGA logic cells are logically equivalent. In following resource graph, source represent logic block itself, it will always connect output pin of this logic block wires are also nodes in rr graph, switches will be treated as edges, There are two types switch, one is unidirectional, the other is bidirectional. So for bidirectional there
will be two edges. Finally wires connect input pins and back to logic block. For black box, we build same resource graph, but it will have one source and one sink, even it may occupy several logic cells. No wires are allowed to be inside black box. All the rest is same as general logic cell. Each pin on black box is not logically equivalent, in final routing, if net pin is on black box we need to make sure that it will routing the specific pin.

Figure 3.9 Connections between Wires [17]
Figure 3.10 Routing with Black Boxes Fully Connected

Figure 3.10 shows implementation of a design on FPGA with black boxes. No wire is passed through the black boxes. In this project we assume that no wire cross over the embedded cores. However, in today’s FPGA limited number of wires is allowed to cross over. Our tool can be easily extended to include this capability as well.
In Figure 3.11, grey colored boxes are logic cells, red one is black box and green lines are switches. Yellow pin is output pin; blue pin is input pin, black line is y track or x track. Each pin on the logic cell or black boxes are fully connected to x tracks or y tracks. Above rr graph has uniform channels in all x and y direction. In our HVPR, it could also build rr graph with non uniform channels.
Figure 3.12 Single Segment Routing

In the above routing resource graph, the wire length is 1 which we call it single segment. For multiple segments, if wire length is 4, and now there is a black box, this wire must be split into several wires as indicated in following graph.
Originally, wire1 and wire2 are parts of one long wire. Since no wire could pass through the black box, the wire is split into two separate wires. In the resource graph, there will be two wire nodes. The following is a snapshot of FPGA routing with multiple segments. Compare to single segments, some wires pass switches without connections.
3.2.5 Placement

The purpose of placement is to locate the blocks with high connectivity close to each other, minimize the wiring resources and critical path in this net.

3.2.5.1 Compute delay between logic cells

In placement, we need to know the delay between two logic cells. In the original VPR code, because there are no black boxes, each logic cell is uniform, so we could use a
delta value to represent all logic cell delay if the delta x and delta y is same. For example the routing from logic cell (2,4) to logic cell (3,6) is same as logic cell (5,8) to logic cell (6,10) because they have same delta value (1,2). A timing driven router is employed to determine the shortest delay between two blocks. Usually we will choose the center of FPGA as start position then calculates the delay from this cell to all other ones. This delay will be treated as standard delta value.

But now our FPGA contains a black box, if we still use original method, the delta value may not calculate correctly. See following example, assume we have input pin on all side of logic cells, which means we could go to any pin of logic cell. Cell (2,0) to (3,3) is not same as (3,0) to (4,3) even they have same delta value (1,3).
Since we could not use delta value to represent all delays, I tried to calculate all logic cell delays, assume we have logic cells occupy $n$ rows and $m$ column, and calculate one logic cell to another one need $K$ time. So if we want to get delay from any logic cell to any logic cell we need $O(m^2n^2K)$ time, this is too long, if $m$ is above 30 $n$ is 30, it will run 3 hours for just computation. We need to find some alternative ways to get delay between logic cells in short time. The solution is as follows, first build a resource graph that allows wires to route through the black box, so this is as general resource graph then calculate the delta value compared to original one. In late placement we will use this
delta value as basic delay value, if we found there is black box between those two logic cells, we will multiply by a constant so the delay will be larger than basic delta. I will explain this in detail when come to placement part.

After we get standard delta value, we then build resource graph with black box, because we need this resource graph for routing.

3.2.5.2 Swap block

In placement process, we use path timing driven algorithm. The placement cost is linear congestion. VPR placement algorithm is based on simulated annealing. In .arch file the black box already has its fix position. So initially we will put logic cells which are not marked as black boxes on physical position. Then we will randomly choose the blocks to be moved and randomly get the target position. We have to make sure that target position will not be black box position. Then cost function will be applied to evaluate net wires and critical path delay time. If new net cost will reduce the original net cost, this move will be accepted, other wise if it goes the worse cost this move may still accepted provided, this could prevent placement goes into local minimum.

3.2.5.3 Placement cost function

Placement cost function has two portions. One is wire cost, the other is timing cost.

\[
\text{Wiring\_cos} = \sum_{i=1}^{N_{\text{net}}} q_i \left( \frac{bby(i)}{cav, x(i)^\beta} + \frac{bby(i)}{cav, y(i)^\beta} \right)
\]

\[
\text{Timing\_cos} = \sum_{i=1}^{N_{\text{net}}} \sum_{j=1}^{M_{\text{block}}} t_{ij}
\]
qi is a factor related to number of net terminals, if terminal is less than 3, it is 1, then it will slightly increased to 2.79 when terminals increased to 50, if terminals is greater than 50, qi will be increased linearly.

![Figure 3.16 Example of Placement Cost](image)

Timing driven algorithm [28] is based on bounding box. In above graph, pin on black box (4,4) is one pin in the whole net, we have total 7 terminals, even black box pin is on logic cell (4,4), because it is black box, and there are no wires through the black box, so for black box the xmin is 4 ymin is 3 xmax is 6 ymax is 5. Final bounding box, xmin is 1 ymin is 3, xmax = 7 ymax = 7.

The variables bbx is wires span on x direction, bby is wires span on y direction. For above graph, in x direction, from 1 to 7, wire is 7, and we have 5 rows so total wire in x
direction is $7 \times 5 = 35$, but we need to subtract number of wires inside black box, which is 3 in our case, so final result is 32. Same as y direction, wires in y direction is 26.

For timing cost portion, delay from (1,4) to (7,4) is based on delta value without black box, but since now we have a black box between them, so we multiply 1.1 to original one, if there are two black box we multiply another 1.1 and etc. This is for some penalty if there is some black box between two logic cells. How to find a way which is fairly evaluating delay time when there is black box between them will be an interesting topic in future research.

![Initial placement](image)

Figure 3.17 Initial placement
3.2.6 Routing

What is routing? Routing is connection between two nodes which represent pins on logic block in resource graph. Generally it will find short path between two nodes. But we also know there is capacity for each channel, router shall also avoid using too many limited wires in FPGA.
VPR uses a timing driven router [17] to perform routing. It is based on pathfinder routing algorithm [29] which will perform a fixed number of iterations, in each iteration the nets will be reroute to resolve the congestion. It will use Dijsktra [31] algorithm to find shortest path. Also when routing net section that is critical path, it will try to optimize timing by using short path. The route uses Elmore delay [30] to calculate routing cost between connections. The Elmore delay is a model which could calculate the source to sink delay time. A binary search algorithm is used to determine the width of routing channels. Initially, channel width is set at 2 times the number of pins in a CLB. Then the Dijsktra algorithm is used to find the shortest part between each node in whole net. If the net is successfully routed we will decrease the channel width by 50% and try again. If routing operation failed, we will double the routing channel width. Since we already modified the rr graph, and all routing is based on rr graph, so we do not need to do further modifications.
Figure 3.19 Placement and Route with Black Boxes
Chapter 4 – H-VPR Benchmarks

In order to find out if widen channels around logic cores improves final routing results, we need to compare some big and small benchmark circuits. We will use benchmark circuits provided by Altera Quip program and Synopsys flow.

4.1 Altera Benchmark Circuits

Altera Quip program provides benchmark circuits [32] which could be used to produce BLIF file for VPR application. However, if the benchmark circuits have RAM blocks, the BLIF file can not be generated. Also if the benchmark circuit has asynchronous signals, then the BLIF file can not be produced correctly. We finally selected four benchmarks circuits. Benchmark circuit oc_fpu and oc_cordic_r2p each contain more than 3 different sub-components. For VPR demonstration, we will generate a modified netlist so a black box replaces one of the components. For example in oc_fpu_multiplier circuit, the component multiplier will be replaced by a black box. From VHDL code we will remove the multiplier component. By default, this BLIF file contains 4LUT, from [34] we know LUT input pins 4 to 6 will be most area efficient. So we will use T-vpack with LUT input pin size equal to 4. In T-vpack, we will pack 1 LUT and 1 FF into one CLB. Then we will use Altera Quartus to generate the BLIF file and determine the number of logic cells of this benchmark circuit. We can then use Synopsys Design Compiler to determine the number of gates required for the multiplier component. The number of transistors required is four times the number of gates. Finally we divide the number of gates by 213 (number of transistors in one CLB) to determine the number of CLBs required to realize the multiplier.
For circuits oc_des_perf_opt and fip_cordic_cla, each circuit contains multiple instances of same component. To exercise VPR’s capability of handling multiple black boxes, we generate modified circuits with 1, 2 and 3 black boxes. For example oc_des_perf_opt_b2 will contain two black boxes and fip_cordic_cla_adder3 will contain three black boxes. Following table describe some attributes for each benchmark circuit.

Table 4.1 Summary of H-VPR Results on Benchmark Circuit

<table>
<thead>
<tr>
<th>Index</th>
<th>Benchmark Circuit</th>
<th>Logic cell</th>
<th>Array size</th>
<th>Black box logic cell</th>
<th>Black box gates</th>
<th>Black box transistors</th>
<th>Number of black box</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oc_fpu</td>
<td>8017</td>
<td>108*108</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_multiplier</td>
<td>7035</td>
<td>103*103</td>
<td>988</td>
<td>7775</td>
<td>31100</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_divider</td>
<td>3988</td>
<td>79 * 79</td>
<td>4011</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_post_norm</td>
<td>6576</td>
<td>101*101</td>
<td>1453</td>
<td>3442</td>
<td>13768</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Oc_cordic_r2p</td>
<td>2639</td>
<td>77 * 77</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_pre</td>
<td>2627</td>
<td>77 * 77</td>
<td>78</td>
<td>908</td>
<td>3632</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_cordic</td>
<td>441</td>
<td>41 * 41</td>
<td>2483</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_post</td>
<td>2410</td>
<td>73 * 73</td>
<td>248</td>
<td>2450</td>
<td>9800</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Oc_des_perf_opt</td>
<td>5336</td>
<td>100*100</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b1</td>
<td>5158</td>
<td>100*100</td>
<td>236</td>
<td>1493</td>
<td>5972</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b2</td>
<td>4980</td>
<td>100*100</td>
<td>236</td>
<td>1493</td>
<td>5972</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b3</td>
<td>4802</td>
<td>100*100</td>
<td>236</td>
<td>1493</td>
<td>5972</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Fip_cordic_cla</td>
<td>412</td>
<td>26 * 26</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder1</td>
<td>342</td>
<td>25 * 25</td>
<td>88</td>
<td>277</td>
<td>1108</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder2</td>
<td>271</td>
<td>24 * 24</td>
<td>88</td>
<td>277</td>
<td>1108</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder3</td>
<td>234</td>
<td>23 * 23</td>
<td>88</td>
<td>88</td>
<td>277</td>
<td>3</td>
</tr>
</tbody>
</table>

The divider component in oc_fpu could not be synthesized by Synopsys due to unavailability of divider cell in Synopsys library. However we could use Altera Quartus to get number of logic cells required for divider, which is 4011. For multiplier component it occupies 988 logic cells. That translates to 146 CLB required. This is an indication that 1 logic cell will have 0.14 CLBs. However for post_norm component it occupies 1453 logic cells, but only has 64 CLBs. This means one logic cell corresponds to 0.04 CLBs. So for the sizing the divider component, we choose a median value which
is one logic cell correspond to 0.1 CLB. So the divider component is calculated to require 401 CLBs. The same is applied to r2p_cordic component in oc_cordic_r2p. The component has 2483 logic cells, which correspond to 248 CLBs.

Table 4.2 shows some information about the individual circuit blocks within the benchmark circuit that will be used to encapsulate with a black box before feed into H-VPR tool for placement and route operation.

<table>
<thead>
<tr>
<th>Index</th>
<th>Benchmark</th>
<th>Black box clbs</th>
<th>Black box size</th>
<th>Number of black box</th>
<th>Input pins</th>
<th>Output pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oc_fpu_multiplier</td>
<td>146</td>
<td>12 * 12</td>
<td>1</td>
<td>75</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_divider</td>
<td>401</td>
<td>20 * 20</td>
<td>1</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_post_norm</td>
<td>64</td>
<td>8 * 8</td>
<td>1</td>
<td>75</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>Oc_cordic_r2p_pre</td>
<td>17</td>
<td>4 * 4</td>
<td>1</td>
<td>34</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_cordic</td>
<td>248</td>
<td>15 * 15</td>
<td>1</td>
<td>54</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_post</td>
<td>46</td>
<td>9 * 5</td>
<td>1</td>
<td>45</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>Oc_des_perf_opt_b1</td>
<td>28</td>
<td>7 * 4</td>
<td>1</td>
<td>80</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b2</td>
<td>28</td>
<td>7 * 4</td>
<td>2</td>
<td>80</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b3</td>
<td>28</td>
<td>7 * 4</td>
<td>3</td>
<td>80</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Fip_cordic_cla_adder1</td>
<td>5</td>
<td>2 * 2</td>
<td>1</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder2</td>
<td>5</td>
<td>2 * 2</td>
<td>2</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder3</td>
<td>5</td>
<td>2 * 2</td>
<td>3</td>
<td>34</td>
<td>17</td>
</tr>
</tbody>
</table>

### 4.2 Embed Blackbox Circuit using Synopsys Flow

In addition to Altera benchmark circuits, Synopsys also has a benchmark circuit named chip. The Synopsys tool only produces EDIF file. We would need to use `edif2blif`, a FPGA mapping application in order to get the final BLIF file. Table 4.3 summarizes some attributes of Synopsys benchmark circuit and its component circuit blocks within. Since only Altera provided logic cells of circuit, so there will be no logic cells column in the table for the Synopsys benchmark circuit.
### Table 4.3 Synopsys Benchmark Circuit Attributes

<table>
<thead>
<tr>
<th>Index</th>
<th>Benchmark Circuit</th>
<th>Array size</th>
<th>Black box gates</th>
<th>Black box transistors</th>
<th>Black box clbs</th>
<th>Black box size</th>
<th>Input/output pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chip</td>
<td>39*39</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Chip_add16x16</td>
<td>39*39</td>
<td>542</td>
<td>2168</td>
<td>10</td>
<td>5 * 2</td>
<td>34/17</td>
</tr>
<tr>
<td></td>
<td>Chip_mul16x16</td>
<td>30 * 30</td>
<td>3425</td>
<td>13700</td>
<td>64</td>
<td>8*8</td>
<td>33/32</td>
</tr>
<tr>
<td></td>
<td>Chip_mul8x8</td>
<td>37*37</td>
<td>751</td>
<td>3004</td>
<td>14</td>
<td>7 *2</td>
<td>17/16</td>
</tr>
</tbody>
</table>

### 4.3 H-VPR placement and routing

I used H-VPR to perform detailed routing on all above twenty benchmark circuits. All benchmark circuits were processed through the flow successfully. In following table, we compare the channel width requirement for original circuit and requirement for modified circuit with embedded logic core.

#### Table 4.4 Channel Width Comparison

<table>
<thead>
<tr>
<th>Index</th>
<th>Benchmark Circuit</th>
<th>Original channel width</th>
<th>Logic core Channel width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oc_fpu</td>
<td>12</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_multiplier</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_divider</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Oc_fpu_post norm</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Oc_cordic_r2p</td>
<td>7</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_pre</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_cordic</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Oc_cordic_r2p_post</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Oc_des_perf_opt</td>
<td>8</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b1</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b2</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Oc_des_perf_opt_b3</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Fip_cordic_cla</td>
<td>7</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder1</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder2</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Fip_cordic_cla_adder3</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>Chip</td>
<td>7</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Chip_add16x16</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Chip_mul16x16</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Chip_mul8x8</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
From table 4.4 we can determine that most of the benchmark circuits with logic core require increase in channel width. This could be attributed to presence of black box blocks increased the congestion in the whole circuit. In next chapter we will perform non-uniform channels routing and see if this would help to reduce the circuit area and improve circuit delay.
Chapter 5 - Design Space Exploration

In chapter 4, we created several benchmark circuits to demonstrate the effectiveness of H-FPGA flow with embedded circuits as black boxes. In order to determine if widen channels around logic core will improve criteria such as final tracks, critical time and circuit area, we will need to conduct detailed routing with timing driven algorithm twice. On first run the routing channel will be uniform. In the second time the channel width will be non-uniform. Then we will compare the routing tracks, critical time and circuit area to see what differences of using non-uniform routing would make.

5.1 Uniform channel routing

Timing driven algorithm uses binary search method to determine the channel width required for routing the design. For example, if the number of pins on a CLB block is 6, then original channel width is twice of 6 which correspond to 12. The channel width would 12 for the entire circuit area. Then the VPR will perform placement and routing based on uniform channel width. If after n number of iterations the routing is successful with channel width of 12, the channel width will be reduced to 6. If routing is not successful with channel width of 12, the channel width will be increase to 24. This process will be repeated until smallest channel width that can accommodate the circuit is found.

5.2 Non-uniform channel routing

For non-uniform channel routing, the number of channel width around the black box will be twice that of initial channel width for the rest of FPGA. An example is shown in the following figure. The size of black box is two by four. The channels wire1, wire2,
wire4 and wire5 are routing channels around the logic core. To accommodate potential congestion around the black box circuit, the channel adjacent to the black box will be assigned a width that is twice the initial channel width. So for those routing channels, the initial channel width will be 24. Depends on the size of the black box, we also assign more resources to channels further away from the black box. For example, if the width of the black box is 2, then we do not assign more resources to any channel other than those adjacent to the black box in the Y direction. The height of the black box is 4, so we assign more resource to 2 additional channels in the X direction around the black box. The amount of additional resource in the X channel is calculated as 1.5 times the initial width. So for channels wire3 and wire6 the channel width is 1.5 * 12 = 18. With the initial width of routing channels around the black box determined, the VPR would then perform placement and routing.
Figure 5.1 Increase Routing Channel Width for Areas around Black Box
5.3 Experimental Results

5.3.1 Total number of tracks

Total number of tracks criteria describes the total routing tracks required for the benchmark circuit. The following charts from Figure 5.2 to Figure 5.6 show the number of routing tracks required for all five benchmark circuits with uniform and non-uniform channel width. For original benchmark, there is no hard core, so we only conduct uniform channel test. Because each benchmark circuit contains several circuit blocks that can be encapsulated as black box, several benchmark circuits are created, each one reducing a particular circuit block into a black box.

![Figure 5.2 Routing Track Summary for Synopsys Benchmark Circuits](image_url)
Figure 5.3 Routing Track Summary for Oc_cordic_r2p Benchmark Circuit

Figure 5.4 Routing Track Summary for Fip_cordic_cla Benchmark Circuit
Figure 5.5 Routing Track Summary for Oc_des_perf_opt Benchmark Circuit

Figure 5.6 Routing Track Summary for Oc_fpu Benchmark Circuit
From Figure 5.7 we could see that for more than 85% of benchmark circuits, routing track efficiency was improved using non-uniform routing.

5.3.2 Delay Modeling

In chapter 3 we already described the VPR timing driven router. After routing is completed, a path based timing analysis [33] will evaluate the delays between each node. Then the critical path will be determined. The critical time is the time used for critical path within the circuit. Critical path is the longest path in the collection of nets. The following charts show the critical time for all benchmark circuit with uniform and non-uniform channel width. For original benchmark, since no hard core exists, only uniform channel test is conducted.
Figure 5.8 T-time Comparison for Synopsis Benchmark Circuit

Figure 5.9 T-time Comparison for Oc_cordic_r2p Benchmark Circuit
Figure 5.10 T-time Comparison for Fip_cordic_cla Benchmark Circuit

Figure 5.11 T-time Comparison for Oc_des_perf_opt Benchmark Circuit
Figure 5.12 T-time Comparison for Oc_fpu Benchmark Circuit

Figure 5.13 T-time Improvement Summary
From Figure 5.8 to 5.12, we can see critical time will be improved a lot if embedded hard core’s size is big enough. Otherwise critical time may even worse compare to original one. From Figure 5.13, for T critical time criteria, using non-uniform channel routing resulted in degraded performance for 63% of benchmark circuits compared to uniform channel routing.

5.3.3 Circuit Area

The total area criterion is the circuit area for the placed and routed circuit. This includes logic block area and the switches that connected to those logic blocks. Author of VPR used minimum width transistors area which is a layout area that requires by the smallest transistors. VPR will build the whole structure and sum the total number of required minimum width transistors to calculate the final circuit area.

![Figure 5.14 Final Circuit Area for Synopsys Benchmark Circuit](image-url)
Figure 5.15 Final Circuit Area for Oc_cordic_r2p Benchmark Circuit

Figure 5.16 Final Circuit Area for Fip_cordia_cla Benchmark Circuit
Figure 5.17 Final Circuit Area for Oc_des_perf_opt Benchmark Circuit

Figure 5.18 Final Circuit Area for Oc_fpu Benchmark Circuit
Figure 5.19 Circuit Area Summary for all Benchmark Circuits

For circuit area criteria, from Figure 5.19 we could see that 90% of benchmark circuits were improved using non-uniform routing. Compared to original benchmark, only those benchmarks contain big hard core, final area will be reduced, otherwise final area could be increased.
5.3.4 Conclusion

The above chart summarizes the key parameters when performing uniform and non-uniform channel detailed routing on various benchmark circuits. We could see that with non-uniform routing, the total number of routing tracks and circuit area are improved around 10%. However for critical time, there is some performance degradation between uniform and non-uniform routing for benchmark circuits.

5.3.5 Future work

In today’s FPGA, some FPGA allows limited wires through logic core, Modifications to our H-VPR needed to accommodate this new feature. Current the logic core position is fixed during placement. We could also modify H-VPR to make logic core position moveable and test with multiple benchmarks to find the best location for most benchmark.
Chapter 6 - Summary and Conclusion

Traditional FPGA devices were homogeneous, consists of just logic elements and routing resources. Driving by demand and technology advances, state-of-art FPGA devices are becoming heterogeneous by embedded hard macro IP cores within the FPGA fabric. Versatile Place and Route (VPR) tool was used in academic arena to study the impact of various FPGA architectures on placement and routing. Since VPR does not support embedded IP blocks, there exist a need to enhance the capability of VPR for heterogeneous FPGA devices.

We have developed the heterogeneous FPGA (H-FPGA) CAD flow to take a circuit with embedded IP cores from HDL level to placed and routed implementation. We have modified VPR source code extensively to support placement and route with embedded black boxes. The enhanced tool is called heterogeneous VPR (H-VPR). H-VPR is targeted for UNIX platform, but can be used on a PC using Cygwin environment. We have demonstrated the flow and tool capability on a number of benchmark circuits of various size and complexity. The H-FPGA flow and associate H-VPR tool would enable researchers to perform design space exploration of various FPGA architectures.
References

1. Jeff Mitchell, President, Digital Solutions, MacTutor Contributing Editor
   “Programmable Logic Devices”,
   http://www.mactech.com/articles/mactech/Vol.01/01.11/PAL/

   http://www.netrino.com/Articles/ProgrammableLogic/

3. Monolithic Memories Inc. acquired by Lattice Semiconductor in 1999


    Research,” Seventh International Workshop on Field-Programmable Logic and

13. J. He, J. Rose, "Advantages of Heterogeneous Logic Block Architectures for
    FPGAs," IEEE Custom Integrated Circuits Conf. 1993, (CICC 93), San Diego, May
    1993 pp. 7.4.1 - 7.4.5.


22. Synthesis Design Flows Using the Quartus University Interface Program (QUIP).
   “Quip_synthesis_interaces.doc” in the QUIP package. www.altera.com


32. Benchmark Designs for The Quartus University Interface Program (QUIP) Version 1.0 May 24, 2005
