Technology Mapping and Architecture of Heterogeneous Field-Programmable Gate Arrays

by

Jianshe He

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science in the
Dept. of Electrical and Computer Engineering
University of Toronto, Toronto
Ontario, Canada

Abstract

As Field-Programmable Gate Arrays (FPGAs) become more accepted and integral to the digital design process, there will be a strong drive to produce faster and higher-density devices. One architectural dimension that needs to be explored for its speed and density benefits is that of *heterogenous* FPGAs which employ more than one basic kind of logic block instead of the common homogeneous FPGA with identical logic blocks. There are two architectural reasons to believe that an FPGA with a selection of heterogenous blocks will provide superior speed and density:
1. Different logic may be more efficiently implemented with different kinds of blocks.
2. Previous studies have shown that coarse grain blocks exhibit superior speed to

fine grain blocks, yet the smaller blocks have better density. A mixture of the two

may provide superior speed-area tradeoff.

This thesis makes two contributions in the area of heterogeneous FPGAs. First it presents a technology mapping algorithm for heterogenous FPGAs with two different sizes of lookup table (LUT) logic block. Synthesis for this type of FPGA is more difficult than for homogeneous FPGAs because the cost function is not a linear count of the number of LUTs used. To solve this problem a general optimization approach is proposed and applied at the tree-level and across multiple trees. The latter algorithm is shown to be optimal. Experimental results show that this approach is superior to synthesizing heterogeneous FPGAs from a post-process of a homogenous mapper.

Secondly, the thesis also presents an architectural investigation into the area-efficiency of heterogeneous FPGAs. Experimental results on a set of benchmark circuits indicate that several heterogeneous architectures achieve significant reduction in the number of programming bits and logic block pins compared to the best homogeneous FPGA: a 4-input lookup table (4-LUT) FPGA. Furthermore, a 6-LUT/4-LUT combination will likely exhibit better performance with nearly equivalent area to a homogeneous 4-LUT FPGA.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Jonathan Rose, for his guidance and support. His patience, wisdom, and advice have provided me with a continuous source of inspiration and encouragement. I also thank Jonathan for introducing me to the exciting FPGA world and later to the heterogeneous FPGA area.

I would like to thank Dr. Bob Francis, Kevin Chung, Qing Zheng, Qiang Wang, and Keith Farkas for many helpful discussions. Many thanks to the Ph.D and Master's students and staff of the Computer and Electronics Groups for their cooperation.

Financial assistance from Information Technology Research Center is gratefully acknowledged.

Last but not the least, I thank my wife, Yingchun, for her continuous understanding, encouragement, and especially wholehearted support in bringing up our son, George, while she was studying.

Contents

Chapter 1

Introduction

1.1 Field-Programmable Gate Arrays

The Field-Programmable Gate Array(FPGA) is a new implementation medium for Application Specific Integrated Circuits (ASICs). It provides the user with large scale integration and user programmability. An FPGA consists of a general array of uncommitted logic blocks that are interconnected by programmable routing switches. The user can program and sometimes reprogram an FPGA into different devices by turning on or turning off these switches in the "field". The user-programmability of the FPGA results in a reduction of the turn-around time from months to minutes and cuts the manufacturing costs of a prototype of ASIC by a factor of a thousand over traditional fabrication techniques.

While the FPGA offers significant advantages over traditional ASIC fabrication technologies, such as Mask-Programmable Gate Arrays (MPGAs), it has lower density and slower speed due to the technology used in the programmable switches. In commercial architectures this programming technology is one of pass transistors driven by static RAM, anti-fuses, or floating gate transistors [Brow92]. In all cases these programming technologies have large resistance and capacitance which require much larger area and cause more delay than the metal wires used in an MPGA to make connections. As a result, an FPGA has lower logic density and is slower than an MPGA. The effect of these two drawbacks can be reduced by careful choice of the FPGA *architecture.*

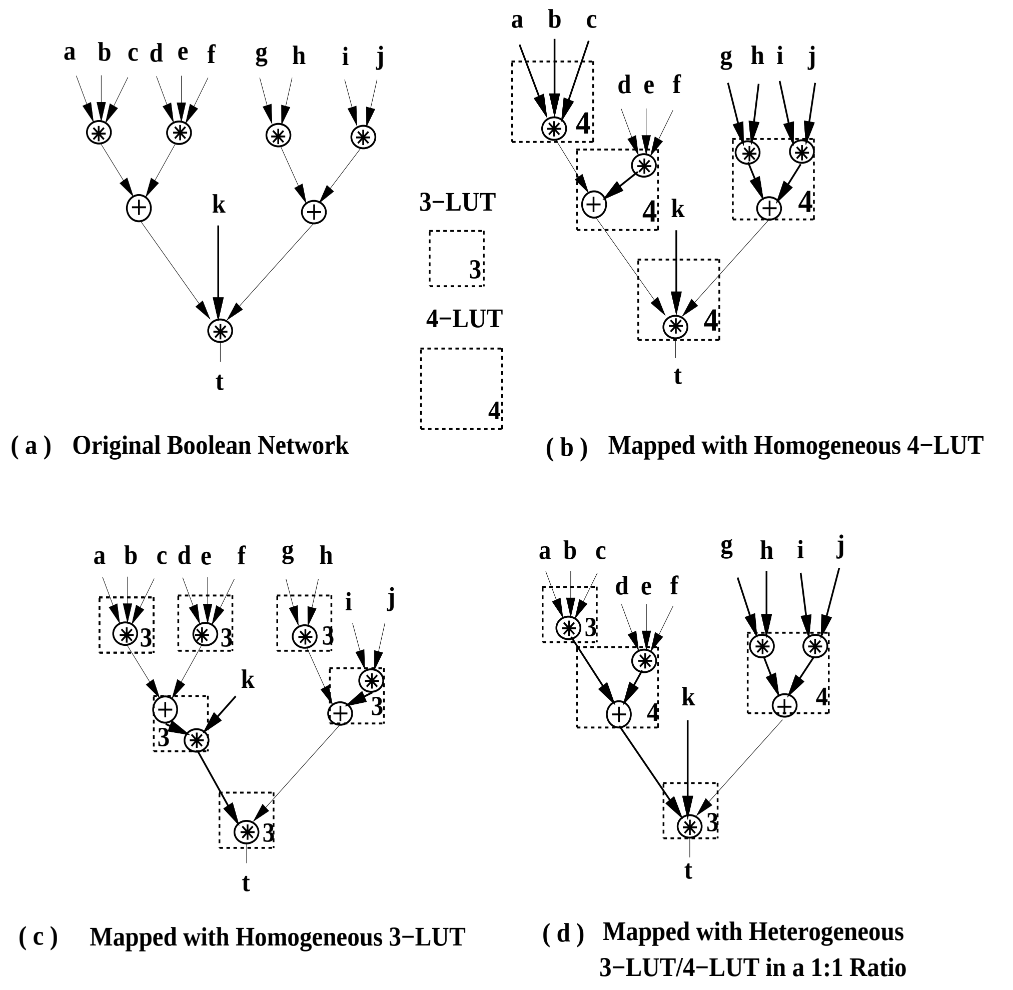
1.2 Thesis Motivation

Most FPGAs consist of an array of identical logic blocks [Cart86] [Hsie90] [ElGa89] [Ahre90] [Wong89] [Wils92] [Algo89], or logic blocks that have very similar levels of functionality. Such an FPGA is termed *homogeneous*. Previous studies on logic block architecture [Rose90] [Koul92] have concluded that while 4-input lookup tables (4-LUTs) make efficient use of area, more coarse-grained blocks such as 5-LUTs, 6-LUTs and 7-LUTs are superior in terms of system speed [Koul91] [Sing91] [Sing92]. These results suggest that a mixture of different size LUTs (for example 4-LUTs and 6-LUTs) may provide a better tradeoff between speed and density.

Furthermore, all such studies have considered only the homogeneous case; it is possible that a heterogeneous mixture of logic blocks may provide superior area-efficiency (which relates to logic density) because some parts of logic may simply be more efficiently implemented with one particular type of logic block than another. For example, consider the boolean network pictured in Figure 1.1a. Figure 1.1b is a mapping of that network using 4-LUTs and Figure 1.1c is a mapping of that network using 3-LUTs. As shown in Table 1.1, the 4-LUT solution uses one-third more lookup table bits (64 vs 48) but 20% fewer pins than the 3-LUT solution. Note that a single 3-LUT has 8 bits and 4 pins and a single 4-LUT uses 16 bits and 5 pins. Pin counts and bit counts are common metrics of area cost in a LUT-based FPGA. We will discuss these metrics in more detail in Chapter 4.

Now suppose that the network is mapped into a heterogeneous FPGA that contains 3-LUTs and 4-LUTs in equal numbers, as illustrated in Figure 1.1d. This circuit uses exactly two 3-LUTs and two 4-LUTs and hence requires only 48 bits and 18 pins. This heterogeneous FPGA requires 25% fewer bits and 10% fewer pins than the 4-LUT homogenous FPGA implementation. It also has the same number of bits and 25% fewer pins than the 3-LUT homogenous FPGA to implement this example. While this is a “cooked” example, it demonstrates that a heterogeneous mixture of logic blocks *may* exhibit superior area-efficiency. In practice, this situation often occurs.

One difficulty with heterogeneous FPGAs is that there are no CAD tools aimed at solving the heterogeneous mapping problem. Although many technology mappers have been developed for homogeneous architectures [Fran90] [Fran91] [Fran91b] [Murg90] [Murg91a] [Murg91b], [Abou90] [Filo91] [Karp91] [Woo91] [Chen92] [Cong92] [Sawk92], to our knowledge there is no prior re-



a b c d e f g h i j
k
t
3−LUT
3
4−LUT
4
(a) Original Boolean Network
(b) Mapped with Homogeneous 4−LUT
(c) Mapped with Homogeneous 3−LUT
(d) Mapped with Heterogeneous 3−LUT/4−LUT in a 1:1 Ratio

Figure 1.1: An illustration of homogeneous and heterogeneous mappings

	LUT types	#LUTs	#Bits	#Pins
Homogeneous	only 3-LUT	6	48	24
	only 4-LUT	4	64	20
Heterogeneous	3-LUT/4-LUT	2/2	48	18

Table 1.1: Comparison of Hetero. vs. Homo. FPGAs for Circuit in Figure 1.1

search on the heterogeneous problem.

It should be noted that Xilinx's 4000 series FPGA does use two sizes of lookup table. However, in the Xilinx 4000, two 4-LUTs are *hardwired* to a 3-LUT and so are tightly linked. The notion of heterogeneity used in this thesis is that the different logic blocks must be completely independent in the routing.

The goal of this thesis is two-fold:

1. Develop a technology mapping algorithm for heterogeneous FPGAs and
implement it.

2. Use this new CAD tool to investigate the advantages of heterogeneous FPGAs
using an empirical approach: implementing benchmark circuits in a variety of
heterogeneous FPGAs and measuring their area-efficiency.

1.3 Thesis Organization

This thesis is organized as follows: Chapter 2 provides the necessary background and discusses previous related work. Chapter 3 describes a new technology mapping algorithm for heterogeneous FPGAs and compares the algorithm's effectiveness with a modified homogeneous mapper. Chapter 4 presents the experimental method used to evaluate a set of heterogeneous architectures and gives the experimental results and conclusions. The final chapter contains the conclusions of the thesis work and suggestions for future work.

Chapter 2

Background and Previous Work

This chapter presents the background and previous related work on heterogeneous FPGAs. This includes lookup table (LUT)-based FPGA technology mapping and previous work on heterogeneous FPGA architecture. Finally, the architectures of commercial FPGAs with some heterogeneity are briefly described.

2.1 LUT and LUT-Based FPGAs

The logic block of an FPGA is used to implement combinational and sequential logic. Logic blocks can be designed in many different ways. Some FPGA logic blocks are as simple as 2-input NAND gates, while other blocks have more complex structures, such as lookup tables, multiplexers, or very wide-input AND-OR structures.

Previous studies have shown that lookup tables are a good choice due to their high functionality [Sing91] and now LUT-based FPGAs are the most common type of commercial FPGAs [Hsie90] [Wils92] [Brit93]. A K-input lookup table (K-LUT) is a digital memory with K address lines and a one-bit output. This memory contains 2^K bits and is capable of implementing any boolean function of K variables. A lookup table of K inputs can implement 2^{2^K} different boolean functions.

A simple example of a 2-input lookup table implementing a function $f = a + \bar{b}$ is shown in Figure 2.1, where the LUT is described in terms of a multiplexer. In the lookup table, there are two address lines (a and b) and 4 (2^2) memory cells. The inputs to the lookup table are the select lines of the multiplexer and the memory cells serve as the inputs to the multiplexer. If $a = 0$ and $b = 1$, for example, then the memory cell corresponding to select line **01** will be connected to the output f. The contents of the memory cells are calculated from the evaluation of $f = a + \bar{b}$ for all the combinations of logic values of a and b.

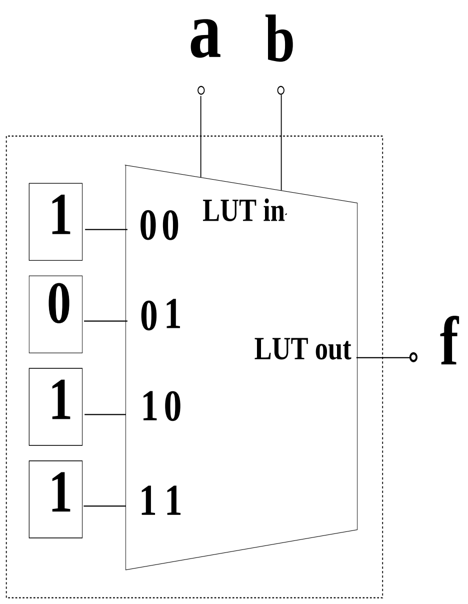


Figure 2.1: An Illustration of 2-LUT Implementing $f = a + \bar{b}$

2.2 Technology Mapping for LUT-Based FPGAs

As logic design becomes more complicated, there is an increasing need for automatic synthesis of logic. Logic synthesis is the process of translating a register-transfer-level description of a design into a gate-level representation. Typically, there are two separate phases in the logic synthesis:

technology-independent logic optimization and technology mapping [Bray90]. The technology-independent logic optimization step optimizes a network by algebraic and boolean simplifications such as removing redundancy and eliminating common sub-expressions to reduce the complexity of the network. As shown in Figure 2.2, the inputs to the logic synthesis or logic optimization step are boolean equations or its graph representation, the directed acyclic graph (DAG). The output of this step is an optimized network. Since this phase doesn't consider the type of block that will be used for the final circuit, it is called technology-independent logic optimization.

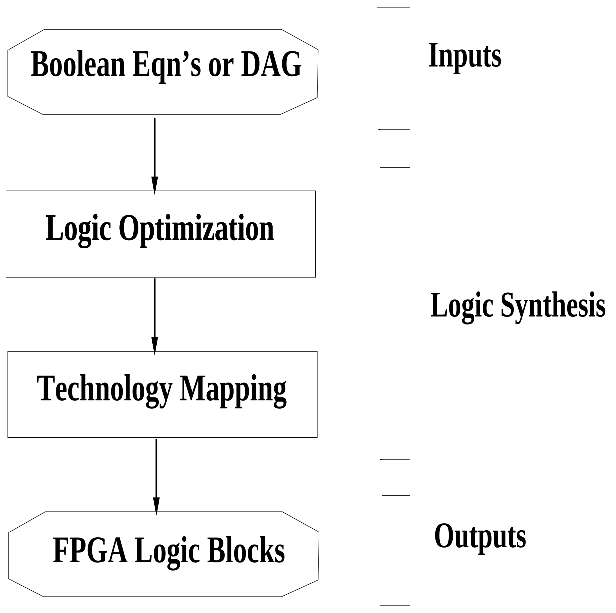


Figure 2.2: The Flow of Logic Synthesis

For FPGAs, the technology mapping step transforms the optimized boolean expressions into a circuit using the FPGA's particular logic blocks. The goal of technology mapping is to optimize the resulting netlist of logic blocks for area, delay, or some combination of area and delay constraints. Technology mapping for lookup table-based FPGAs is the problem of mapping a given DAG into LUTs with a fixed set of sizes. The technology mapping problem is called *homogeneous* if only one size of LUT is used in the mapping; otherwise it is called *heterogeneous*.

Many LUT technology mappers have been developed in the past few years, including Chortle [Fran90] [Fran91a] [Fran91b], MIS-pga [Murg90] [Murg91a] [Murg91b], Asyl [Abou90], Hydra [Filo901], Xmap [Karp91a], VISMAP [Woo91a], and DAG-Map [Cong92]. All of these map a boolean network into a homogeneous circuit of K-input LUTs, attempting to minimize either the total number of LUTs or the number of levels of LUTs (to improve speed). In the following section, only the Chortle technology mapper is described. A summary of other algorithms is provided in [Brow92] and [Fran92a].

2.2.1 The Chortle Technology Mapper

Chortle has two versions, Chortle-crf and Chortle-d. The former is used to minimize the total number of lookup tables, while the latter minimizes the number of levels of logic blocks. Only Chortle-crf is described below.

The input to Chortle is a boolean network in the form of a DAG of only AND, OR, and NOT nodes. The DAG is first partitioned into a forest of fanout-free trees and each tree is separately mapped into a K-input lookup table circuit using dynamic programming. Each tree is traversed from its leaf nodes to its root. At each node, an optimal circuit of LUTs is constructed implementing the cone rooted at this node and extending to the leaves of the tree. To find the solution at the current node, only the immediate fanin information is important. The order of the traversal ensures that these immediate fanin circuits have been previously constructed.

At each node during the traversal, the problem solved by Chortle is how to implement the current node and its fanin LUTs by using the fewest number of LUTs possible. Chortle converts this problem into a bin-packing problem. Given the previously mapped fanin LUTs (called "boxes" in [Fran91a]) to each node, the bin-packing problem is to find the smallest number of "bins" (the set of resulting LUTs) that the boxes can be fitted into.

Figure 2.3 illustrates how a node is mapped. In the figure, each dashed rectangle represents a lookup table whose size is $K = 5$. The figure illustrates the point where all of the fanins to the current node have already been mapped, but the node itself has not yet been mapped. Clearly, the LUTs in Figure 2.3a are not used to their full capacity. These LUTs can be packed into a set of "bins" to achieve higher utilization. Figure 2.3b shows such a re-arrangement. The fanin LUTs O1 and O2 have been packed into a new LUT output at O7, and the fanin LUTs O3 and O4 are packed into the LUT output at O8. Note that the packing process is accompanied by a gate-level decomposition of the current node.

The bin-packing is a process for re-arranging the *fanin* LUTs with a decomposition of the current node. To this point the current node has not been completely implemented. In realizing the current node, Chortle makes further use of the unused inputs of the packed LUTs by placing a LUT's output into another LUT's input if possible. In figure 2.3c the node is further decomposed to connect all the packed LUTs. Output O7 is placed into the LUT that produces output O8. As few used inputs as possible are packed into the root LUT to benefit implementation of the next stage, since the root LUT of the current node will be the fanin LUT of the next stage.

[Fran92a] proves that for each input tree, Chortle generates an optimal tree implementation provided that the value of K is less than or equal to 5. However, partitioning the original DAG into a set of fanout free trees makes the final solution sub-optimal. To improve the quality of the mapping, Chortle exploits reconvergent paths and replicates logic at fanout nodes after each mapped tree is assembled into a circuit [Fran91a].

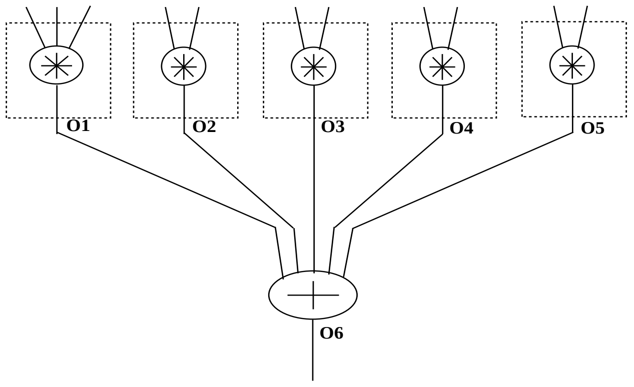
2.3 Previous Work on Heterogeneous FPGAs

There were two previous projects that performed initial investigations of heterogeneous FPGA architecture. Carl Mizuyabu's Bachelor's thesis [Mizu92] looked at heterogeneous LUTs and Keith Farkas investigated heterogeneous NAND gates [Fark92], in a course project.

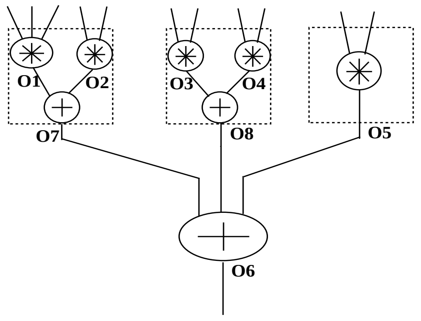
2.3.1 Heterogeneous NAND Gates

Technology Mapping

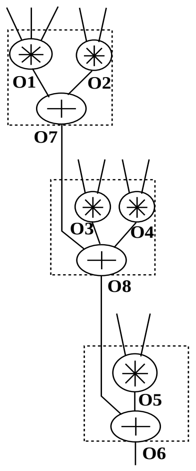
[Fark92] employed an experimental approach to investigate heterogeneous NAND gates based FPGAs. One key issue here is the technology mapping problem for heterogeneous NAND gates. Since there are no existing software



(a) Circuit Before Bin−Packing



(b) Circuit After Bin−Packing



(c) Circuit After Further Decomposition

Figure 2.3: An Illustration of Node-Mapping in Chortle

tools, as an approximation, the MIS [Bray87] software package was used as a technology mapper to map the input boolean network into a circuit of two sizes of NAND gates. In this work, the concept of a *supertile* was introduced. A supertile is defined as a group of a single primary gate (p-NAND) plus a number of secondary gates (s-NANDs), where $p > s$. Another important parameter is the ratio of the number of s-NANDs to p-NANDs, r, in each supertile. The heterogeneous nature of an FPGA is thus characterized by p, s, and r. The goal of the mapping is to minimize the number of supertiles to implement a boolean network, given p, s, and r. The mapping process used was as follows:

First, logic minimization and technology mapping was performed by using MIS to implement the given network into an FPGA of p-NANDs and s-NANDs. This is done by creating a library of p-NAND and s-NAND, each of which has a different cost. MIS will map the nodes of the network into either a p-NAND or an s-NAND or a mixture of them depending on their cost. The cost can affect the use of the NANDs, but the ratio r cannot be controlled, because the MIS algorithm cannot be constrained in this way.

After mapping, the number of p-NANDs and the number of s-NANDs are counted. If the number of p-NANDs used is N_p and the number of s-NANDs used is N_s, then the number of supertiles, N_{sup}, for a given value of r, can be calculated from equation (2.1):

$$N_{sup} = \mathbf{max}(N_p, \lceil \frac{N_s}{r} \rceil) \tag{2.1}$$

To minimize N_{sup}, the two terms in the **max** function of equation (2.1) should be equalized. In case $N_p \neq \lceil \frac{N_s}{r} \rceil$, conversion of a p-NAND into an s-NAND or a set of s-NANDs into a p-NAND is made. Figure 2.4 illustrates such a conversion where five 2-NANDs are used to replace a 4-NAND. Note that the conversion procedure will always improve or maintain the supertile count, never making it worse.

Architectural Investigation

To test the hypothesis that a heterogeneous NAND-gate FPGA will have superior density to a homogeneous one, 57 MCNC benchmark circuits were mapped into heterogeneous FPGAs with different combinations of p and s and then the number of supertiles were counted. Since supertiles for different

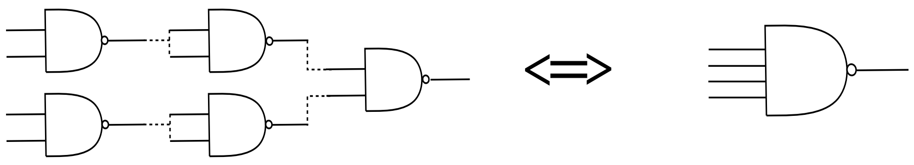


Figure 2.4: A set of 2-NANDs used to make a 4-NAND

p, s, and r have different sizes, comparison of number of supertiles cannot be used for the evaluation of area. Thus [Fark92] uses an area cost function based on the total number of gate pins as the routing area from pins is far more important than the active area used by logic blocks. The total number of pins of an FPGA is calculated by multiplying the total number of supertiles by the number of I/O pins per supertile.

[Fark92] finds that in the best case a 17% reduction in pins is achieved by using heterogeneous logic blocks. The best (p, s) is found to be (3, 2), (4, 2), or (5, 2), with ratios of three to five 2-NANDs to one of the larger size NAND gate.

2.3.2 Investigation of LUT-Based Heterogeneous FPGAs:

Heterogeneous architectural investigation of LUT-based FPGAs was first tried in [Mizu92]. Since there was no heterogeneous technology mapper available then, the investigation was done using the Chortle homogeneous mapper [Fran92a] and some post-processing of the output. After running the homogeneous mapper, a mapped circuit consisting of homogeneous K-LUTs was obtained. Since not all inputs of the LUTs will be completely used, those LUTs whose number of used inputs is fewer than K may be replaced by LUTs with fewer than K inputs. In this way a heterogeneous FPGA circuit may be realized from processing a homogeneous mapping. This method of mapping is called *post-process-homo* (PPH). In the following discussion, the same terminology is used as those in mapping the NAND gates in section 2.3.1 except that it is applied to lookup table logic blocks.

Technology Mapping: Post-Process of a Homogeneous Mapper

The post-process-homo method works as follows: to map a network with a given p, s, and r, the homogeneous mapper is executed to map the network into LUTs of size p. For each LUT produced by the homogeneous mapper, let the number of inputs that are actually used be u. The LUTs for which $u \leq s$, are mapped directly into s-LUTs, resulting in N_s s-LUTs. The LUTs for which $s < u \leq p$ are mapped into p-LUTs, resulting in N_p p-LUTs. The number of supertiles N_{sup} can be calculated from equation (2.1). Whenever there is an imbalance of N_p and $\lceil \frac{N_s}{r} \rceil$, a transformation between p-LUTs and s-LUTs can always be beneficial. The transformation process is described as follows:

When the post-process-homo results in fewer p-LUTs than $\lceil \frac{N_s}{r} \rceil$ s-LUTs, that is, if $N_p < \lceil \frac{N_s}{r} \rceil$, then some of the p-LUTs will be left unused if we take an FPGA with $\max(N_p, \lceil \frac{N_s}{r} \rceil)$ supertiles. In this case the extra p-LUTs in the supertiles can be used to implement s-LUTs to reduce the number of supertiles. Note that each p-LUT can be used to replace one and only one s-LUT. Let $\triangle_s$ be the total number of s-LUTs converted from p-LUTs, then to minimize the number of supertiles we have

$N_p + \triangle_s = \lceil \frac{N_s - \triangle_s}{r} \rceil$

Solving this equation for $\triangle_s$ results in

$\triangle_s = \lfloor \frac{N_s - rN_p}{r+1} \rfloor$.

In the case that $N_p > \lceil \frac{N_s}{r} \rceil$, however, more p-LUTs are needed to achieve a balance and some of the s-LUTs can be transformed into p-LUTs. Since an s-LUT has less functionality than a p-LUT, several s-LUTs are needed to implement a p-LUT. The number of s-LUTs needed for a p-LUT depends on both s and p. Figure 2.5 shows how three 3-LUTs are converted to a 4-LUT, where the last LUT serves as a 2-input multiplexer with selection input d. Table 2.1 gives the number of s-LUTs needed to implement a p-LUT for various combinations of p and s [Mizu92]. We refer to this number as n.

Given p, s, r, and n, the number of p-LUTs needed to be transformed from s-LUTs can be easily calculated. Let $\triangle_p$ be the total number of p-LUTs converted from s-LUTs, then we have

$N_p - \triangle_p = \lceil \frac{N_s + n\triangle_p}{r} \rceil$

Further solving for $\triangle_p$ gives $\triangle_p = \lfloor \frac{rN_p - N_s}{r+n} \rfloor$.

Architectural Results

Based on the method described in the previous subsection, [Mizu92] con-

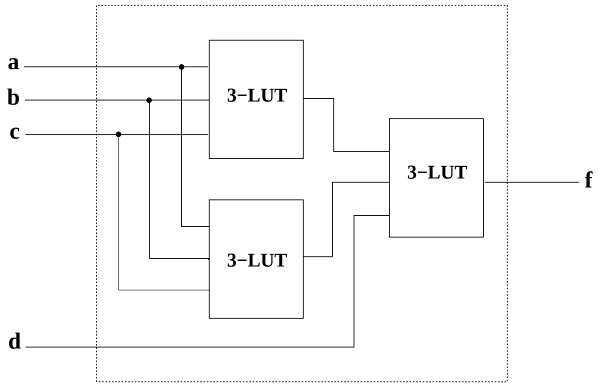


Figure 2.5: Realization of a 4-LUT by using three 3-LUTs

n		**p**				
		3	**4**	**5**	**6**	**7**
s	**2**	5	13	29	61	125
	3		3	7	15	31
	4			3	7	15
	5				3	7
	6					3

Table 2.1: Number of s-LUTs required to implement a p-LUT

ducted many experiments using the Chortle technology mapper to make an area comparison between homogeneous and heterogeneous FPGAs. The overall conclusion is that there is no advantage in using heterogenous FPGA for area. [Mizu92] attributed this failure to the poor quality of the heterogeneous mapper. However, from Table 2.1 it is clear that the transformation of small LUTs to a large LUT will cause a significant increase in the number of pins and bits. For example, to make one 5-LUT, three 4-LUTs are needed. However, using one 5-LUT requires only 6 pins and $2^5 = 32$ bits, while 3 4-LUTs costs $3 \times (4+1) = 15$ pins and $3 \times 2^4 = 48$ bits, which is much worse than an individual 5-LUT.

This method can be improved by translating s-LUTs into an u-LUT instead of a p-LUT when $N_p > \lceil \frac{N_s}{r} \rceil$, where u is the number of used inputs of a p-LUT after homogeneous mapping and $u \leq p$. Detailed discussion of this improvement is postponed to Chapter 3.

2.4 Commercial FPGAs with Heterogeneity

2.4.1 Xilinx FPGAs

Xilinx FPGAs are a symmetrical array of logic blocks called Configurable Logic Blocks (CLBs). There are three generations of Xilinx FPGAs, the XC2000, XC3000, and XC4000. The core component of the CLBs of all the three generation FPGAs are lookup tables. As shown in Figure 2.6 (a), the XC4000 CLB contains two kinds of LUT, two 4-LUTs hardwired into a 3-LUT [Hsie90].

This kind of architecture contains limited heterogeneity. By “limited” it is meant that the LUTs are not entirely independent, since outputs of the two 4-LUTs must be hardwired to a 3-LUT. This constraint may prevent the free selection of LUTs in the mapping.

It is interesting that the Xilinx 3000 FPGAs also has an aspect of heterogeneity. As shown in Figure 2.6 (b), the XC3000 logic blocks are made of two 4-LUTs fed into a multiplexer. The two 4-LUTs can be either used as separate LUTs or made into a 5-LUT as in Figure 2.7. This kind of architecture can be used to make a heterogeneous FPGA provided that a mapper can make use of the heterogeneous property. Note that the XC2000 has similar heterogeneous property of the XC3000 except that the XC2000 logic block is

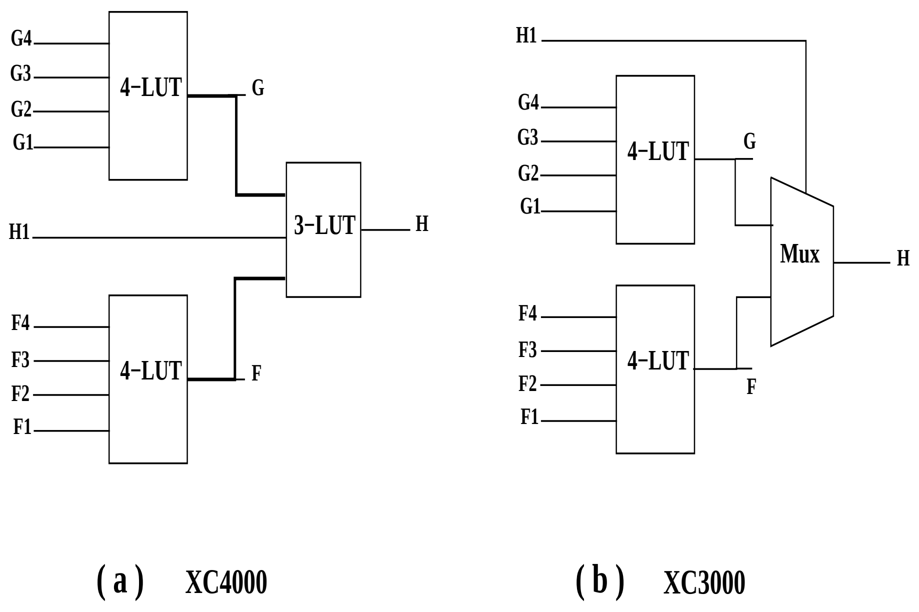


Figure 2.6: XC3000 and XC4000 Logic Block Architectures.

made of two 3-LUTs fed into a multiplexer.

2.4.2 The Architecture of AT&T ORCA FPGAs

AT&T's second generation of SRAM-based FPGAs, the Optimized Reconfigurable Cell Array (ORCA) [Brit93], also contains some aspects of heterogeneity. While the ORCA FPGA consists of array of identical, symmetrical logic blocks, called Programmable Logic Cells (PLCs), the PLCs can be used to implement different functions. Figure 2.8 shows a PLC block diagram. There are four 4-LUTs and four registers in each PLC. The four 4-LUTs can be used individually as 4-LUTs or used as into two 5-LUTs or one 6-LUT by properly programming the multiplexers shown in Figure 2.7.

This property of variable functionality of the logic block offers some of the advantages of heterogeneity. In the critical paths, PLCs can be made into coarse-grain blocks (6-LUTs, for example) to reduce the number of levels of logic blocks, while in the non-critical paths PLCs can be used as 4-LUTs to improve density (since 4-LUT is best for area [Rose90] [Koul92]). The

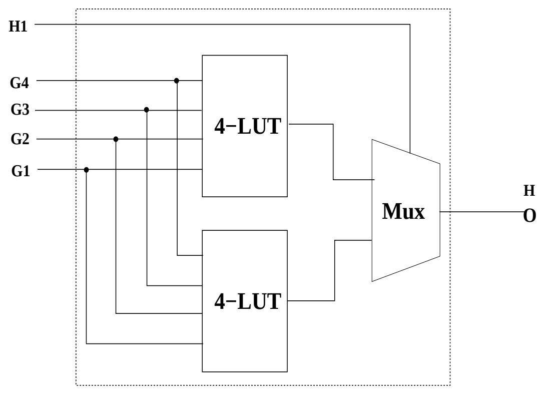


Figure 2.7: Two 4-LUTs is transformed into a 5-LUT

ORCA architecture has the significant advantage that during the technology mapping it is not necessary to control the ratio of the number of coarse-grained LUTs and the number of fine-grained LUTs. The conversion of 4-LUTs into larger-than 4-LUTs is done only when needed. However, the architecture also has a significant drawback: to make a 6-LUT from four 4-LUTs, the routing resources needed to connect to the many more pins of the four 4-LUTs are still in place, making this 6-LUT very expensive in area.

2.4.3 The Altera MAX Architecture

The logic block architecture of the Altera MAX EPLD family [Wong89] offers another example of a heterogeneous FPGA. The MAX FPGAs consist of an array of Logic Array Blocks (LABs), interconnected by a routing resource called the Programmable Interconnect Array (PIA). The general architecture of Altera FPGAs is shown in Figure 2.9 [Wong89].

Each LAB contains two kinds of logic blocks, the macrocells and the expanders, as illustrated in Figure 2.10. There are between 16 and 32 macrocells in a LAB. In each of the macrocells three product terms are ORed together to feed into a register or bypass it. Expanders are single product terms with inverted outputs feeding back into the LAB. A simplified diagram of the macrocell and expander is given in Figure 2.11. The expanders are shared by all the macrocells in a LAB. If there are more than three product terms are required, the expanders will meet such a need. This kind of architecture

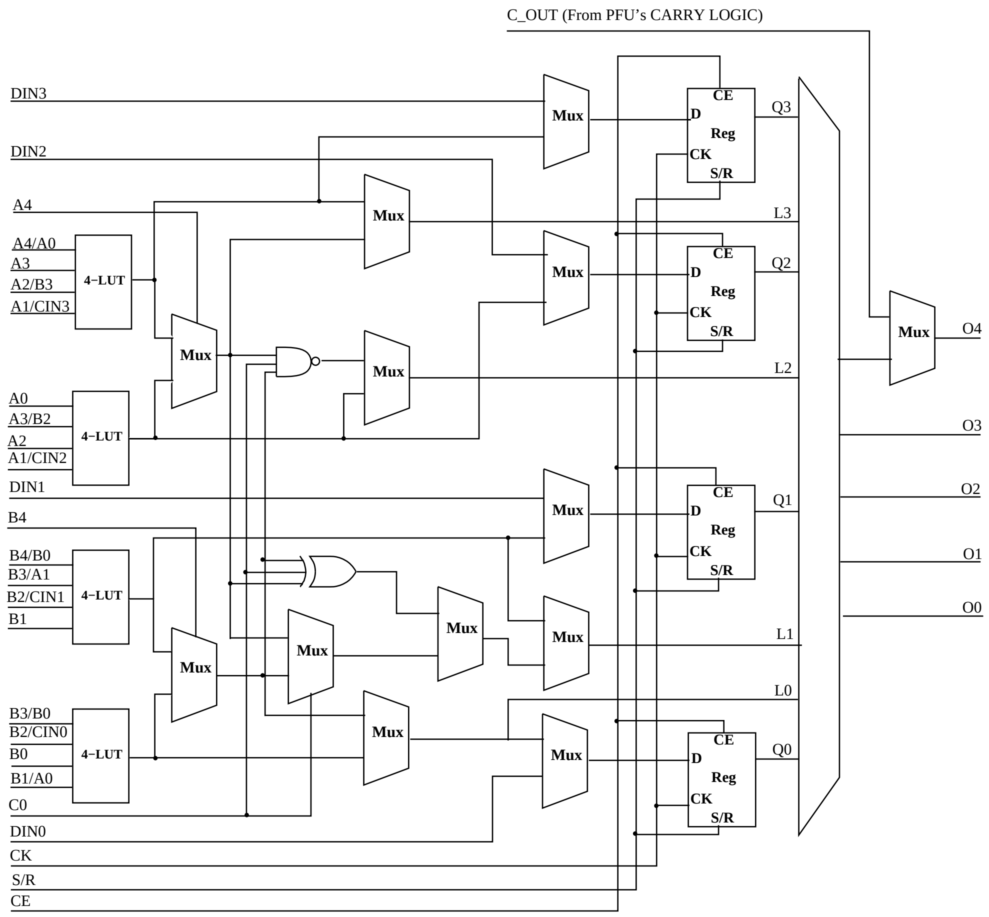


Figure 2.8: AT&T PLC Block Diagram

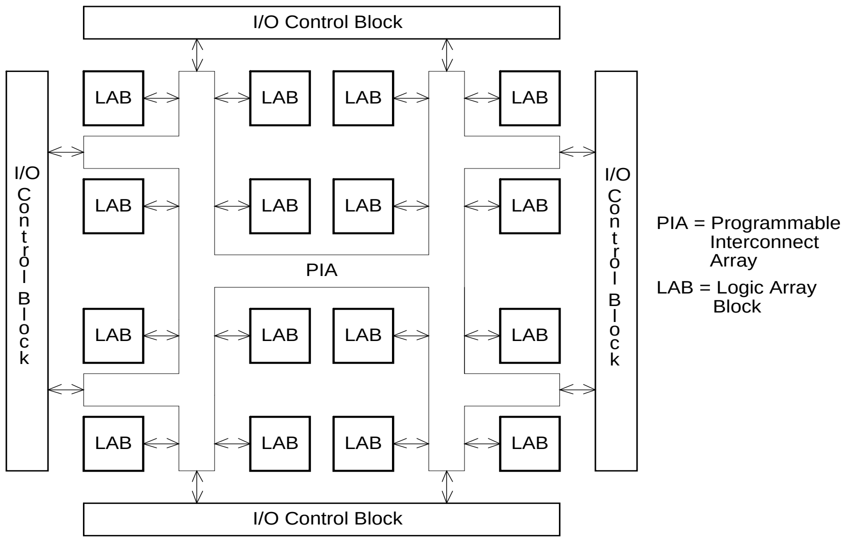


Figure 2.9: Altera FPGA Architecture (Figure From [Brow92])

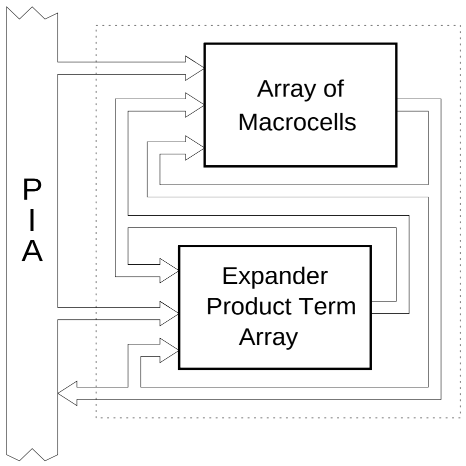


Figure 2.10: Altera Logic Array Blocks (Figure From [Brow92])

is heterogeneous.

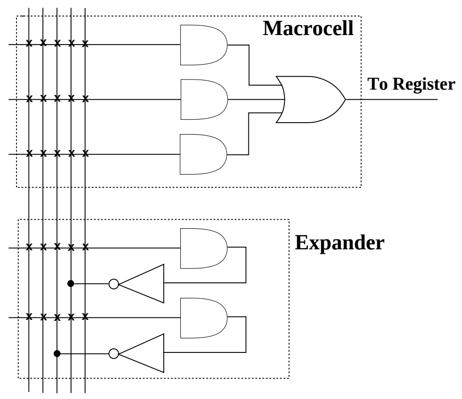


Figure 2.11: An Illustration of the Altera Macrocells and Expanders

2.4.4 Actel's Act2 FPGAs

Actel FPGAs are row-based devices. Its logic blocks, called Logic Modules (LM), and horizontal routing channels are alternated in rows, as depicted in Figure 2.12.

There are two generations of Actel FPGAs, Act-1 and Act-2. The Act-1 device uses only one general purpose module (and hence is homogeneous) which can implement all combinational functions of 2 inputs, many of 3 or 4 inputs, and others ranging up to 8 inputs. A sequential macro can be configured from one or more modules by using appropriate feedback routing.

The Act-2 FPGA, an enhanced version of the Act-1, contains two kinds of blocks, the C (Combinational) module which is used to implement combinational logic and the S (Sequential) module which is optimized to realize sequential elements.

Actel claimed that by using heterogeneous scheme, the number of modules required for a block of logic can be reduced by up to a factor of 3 [Ahre90]. On average, logic density per module is increased by over 50%. Furthermore, because the density is increased, the number of routed nets in a typical critical path is reduced. This heterogeneous architecture significantly improves

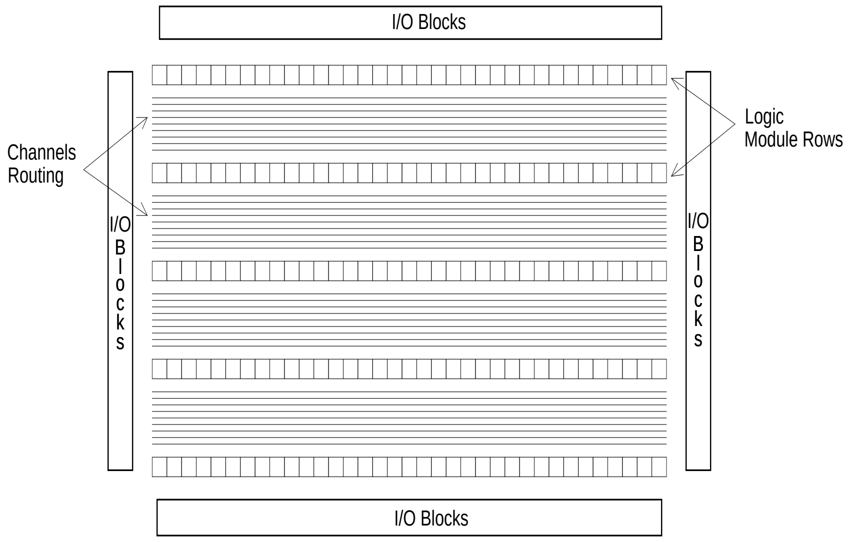


Figure 2.12: General Architecture of Actel FPGAs (Figure From [Brow92])

speed. Note that the functionalities of the combinational parts of the C and S modules are almost identical, and so this is not the kind of heterogeneity discussed in this thesis.

2.5 Summary

In this Chapter, we first introduced LUT and LUT-based FPGA technology mapping. A technology mapper for LUT was then described. Two previous initiatives of heterogeneous architecture were discussed for NAND and LUT-based FPGAs. A method for technology mapping of heterogeneous FPGAs based on a post-process of a homogeneous mapper was described. This method suffers from serious deterioration when the resulting number of heterogeneous blocks are heavily unbalanced.

Commercial FPGA architectures with heterogeneity were also outlined in this chapter.

In the next chapter, an algorithm designed specifically for heterogeneous FPGA technology mapping will be described and it will be shown that it overcomes the difficulties with the PPH mapping.

Chapter 3

Heterogeneous Technology Mapping

This chapter presents a technology mapping algorithm for heterogenous FPGAs with two different sizes of lookup table (LUT) logic blocks. Synthesis for such FPGAs is more difficult than for homogeneous FPGAs. For example, if there are two kinds of LUT which are present in equal numbers then the mapper must use the LUTs in equal proportion. This leads to a cost function that is not a linear count of the number of LUTs used and hence is difficult to optimize.

In this chapter a general method to solve problems of this nature is presented. This encompasses three steps. First, partition the input boolean network, in the form of a Directed Acyclic Graph (DAG), into a forest of fanout-free trees, then map each tree, and finally conduct an optimization amongst these trees. The cross-tree optimization algorithm will be shown to be optimal.

This chapter is organized as follows: the next section describes the basic notation and defines the technology mapping problem. Section 2 gives an overall strategy to solve the heterogeneous mapping problem. Section 3 describes the heterogeneous technology mapping algorithm on a single tree, while Section 4 provides the multi-tree algorithm and a proof of its optimality. The last section compares the quality of the proposed algorithm with a post-process of a homogeneous mapper.

3.1 Notation and Problem Definition

In this work we will consider heterogeneous FPGAs with only *two* sizes of lookup table. The larger lookup table will be referred to as the p-LUT, and the smaller as the s-LUT ($p > s$). An important architectural parameter of a heterogenous FPGA is the ratio r which is defined as the number of s-LUTs to the number of p-LUTs in the FPGA. This ratio is fixed for a given FPGA, because FPGAs are pre-fabricated. For simplicity, r is assumed to be either an integer, when $r \geq 1$, or the reciprocal of an integer, when $r < 1$. Thus if $r \geq 1$ then there are r s-LUTs for each p-LUT, and if $r < 1$ there are $\frac{1}{r}$ p-LUTs for every s-LUT. Therefore, the tuple $H = (p, s, r)$ defines a heterogeneous FPGA block architecture.

The heterogenous technology mapping problem can be stated as follows: given a boolean network, G, produce a mapped circuit M, which is a network of p-LUTs and s-LUTs of equivalent functionality to G. We are interested in minimizing the size of the FPGA needed to implement the boolean network. Since r is fixed for heterogenous FPGAs, the basic unit of *size* of a heterogenous FPGA is r s-LUTs and one p-LUT for $r \geq 1$ ($r \geq 1$ will usually be assumed for the sake of brevity, but similar definitions apply when $r < 1$). Hence we wish to minimize the number of these units, which we call a *supertile*.

Figure 3.1(a) gives an example of supertile with $H = (3, 2, 2)$. Figure 3.1(b) illustrates an array of such supertiles.

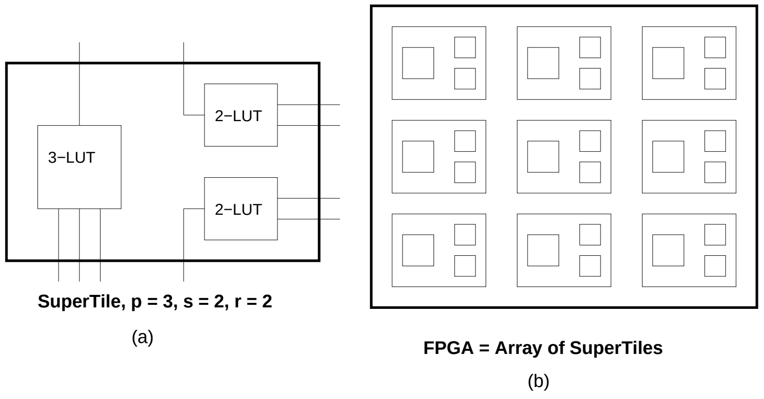


Figure 3.1: Example Supertile and Heterogenous FPGA

If we designate the number of p-LUTs in the mapped network M to be N_p, and the number of s-LUTs to be N_s, then the number of supertiles, N_{sup}, is given by:

$$N_{sup} = \mathbf{max}(N_p, \lceil \tfrac{N_s}{r} \rceil) \qquad (r \geq 1)$$

The max function makes it a non-linear cost function and hence it is difficult to minimize. For example, if $r = 1$, then for *every* p-LUT that is used in M, an s-LUT *must* be used by the mapper, or else it is wasted. This is different from standard technology mapping into an ASIC library, in which a mapper is free to choose *any* number of each kind of library element.

It is important to note that Figure 3.1 displays only the *abstraction* of a supertile, and is not meant to speak to the actual positioning or interconnection of the lookup tables. While this is an important issue, our purpose in this work is to explore the benefits of heterogenous architectures at the logic level. Should it prove successful, this will motivate subsequent work on the actual physical design of such an FPGA.

3.2 General Approach and Overall Flow

To solve the non-linear optimization problem, the general approach used here is to break it up into a set of linear optimization problems, each of which is more tractable. The essence of the approach is that we map the network several times, with different constraints each time. In the first mapping the number of p-LUTs (N_p) in the circuit is constrained to be zero and the number of s-LUTs (N_s) is minimized. In the second mapping, N_p is constrained to be exactly one, and N_s is again minimized. This process continues with the fixed value of N_p increasing by one until the value of N_s achieved reaches zero. This process results in several mappings. Given the value of r, one of these mappings will result in the minimum number of supertiles, as defined above, and the number of supertiles can be easily determined by calculating N_{sup} for each mapping.

The overall flow of the algorithm is as follows: as in [Keut87], it begins by breaking the boolean network into a forest of maximal fanout-free trees, and thereafter each tree is mapped separately. Each tree is mapped several times as described above, resulting in multiple implementations for each tree. This is followed by an algorithm which optimally selects the set of mappings, one for each tree, that minimizes the number of supertiles in the entire circuit.

The mapping flow is depicted in Figure 3.2.

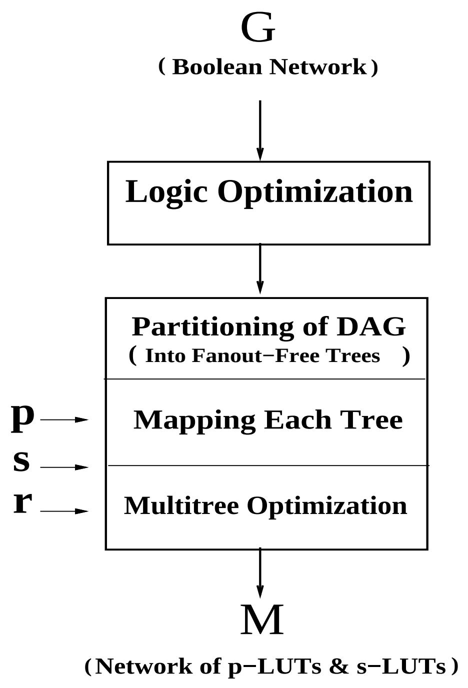


Figure 3.2: Overall Flow of the Heterogeneous FPGA Synthesis

3.3 Mapping a Single Tree

The principal tree mapping technique used in the algorithm is a generalized version of dynamic programming [Corn87]. As in dynamic programming for technology mapping, the combinational network is traversed from the inputs of the tree and proceeds to the root. Every node is mapped based on the mapped fanin information. The ordering of traversal guarantees that the fanin circuits have already been constructed.

3.3.1 Mapping a Node

At each node, a *list* of best circuits is constructed, each of which has a different number of p-LUTs. That is, each node is implemented several times, for $N_p =$ 0, 1, 2 and so on, while the number of s-LUTs, N_s, is minimized. The circuit list terminates when $N_s = 0$. Each circuit implements the cone extending from the node to the inputs of the tree.

If the node is a leaf, an s-LUT is used since it is smaller than a p-LUT and can always be changed into a p-LUT later if that is beneficial. For a non-leaf node, a set of best circuits are constructed from the list of circuits that have already been constructed on its fanin edges. Figure 3.3 gives the pseudo-code to illustrate the mapping of a single tree.

```
MapTree(tree,p,s)
 { Traverse tree from leaves to root, at each $node$:
   { if $node$ is a leaf
          BestList[$node$] $\longleftarrow$ $s$-LUT
     else BestList[$node$] $\longleftarrow$ MapNode($node$)
      // BestList contains one circuit for each $N_p$
   }
    return(BestList[root])
 }
```

Figure 3.3: Pseudo-code of Dynamic Programming for Mapping a Tree

Figure 3.4 gives a pseudo-code description for mapping a single non-leaf node. The input is a list of best circuits (one circuit for each value of N_p) for each fanin edge. The output is a similar list describing the best circuits (with the fewest s-LUTs) for each value of N_p.

Notice that many different combinations of the fanin circuits will lead to a node circuit that has a fixed value of N_p. For example, suppose there are two fanin edges, a and b, to a node and each of the edges has a list of two circuits, $\{C_0{}^a, C_1{}^a\}$ and $\{C_0{}^b, C_1{}^b\}$, where subscript in each circuit represents its number of p-LUTs. There are three combinations of these fanin circuits that may lead to a node circuit with $N_p = 1$: $C_1{}^a$ & $C_0{}^b$, or $C_0{}^a$ & $C_1{}^b$, or $C_0{}^a$ & $C_0{}^b$. In the last case the p-LUT would be created in the mapping of the node itself; in the two former cases the p-LUTs are inherited from the fanins and no p-LUTs are created in these two cases. Note that once a p-LUT is created it cannot turn back into an s-LUT, because $p > s$. However, an s-LUT can later become a p-LUT.

Since it is not known which of the fanin circuits will result in the very best value of N_s, every possible combination of the input circuit lists is evaluated, and the best is selected.

While this could result in a large number of combinations, experience on a range of benchmark circuits indicates that the number is tractable. In the worst case the total number of combinations did not exceed 414,724 and only 13 cases fall in the range $10^4 - 10^6$ for 40 MCNC benchmark circuits. Over 98.7% of the nodes required fewer than 50 combinations of input fanin circuits. This happens likely because we operate on fanout-free trees, which are typically small.

The outer loop in procedure MapNode is to enumerate each such combination. In each inner loop iteration in procedure MapNode, the desired number of p-LUTs is fixed. The lower limit on this value is the sum of the number of p-LUTs in the immediate fanin circuits. The loop runs until the number of s-LUTs is reduced to zero.

Inside the inner loop, the following problem is solved: given a fixed number of p-LUTs to create, N_p_create, and a fixed set of mapped fanin circuits, map the current node using exactly N_p_create p-LUTs and the minimum number of s-LUTs.

At this point the algorithm uses a bin-packing strategy similar to [Fran91a]. The problem to be solved is more difficult, however, because there are two kinds of LUTs to pack the logic into. The following sections describe the *packing* of the fanin circuits into two different sizes of bins, and the final construction of the tree at the current node.

3.3.2 Packing Fanin Lists into Heterogeneous LUTs

Francis' Chortle algorithm [Fran91a] makes use of a bin-packing algorithm to pack the root LUTs of the fanin circuits and the current node into an optimal tree circuit with the best possible decomposition of the current node. This is based on the observation that only the number of used inputs in the LUTs is important in determining if logic will fit into a LUT. In [Fran91a] the fanin

```
MapNode($node$)
 { BestList[$node$] $\leftarrow$ $empty$
   For each combination of fanin circuits to $node$ {
     $N_p\_in$ $\leftarrow$ total #$p$-LUTs of immediate fanins
     $N_p\_create$ $\leftarrow$ $N_p\_in$
     While ($N_s \neq 0$) {
        // pack into $N_p\_create$ $p$-LUTs and minimum #$s$-LUTs
        Packing $\leftarrow$ BinPack($node$,$N_p\_create$)

        // make packed LUTs into a tree
        Tree $\leftarrow$ TreeForm(Packing)

        if Tree best so far with value of $N_p$, then
          record it in BestList[$node$]

        $N_p\_create$ $\leftarrow$ $N_p\_create$ + 1
      }
    }
 }
```

Figure 3.4: Pseudo-code for Mapping a Node

root LUTs correspond to "boxes" to be packed, and the resulting LUTs are the receiving "bins", of size K. We apply the same approach, except that the problem is more difficult because there are now two sizes of bin, p and s.

An illustration of heterogeneous bin packing is given in Figure 3.7(a) and 3.7(b), for $N_p\_create = 2$, where $p = 5$ and $s = 4$.

The heterogeneous bin packing problem can be stated as follows: given a number of p-LUTs to create (N_p_create), and the fanin circuits' root LUTs, pack the fanin root LUTs into exactly N_p_create p-LUTs and a minimum number of additional s-LUTs.

We apply a variation of the First-Fit Decreasing algorithm: first create the number of p-LUTs that already exist in the fanin root LUTs. Then sort the remaining fanin root LUTs ("boxes") into decreasing order and put them into the first LUTs fitted. If a new "bin" is needed, create a p-LUT if N_p_*create* p-LUTs have not yet been created, and otherwise create an s-LUT. Figure 3.5 gives the pseudo-code outline of this packing algorithm. Although not shown in Figure 3.5, we also apply the re-convergent fanout optimization by using the Maximum Share Decreasing algorithm described in [Fran92a] and [Brow92]. However, replication of logic is not exploited due to the complexity of such a procedure.

3.3.3 Forming a Tree

After the packing of the fanin root LUTs is completed, these packed LUTs are connected to form a tree to realize the current node and its fanins. The LUTs are sorted by decreasing order of number of used inputs and the output of the largest is connected to any unused inputs in the subsequent bins. The purpose of this procedure is to make the root node have as many unused inputs as possible. This is beneficial because the unused inputs can be utilized by subsequent nodes, as described in [Fran91a].

If there are insufficient inputs to connect all the LUTs together, then new s-LUTs are created. Figure 3.6 gives the pseudo code of the tree forming procedure and Figure 3.7(c) illustrates the tree forming procedure for the packed circuit of Figure 3.7(b).

3.4 Multi-tree Optimization

3.4.1 The Multi-tree Optimization Procedure

After each tree T_i (i=1, 2,$\cdots$, m) has been mapped, the algorithm has produced a set of circuits $\{ C_j{}^i \}$ where i is the tree number, and j is the number of p-LUTs in the mapped solution for that tree. For each circuit, $C_j{}^i$, let S_j^i be its number of s-LUTs. Table 3.1 gives an example of several typical values of S_j^i for $p = 5$ and $s = 4$.

Recall that the optimization goal is to find the minimum number of supertiles given by: $N_{sup} = \mathbf{max}(N_p, \lceil \frac{N_s}{r} \rceil)$, where for Table 3.1, $N_p = j$ and $N_s = S_j^i$.

For each tree, N_{sup} can be easily calculated simply from N_p and N_s in each mapped solution, and the best selected. For example, from Table 3.1,

j $(= N_p)$	0	1	2	3	4	5	6	7
S_j^i $(= N_s)$	11	9	7	6	4	2	1	0

Table 3.1: Example mapping counts of p-LUTs and s-LUTs for one tree

if $r = 1$, then the solution with $N_p = 4, N_s = 4$ (4 supertiles) table entry is minimal and the corresponding circuit should be selected. If $r = 2$, then $N_p = 3, N_s = 6$ (3 supertiles) is minimal.

While this is simple for a single tree, the problem becomes more difficult when optimizing the number of supertiles across a number of trees. It may be that one tree is efficiently implemented using mostly s-LUTs and a second tree is better with mostly p-LUTs. If the number of supertiles in the trees were optimized individually, as above, this advantage may never be realized, since such a procedure seeks to balance the s-LUTs and p-LUTs according to ratio r on an individual tree basis.

A naive algorithm, however, that evaluates all possible combinations of table entries across all trees has enormous complexity. If the number of table entries per tree is n, and there are m trees, then the number of different combinations of tree solutions is n^m. In the following we will present an algorithm to solve this problem optimally with complexity O$((m \times n)^2)$.

We will first illustrate the basic algorithm on the combination of two trees, with family of solutions $\{{C_j}^1\}$ and $\{{C_j}^2\}$. Let the number of s-LUTs be ${S_0}^1$, ${S_1}^1$, ${S_2}^1$, ... ${S_{n_1}}^1$ for Tree 1, and ${S_0}^2$, ${S_1}^2$, ${S_2}^2$, ... ${S_{n_2}}^2$ for Tree 2, where n_1 and n_2 are integers. When the two trees are taken together, we need to determine ${MS_n}^{1\cup2}$, the smallest number of s-LUTs for a fixed number of p-LUTs, n, where n = 0, 1, 2, ... $n_1 + n_2$ and ${MS_n}^{1\cup2} = \min({S_n}^{1\cup2})$. ${MS_0}^{1\cup2}$ can be found by summing ${S_0}^1$ and ${S_0}^2$. The value of ${MS_1}^{1\cup2}$ is given by $\min({S_0}^1 + {S_1}^2, {S_1}^1 + {S_0}^2)$. Similarly, for higher values of n, ${MS_k}^{1\cup2}$ can be determined by finding the minimum sum of all possible pairs of ${S_x}^1$ and ${S_y}^2$, for which $x + y = n$.

Table 3.2 gives an example of this calculation for two small trees, in which $n_1 = 3$ and $n_2 = 2$. The rows labelled T_1 and T_2 provide the values of $\{{S_j}^1\}$ and $\{{S_j}^2\}$. The next three rows show the possible combinations that provide the corresponding entries of ${S_j}^{1\cup2}$. The final row gives the best of these combinations (that with the smallest number of s-LUTs). Note that if there

	S_0	S_1	S_2	S_3	S_4	S_5
T_1	5 $(C^1{}_0)$	3 $(C^1{}_1)$	2 $(C^1{}_2)$	0 $(C^1{}_3)$		
T_2	4 $(C^2{}_0)$	2 $(C^2{}_1)$	0 $(C^2{}_2)$			
T_1 combined with T_2	5 $(C^1{}_0)$	3 $(C^1{}_1)$	2 $(C^1{}_2)$	0 $(C^1{}_3)$		
	4 $(C^2{}_0)$	4 $(C^2{}_0)$	4 $(C^2{}_0)$	4 $(C^2{}_0)$		
		5 $(C^1{}_0)$	3 $(C^1{}_1)$	2 $(C^1{}_2)$	0 $(C^1{}_3)$	
		2 $(C^2{}_1)$	2 $(C^2{}_1)$	2 $(C^2{}_1)$	2 $(C^2{}_1)$	
			5 $(C^1{}_0)$	3 $(C^1{}_1)$	2 $(C^1{}_2)$	0 $(C^1{}_3)$
			0 $(C^2{}_2)$	0 $(C^2{}_2)$	0 $(C^2{}_2)$	0 $(C^2{}_2)$
$MS^{1\cup 2}$	9	7	5	3	2	0
Circuit	$(C^1{}_0C^2{}_0)$	$(C^1{}_1C^2{}_0)$	$(C^1{}_1C^2{}_1)$	$(C^1{}_1C^2{}_2)$	$(C^1{}_3C^2{}_1)$	$(C^1{}_3C^2{}_2)$

Table 3.2: Example of Tree Combination

is a tie, the circuit first encountered is chosen.

This algorithm can be extended to multiple trees by combining, in turn, a subsequent tree with the results of the previous trees. This forms, in turn, $MS^{1\cup 2\cup 3}$, $MS^{1\cup 2\cup 3\cup 4}$, ... , and $MS^{1\cup 2\cup \ldots \cup m}$. Using this final table, the optimal number of supertiles can be determined for a given ratio, r, by applying the above equation for N_{sup}, and choosing the entry with the minimal numbers. This choice implies a set of choices of a specific $C_j{}^i$ from each tree, and hence a complete mapping. This algorithm is optimal with respect to the number of supertiles, as shown below.

3.4.2 Proof of Optimality of The Multi-tree Optimization

In this section, we will prove that the multi-tree combination algorithm is optimal. Recall that the input to this step is m mapped trees $T_1, T_2, \cdots, T_m$. Let C^i_j be the mapped circuit in T_i with j p-LUTs and S^i_j be the number of s-LUTs in C^i_j. Then, the multi-tree optimization problem can be stated as follows:

Given: m trees:

$$T_1: \quad S^1_0, S^1_1, S^1_2, \cdots, S^1_{n_1}$$

$T_2: \quad S_0^2, S_1^2, S_2^2, \cdots, S_{n_1}^2, \cdots, S_{n_2}^2$

. . . .

$T_m: \quad S_0^m, S_1^m, S_2^m, \cdots, S_{n_m}^m$

where $n_1, n_2, \cdots, n_m$ are integers and S_j^i is the number of s-LUTs in the solution for tree i constrained to have j $p-$LUTs.

Find: A final table $MS^{*\ 1\cup 2\cup\cdots\cup m}$ which has minimal number of $s-$LUTs for a given number of $p-$LUTs for the entire circuit, i.e., find:

$$MS^{*\ 1\cup 2\cup\cdots\cup m} = MS_0^{*\ 1\cup 2\cup\cdots\cup m}, MS_1^{*\ 1\cup 2\cup\cdots\cup m}, \cdots, MS_{k_m}^{*\ 1\cup 2\cup\cdots\cup m}.$$

$$\textit{where} \qquad k_m = \sum_{t=1}^{m} n_t,$$

$$MS_{j_m}^{*\ 1\cup 2\cup\cdots\cup m} = \min_{(\forall i_t s.t. \sum_{t=1}^{m} i_t) = j_m} (\sum_{t=1}^{m} S_{i_t}^t)$$

$$\textit{where}\ \ 0 \leq i_t \leq n_t, j_m = 0, 1, \cdots, k_m.$$

The **multi-tree optimization algorithm** described in the previous section can

be re-written as the following calculation steps:

$$\begin{aligned}
1.\ MS^{1\cup 2} &= MS_0^{1\cup 2}, MS_1^{1\cup 2}, \cdots, MS_{k_2}^{1\cup 2} \\
\textit{where} \quad & k_2 = n_1 + n_2, \\
& MS_{j_2}^{1\cup 2} = \min_{\forall i_1, i_2, s.t. i_1 + i_2 = j_2} (S_{i_1}^1 + S_{i_2}^2) \\
& where\ i_1 = 0, 1, \cdots, n_1, i_2 = 0, 1, \cdots, n_2, and \\
& \qquad j_2 = 0, 1, \cdots, k_2. \\
2.\ MS^{1\cup 2\cup 3} &= MS_0^{1\cup 2\cup 3}, MS_1^{1\cup 2\cup 3}, \cdots, MS_{k_3}^{1\cup 2\cup 3} \\
\textit{where} \quad & k_3 = n_1 + n_2 + n_3, \\
& MS_{j_3}^{1\cup 2\cup 3} = \min_{\forall j_2, i_3, s.t. j_2 + i_3 = j_3} (MS_{j_2}^{1\cup 2} + S_{i_3}^3), \\
& where\ j_2 = 0, 1, \cdots, k_2, i_3 = 0, 1, \cdots, n_3, and \\
& \qquad j_3 = 0, 1, \cdots, k_3. \\
& \vdots \\
m.\ MS^{1\cup 2\cup\cdots\cup m} &= MS_0^{1\cup 2\cup\cdots\cup m}, MS_1^{1\cup 2\cup\cdots\cup m}, \cdots, MS_{k_m}^{1\cup 2\cup\cdots\cup m} \\
\textit{where} \quad & k_m = \sum_{t=1}^{m} n_t, \\
& MS_{j_m}^{1\cup 2\cup\cdots\cup m} = \min_{\forall j_{m-1}, i_m, s.t. j_{m-1} + i_m = j_m} (MS_{j_{m-1}}^{1\cup 2\cup\cdots\cup(m-1)} + S_{i_m}^m), \\
& where\ j_{m-1} = 0, 1, \cdots, k_{m-1}, i_m = 0, 1, \cdots, n_m, and
\end{aligned}$$

$$j_m = 0, 1, \cdots, k_m.$$

where $MS^{1\cup 2 \cdots \cup m}$ represents the solution obtained in the multi-tree optimization procedure.

Theorem: The multi-tree optimization algorithm described above is optimal. The solution, $MS^{1\cup 2 \cdots \cup m}$, obtained in the multi-tree optimization procedure, has the minimal number of $s-$LUTs for a given number of $p-$LUTs for the entire circuit. That is, $MS^{1\cup 2 \cdots \cup m} = MS^{*\ 1\cup 2 \cdots \cup m}$.

Proof:

We prove the *optimality* of the above algorithm by *induction.*

$\boldsymbol{m = 2}$ (the base case): $m = 2$ is trivial since the calculation of $MS^{1\cup 2}$ is exhaustive. Therefore, $MS^{1\cup 2} = MS^{*\ 1\cup 2}$.

To make the induction step simple and more intuitive we first show how $m = 2$ optimality implies $m = 3$ optimality, by contradiction.

$\boldsymbol{m = 3}$:

Assume that when m = 2, $MS^{1\cup 2} = MS^{*\ 1\cup 2}$.

Suppose, for contradiction, that for some j_3, $MS_{j_3}^{1\cup 2\cup 3}$ (where $0 \leq j_3 \leq k_3$) is not the optimum (minimum); the optimum value is $MS_{j_3}^{*\ 1\cup 2\cup 3}$, which implies $MS_{j_3}^{1\cup 2\cup 3} > MS_{j_3}^{*\ 1\cup 2\cup 3}$.

Since $MS_{j_3}^{*\ 1\cup 2\cup 3}$ must have one of the $S_{i_3}^3$ as its component, where $i_3 = 0, 1, \cdots, n_3$, let that number be $S_{i_3^*}^3$.

Thus, $MS_{j_3^*}^{*\ 1\cup 2\cup 3} = MS_{j_2^*}^{1\cup 2} + S_{i_3^*}^3$, where $i_3^* \in \{0, 1, \cdots, n_3\}$, $j_2^* \in \{0, 1, \cdots, k_2\}$ and $k_2 = n_1 + n_2$, and $j_3^* \in \{0, 1, \cdots, k_3\}$.

Since $MS^{1\cup 2} = MS^{*\ 1\cup 2}$ and

$MS_{j_3}^{1\cup 2\cup 3} = \min_{\forall j_2, i_3, s.t. j_2 + i_3 = j_3}(MS_{j_2}^{1\cup 2} + S_{i_3}^3) = \min_{\forall j_2, i_3, s.t. j_2 + i_3 = j_3}(MS_{j_2}^{*\ 1\cup 2} + S_{i_3}^3)$,

we can show the contradiction $MS_{j_2^*}^{1\cup 2} < MS_{j_2^*}^{*\ 1\cup 2}$ from the following arguments

Since we have: $MS_{j_3}^{1\cup 2\cup 3} > MS_{j_3}^{*\ 1\cup 2\cup 3}, for some j_3 \in \{0,1,\cdots,k_3\}$

$\Rightarrow MS_{j_3^*}^{1\cup 2\cup 3} > MS_{j_3^*}^{*\ 1\cup 2\cup 3}$, for $j_3 = j_3^* \in \{0,1,\cdots,k_3\}$

$\Rightarrow MS_{j_3^*}^{1\cup 2\cup 3} > MS_{j_3^*}^{*\ 1\cup 2\cup 3} |_{\forall i_3, j_2 s.t. i_3+j_2=j_3^*}$

$\Rightarrow MS_{j_2^*}^{*\ 1\cup 2} + S_{i_3^*}^3 > MS_{j_2^*}^{1\cup 2} + S_{i_3^*}^3$

$\Rightarrow MS_{j_2^*}^{*\ 1\cup 2} > MS_{j_2^*}^{1\cup 2}$.

Contradiction! since $MS^{*\ 1\cup 2}$ is the minimum. Therefore, $MS^{1\cup 2\cup 3} = MS^{*\ 1\cup 2\cup 3}$.

Now we prove the general case in a similar way [Hutt93].
***m* = *K*-1**: assume that when $m = K$ - 1, $MS_{j_{K-1}}^{1\cup 2\cdots\cup(K-1)} = MS_{j_{K-1}}^{*\ 1\cup 2\cdots\cup(K-1)}$
,
then we will prove the optimality for $m = K$ (very similar to the proof for $m = 3$).

m* = *K: proof by contradiction.

Suppose that $MS_{j_K}^{1\cup 2\cup\cdots\cup K} \neq MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K}$ which implies that

$MS_{j_K}^{1\cup 2\cup\cdots\cup K} > MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K}$.

Let $S_{i_K^*}^K$ be the component for $MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K}$ and

$MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K} = MS_{j_{K-1}^*}^{1\cup 2\cup\cdots\cup(K-1)} + S_{i_K^*}^K$, where

$k_{K-1} = \sum_{t=1}^{K-1} n_t, k_K = \sum_{t=1}^{K} n_t, i_K^* \in \{0,1,\cdots,n_K\}, j_{K-1}^* \in \{0,1,\cdots,k_{K-1}\}$, and
$j_K^* \in \{0,1,\cdots,k_K\}$.

Then we have $MS_{j_{K-1}^*}^{*\ 1\cup 2\cup\cdots\cup(K-1)} > MS_{j_{K-1}^*}^{1\cup 2\cup\cdots\cup(K-1)}$ since

$\forall j_K \in \{0, 1, \cdots, k_K\}, MS_{j_K}^{1\cup 2\cup\cdots\cup K} > MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K}$

$\Rightarrow MS_{j_K}^{1\cup 2\cup\cdots\cup K} > MS_{j_K}^{*\ 1\cup 2\cup\cdots\cup K}\ |_{j_K=j_K^*}$

$\Rightarrow MS_{j_K^*}^{1\cup 2\cup\cdots\cup K} > MS_{j_K^*}^{*\ 1\cup 2\cup\cdots\cup K}$

$\Rightarrow MS_{j_K^*}^{1\cup 2\cup\cdots\cup K} > MS_{j_K^*}^{*\ 1\cup 2\cup\cdots\cup K}\ |_{\forall i_K, j_{K-1} s.t. i_K + j_{K-1} = j_K^*}$

$\Rightarrow MS_{j_{K-1}}^{*\ 1\cup 2\cup\cdots\cup(K-1)} + S_{i_K}^K > MS_{j_{K-1}^*}^{1\cup 2\cup\cdots\cup(K-1)} + S_{i_K^*}^K\ |_{i_K=i_K^*, j_{K-1}=j_{K-1}^*}$

$\Rightarrow MS_{j_{K-1}^*}^{*\ 1\cup 2\cup\cdots\cup(K-1)} + S_{i_K^*}^K > MS_{j_{K-1}^*}^{1\cup 2\cup\cdots\cup(K-1)} + S_{i_K^*}^K$

$\Rightarrow MS_{j_{K-1}^*}^{*\ 1\cup 2\cup\cdots\cup(K-1)} > MS_{j_{K-1}^*}^{1\cup 2\cup\cdots\cup(K-1)}$

Contradiction! since $MS^{*\ 1\cup 2\cup\cdots\cup(K-1)}$ is optimal for $m = K$ - 1. Therefore, $MS^{1\cup 2\cdots\cup m} = MS^{*\ 1\cup 2\cdots\cup m}$.

3.4.3 Complexity of the Multi-tree Optimization Algorithm

Let N be the complexity of the multi-tree optimization algorithm, N_i be the complexity of each step in the calculation of S'_c in section 3.4.2, and $n = \max_{i=1,2,\cdots,m}(n_i)$. Then

$$\begin{aligned} N_1 &= (n_1+1)\times(n_2+1) \\ N_2 &= (n_1+n_2+1)\times(n_3+1) \\ N_3 &= (n_1+n_2+n_3+1)\times(n_4+1) \\ \cdots &= \cdots\cdots \\ N_{m-1} &= (\sum_{t=1}^{m-1} n_t + 1)\times(n_m+1) \end{aligned}$$

and

$$N = \sum_{i=1}^{m-1} N_i$$

$$\begin{aligned}
&= \sum_{i=1}^{m-1}(\sum_{t=1}^{i} n_t + 1)(n_{i+1} + 1) \\
&\leq \sum_{i=1}^{m-1}(\sum_{t=1}^{i} n + 1)(n + 1) \\
&= (n+1)\sum_{i=1}^{m-1}(i \times n + 1) \\
&= (n+1) \times (n\sum_{i=1}^{m-1} i + m - 1) \\
&= (n+1) \times (n \times \frac{m \times (m-1)}{2} + m - 1) \\
&= (n+1) \times (m-1) \times (\frac{(m \times n + 2)}{2}) \\
&= \mathbf{O}(m^2 n^2)
\end{aligned}$$

3.5 Comparison with PPH Mapping Algorithm

This section gives a comparison between the quality of the heterogenous mapping algorithm with the best alternative approach that could be found: a post-process of output from a homogenous LUT mapper, which takes the non-homogeneity into account.

As mentioned in Chapter 2, the post-process-homo (PPH) was developed in [Mizu92]. To generate a heterogeneous FPGA, this method first used a homogeneous mapper (Chortle) [Fran91a] to map a given network into p-LUTs and then convert those LUTs whose number of used inputs are less than or equal to s into s-LUTs.

Following this step the number of supertiles used in this mapping was calculated using $N_{sup} = \mathbf{max}(N_p, \lceil\frac{N_s}{r}\rceil)$ where N_p is the number of p-LUTs, N_s the number of s-LUTs, and r the ratio of number of s-LUTs to the number of p-LUTs. In the case that there is an imbalance between N_p and $\lceil\frac{N_s}{r}\rceil$, a transformation of p-LUTs and s-LUTs is performed in the following way. If $N_p < \lceil\frac{N_s}{r}\rceil$, then p-LUTs will be transformed to s-LUTs on a one-to-one basis. When $N_p > \lceil\frac{N_s}{r}\rceil$, several s-LUTs will be used to make a p-LUT, as described in section 2.3.2 of Chapter 2. This algorithm assumed that every p-LUT had all its inputs used. Hence to make a p-LUT from a network of s-LUTs requires many s-LUTs, as shown in Table 2.1. In this section we compare the heterogeneous mapper with an improved version of this algorithm which is described below [Fran92b]:

After the homogeneous mapping step, the result is a set of LUTs. The number of inputs actually used on each LUT will range from 1 to p, where p is the size of a homogeneous LUT. Let $D = \{D_1, D_2, \cdots, D_u, \cdots, D_p\}$ describe the distribution of this number, where D_u is the number of p-LUTs having u used inputs. As before the number of supertiles is calculated as $N_{sup} = \mathbf{max}(N_p, \lceil \frac{N_s}{r} \rceil)$, where $N_s = \sum_{u=1}^{s} D_u$ and $N_p = \sum_{u=s+1}^{p} D_u$. If $N_p > \lceil \frac{N_s}{r} \rceil$, instead of transforming a network of s-LUTs into p-LUT, we transform s-LUTs into $(s+1)$-LUTs to achieve a balance. If there are not enough $(s+1)$-LUTs, then we transform s-LUTs into $(s+2)$-LUTs and so on. This is better than the previous algorithm because the LUTs with fewer inputs require fewer s-LUTs in the implementation.

The Chortle-crf homogeneous mapper [Fran92a] was used as the basis for comparison since both the homogeneous mapper and the heterogeneous one have a similar mapping process. Both the new heterogenous mapper and the post-process-homo mapper were run on 40 MCNC logic synthesis benchmarks circuits, and for many different values of s and p. The total number of supertiles obtained for each case was calculated, and these are illustrated in the plot in Figure 3.8. The Y-axis is the total number of supertiles for all the circuits and the X-axis gives the different combinations of s and p, with the ratio $r = 1$ for all cases. It is clear that the new algorithm is superior to the PPH algorithm. On average, for $r = 1$, the new algorithm produces 0.3% to 11.7% fewer supertiles than PPH. As will be shown below, for $r > 1$ the new algorithm provides even greater advantage over PPH.

The trends in Figure 3.8 (in particular, the large "jumps") can be explained as follows.

First, for each given p , as s increases, the number of supertiles decreases. This makes sense since for the same p increasing s will increase functionality of a supertile and this will decrease the number of supertiles needed.

Secondly, for each increase in p, there is a jump in the supertile count. For the heterogeneous FPGAs, this is likely caused by the decrease in functionality of a supertile. For example, supertile $(p, s, r) = (6,\ 2,\ 1)$ has less functionality than (5, 4, 1) and a (7, 2, 1) is less than a (6, 5, 1).

For the PPH algorithm, however, there is another important reason to cause these increases besides the one just mentioned. Recall that in the post-processing of homogeneous FPGAs, there is a transformation between the p-LUTs and the s-LUTs. In the combinations where a big increase occurs, p and s differ significantly. In such cases, when a p-LUT is converted to an s-LUT or s-LUTs to a p-LUT, there can be a significant waste. For example, for $p = 6$ and $s = 2$. If we transform a 6-LUT into a 2-LUT (i.e. to use a 6-LUT to implement a 2-LUT), there will be a large waste of functionality. On the other hand, however, if we implement a 6-LUT using 2-LUTs, a total of 61 2-LUTs are needed (see Table 2.1). The cost is even more since one 6-LUT uses 7 pins and 64 bits but 61 2-LUTs require 183 pins and 244 bits. Therefore, the increase is bigger for PPH than for heterogeneous FPGA at each change of p values.

Figure 3.9 gives another view of the comparison of the heterogeneous and the PPH mappings. This figure presents the improvement percentage of the number of supertiles versus ratio r for combination $(p, s) = (5, 2)$ which is typical of all the (p, s) combinations. The improvement is calculated using equation 3.1.

$$Improve\_pct = -\frac{(N_{sup}\ of\ Het) - (N_{sup}\ of\ PPH)}{N_{sup}\ of\ PPH} \times 100 \tag{3.1}$$

where N_{sup} is the total number of supertiles needed for implementing the 40 benchmark circuits for combination $(p, s, r) = (5, 2, r)$. Observe that for larger values of r, the new algorithm achieves more than 30% fewer supertiles than PPH.

Figure 3.9 can be explained as follows. First, when r is very small (e.g. $r = \frac{1}{10}$), the difference between the heterogeneous and the PPH mappings is not very great. This is because the problem being solved is close to the homogeneous problem for $p-$LUTs. Both homogeneous and heterogeneous mappings are optimized at this point. Note that the PPH algorithm is tuned for this case, but the heterogeneous algorithm is tuned for all r. Thus as r increases, the relative improvement increases greatly. This increase is largely due to the fact that as r increases, there are more s-LUTs in each supertile in a given FPGA, while the number of s-LUTs and the number of p-LUTs after a PPH mapping are fixed for a fixed combination (p, s). This implies that we will need to transform more s-LUTs into p-LUTs for an implementation if r is large. As mentioned previously in this section, this kind of transformation is very costly.

s \ p		**7**	**6**	**5**	**4**	**3**
2	post-homo	2066	2263	2532	2954	3748
	hetero.	1892	2035	2242	2607	3366
	decrease (%)	8.4	10.1	11.5	11.7	10.2
3	post-homo	1798	1927	2140	2414	
	hetero.	1750	1868	2058	2301	
	decrease (%)	2.7	3.1	3.8	4.7	
4	post-homo	1738	1838	2017		
	hetero.	1716	1805	1985		
	decrease (%)	1.3	1.8	1.6		
5	post-homo	1708	1799			
	hetero.	1693	1793			
	decrease (%)	0.9	0.3			
6	post-homo	1702				
	hetero.	1689				
	decrease (%)	0.8				

Table 3.3: Comparison with Post-Process of Homogeneous Mapping ($r = 1$)

Data values corresponding to Figure 3.8 are presented in Table 3.3. Average improvement for each combination (also for $r = 1$) is listed on the third row of each cell of the table. Note that this average improvement is based on the total number of supertiles of the 40 benchmark circuits used, implying that the calculation for the average has unequal weights for different circuits. Larger circuits have larger weights. Table 3.4 gives the average improvement of heterogeneous mapping over the PPH algorithm on an equal weight basis for each circuit. The average improvements in Table 3.4 are calculated by computing the improvement of each circuit and then averaging these numbers. Standard deviations of the improvements are also presented in the same table. The formula for calculating the standard deviation is shown in equation 3.2 below:

$$Std\_Dev = \sqrt{\sum_{i=1}^{40} \frac{(improve\_i - improve\_avg)^2}{40}} \tag{3.2}$$

s \ p		**7**	**6**	**5**	**4**	**3**
2	improve (%)	6.7	7.7	9.3	9.8	8.1
	std. dev (%)	5.8	6.3	7.4	6.2	5.1
3	improve (%)	2.6	2.4	2.7	3.8	
	std. dev (%)	5.4	6.5	5.3	6.0	
4	improve (%)	1.3	1.6	0.8		
	std. dev (%)	4.9	5.0	3.0		
5	improve (%)	0.4	0.2			
	std. dev (%)	1.7	1.2			
6	improve (%)	0.8				
	std. dev (%)	4.0				

Table 3.4: Equal Weight Comparison with PPH Mapping and Standard Deviation

where $improve\_avg = \sum_{i=1}^{40} \frac{improve\_i}{40}$ and improve_i is the improvement on the i-th benchmark circuit.

In general, the standard deviations have the same magnitudes as the improvement averages.

Table 3.5 gives detailed sample results for the heterogenous architecture $p = 5, s = 2$, and $r = 1$. The first column of this table gives the circuit name and the second column gives the number of supertiles using the algorithm described in this paper. The third column gives the supertile count for the post-process-homo algorithm, and the fourth column gives the percentage difference between the two. The fifth column gives the running time of the new algorithm on a Sun Sparcstation 2. The running times are usually in a few seconds and rarely more than a minute.

It should be pointed out that since the heterogeneous algorithm does not replicate logic at fanout nodes, the homogeneous mapping algorithm we use is also prevented from this replication. It is possible that this may affect these comparisons.

Circuit Names	#Supertiles (Hetero Algo)	#Supertiles (Post-Homo)	Difference (%)	Run Time (in Sec.)
C1355	82	82	0.0	1.6
C432	55	59	-6.8	10.0
C880	63	72	-12.5	4.2
alu2	72	80	-10.0	93.8
alu4	124	138	-10.1	92.9
apex6	136	161	-15.5	19.9
apex7	42	48	-12.5	0.8
b9	24	24	-0.0	0.4
c8	23	25	-8.0	0.5
cht	30	35	-14.3	0.4
cm150a	9	10	-10.0	0.3
cm151a	5	5	0.0	0.1
cm85a	7	8	-12.5	0.1
cmb	10	12	-16.7	0.1
count	23	23	0.0	0.4
example2	57	64	-10.9	1.1
frg1	21	24	-12.5	18.1
frg2	150	171	-12.3	3.7
i1	10	10	0.0	0.1
i6	64	67	-4.5	3.2
i7	95	112	-15.2	9.2
i8	181	225	-19.6	12.6
i9	106	123	-13.1	9.8
k2	187	207	-9.7	11.5
my-adder	32	32	0.0	0.4
parity	8	8	0.0	0.1
pcler	16	16	0.0	0.2
pm1	9	10	-10.0	0.1
rot	124	135	-8.1	2.5
sct	14	15	-6.7	0.1
t481	6	6	0.0	0.1
term1	24	28	-14.3	1.2
ttt2	30	33	-9.1	0.5
unreg	18	29	-37.9	0.7
vda	105	114	-7.9	1.7
x1	57	63	-9.5	4.5
x2	9	10	-10.0	0.1

3.6 Summary

A technology mapping algorithm designed specifically for a heterogeneous FPGA was described in this chapter. This algorithm maps a boolean network into a heterogeneous FPGA that has two kinds of logic block in a fixed ratio.

This algorithm first partitions a given DAG into a set of fanout-free trees and then maps each tree into a list of best circuits. These circuits are functionally equivalent, but have different number of p-LUTs. The number of p-LUTs used in each circuit ranges from zero (the tree is mapped all with s-LUTs) to the number where the tree is mapped all with p-LUTs. In this way, we can find out which LUT is most suitable for implementing a portion of network and make advantageous use of the LUTs.

The circuit for the whole DAG must be found by choosing one circuit from each tree. There are a large number of possible combinations of selections of the final circuits. If there are m trees and n circuit choices in each tree, there are n^m possible choice combinations. We solved this combination problem by using a multi-tree optimization algorithm which has a complexity of only $O((m \times n)^2)$. The algorithm is proven to be optimal.

The algorithm was shown to be superior to a previous method for generating heterogeneous FPGAs from post-process of homogeneous mapping, providing 11% improvement for $r = 1$ and up to 50% for large r. It was also shown to be a reasonably fast algorithm.

```
BinPack($node$, $N_p\_create$)
 { BoxList $\leftarrow$ fanin root LUTs sorted by decreasing size
  $N_p\_in$ $\leftarrow$ number of $p$-LUTs in BoxList

  // pack $p$-LUTs into bins of size p because
  // they will not fit into $s$-LUTs, by definition
  BinList $\leftarrow$ $N_p\_in$ $p$-LUTs in BoxList
  $N_p\_created$ $\leftarrow$ $N_p\_in$   // $N_p\_created$ is #$p$-LUTS created

  while (BoxList not $empty$ )
    { box $\leftarrow$ largest LUT in BoxList

      find first bin in BinList such that
        usedInputs(bin) + usedInputs(box) $\leq$ size(bin)

      if such a bin doesn't exist create a new bin:
       { if ( $N_p\_created < N_p\_create$ )
         { create a bin of size p
           $N_p\_created$++
         }
         else create a bin of size s
       }
      pack box into bin
    }
  return (BinList)
 }
```

Figure 3.5: Pseudo-code for Bin Packing

```
TreeForm(Packing)
 { sort BinList by number of unused inputs in increasing order
   while there is more than one bin in BinList
    { src ← first bin from BinList
      find first bin in the remaining bins such that
        usedInputs(bin) + 1 ≤ size(bin)
      if such a bin doesn't exist
        create a new bin of size s
      put src output into new bin
    }
 }
```

Figure 3.6: Pseudo-code for TreeForm

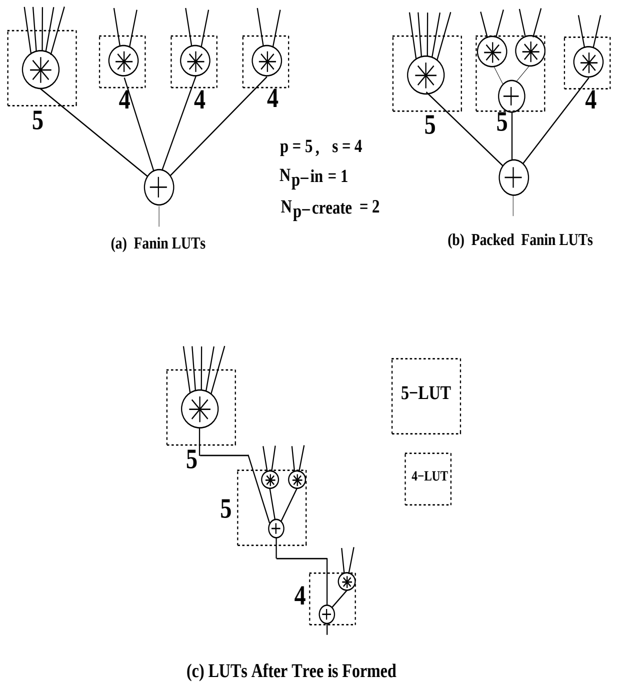


Figure 3.7: Illustrations of Bin Packing and Tree Forming

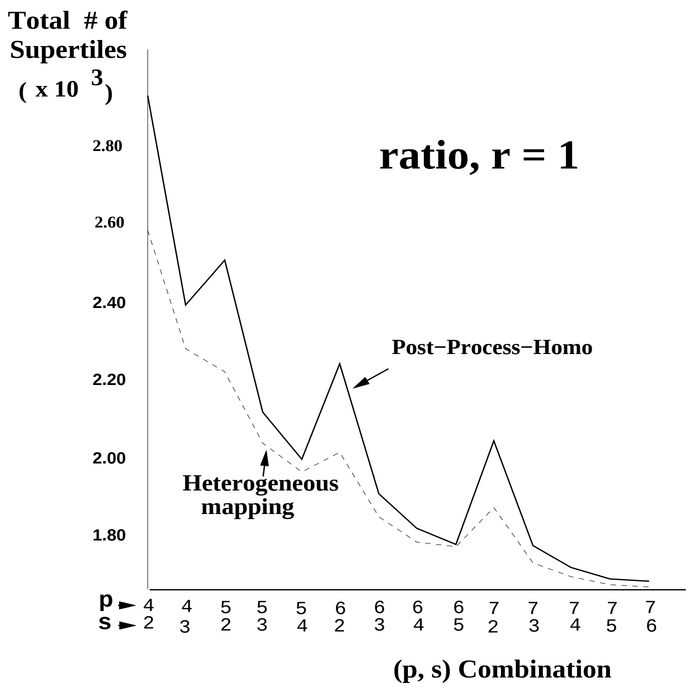


Figure 3.8: A Comparison with Post-Process of Homogeneous Mapping

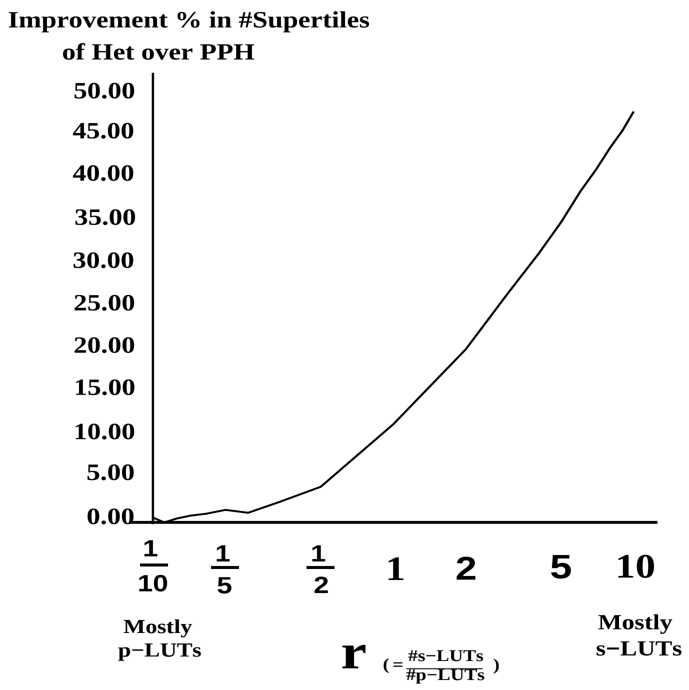


Figure 3.9: Improvement of Heterogeneous over PPH for (5, 2, r)

Chapter 4

Architectural Investigation of Heterogeneous FPGAs

To investigate the architectural advantages of heterogeneous FPGAs, a set of experiments using the heterogeneous mapper presented in Chapter 3 were conducted. This chapter describes the architectural questions investigated, the experimental methods used, and the results of these experiments.

4.1 Architectural Questions

As mentioned in section 3.1, a heterogeneous FPGA architecture is characterized by the tuple $H = (p, s, r)$, which describes the size of p-LUT, the size of s-LUT, and the ratio in which they are present. The first architectural question that arises is: what are good values of (p, s, r)? By "good" it is meant able to achieve higher density. The second question is to determine if a heterogeneous architecture has less area to a homogeneous architecture. In the current work, for simplicity and as the first step of investigation into heterogeneous FPGA, only optimization for area is concerned. No speed performance is considered.

The following section describes the experimental method for answering these architectural questions and Section 4.3 presents experimental results.

4.2 Experimental Procedure

The above architectural questions will be answered using an experimental approach: a set of benchmark combinational circuits will be synthesized into a set of heterogeneous FPGAs, each with different values of s, p and r. We can measure the area-efficiency of each such architecture for each circuit and so address the above questions. We first describe the synthesis procedure, and then describe the way in which the results are used to indicate area-efficiency.

The key issue in synthesizing for heterogenous FPGAs comes in the technology mapping step. As shown in Figure 4.1, combinational circuits are first optimized using technology-independent logic optimization, which produces an optimized boolean network. Then the technology mapping step maps the optimized network into a heterogeneous FPGA with given architectural parameters p, s, and r. The heterogeneous mapper, as described in Chapter 3, produces a heterogeneous FPGA netlist with minimized number of supertiles.

To determine the area of a netlist of logic blocks, one could perform the placement and global routing, and measure the amount of wiring needed, as well as estimate the size of the logic blocks. That approach was taken in previous studies on other aspects of FPGAs [Rose90] [Brow92] and has shown that simple measures, such as counting the number of lookup table *bits* and logic block *pins* used in the design, lead to the same architectural conclusions. So, rather than going through full placement and routing, we calculate the number of pins and the number of bits to "measure" the area of a netlist in the following way:

First, calculate the number of supertiles. If the number of p-LUTs in the netlist is N_p and the number of s-LUTs is N_s, then the number of supertiles is given by:

$$N_{sup} = \mathbf{max}(N_p, \lceil \frac{N_s}{r} \rceil) \tag{4.1}$$

where r is the ratio of the number of s-LUTs to the number of p-LUTs in a supertile.

For a single K-LUT, the number of bits is 2^K. In a supertile the number of bits is given by

$$N_{bit} = 2^p + r \times 2^s, \quad if \quad r \geq 1$$

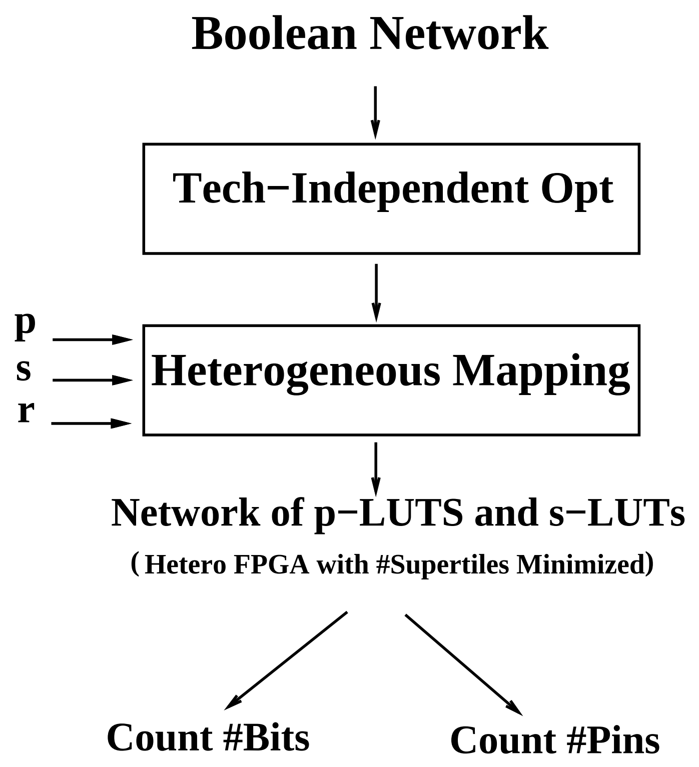


Figure 4.1: Experimental Steps for Heterogeneous FPGA Investigation

or

$$N_{bit} = 2^s + \frac{1}{r} \times 2^p, \quad if \quad 0 < r < 1 \tag{4.2}$$

If the total number of supertiles used in a circuit is N_{sup}, then the total number of bits is given by

$$TotalBits = N_{sup} \times N_{bit} \tag{4.3}$$

Routing area is very important in determining the FPGA area, because it often requires from 50% to over 90% of the total area [Rose90]. For optimized placement and routing, the total number of pins of a circuit directly relates to the total amount of required routing area. For this reason we count the total number of pins in evaluating the routing area.

Similar to the calculation of the number of bits, we can calculate the total number of pins of a circuit. For a supertile, the number of I/O pins, N_{pin}, is a function of p, s, and r and is given by

$$N_{pin} = (p+1) + r(s+1), \quad if \quad r \geq 1$$

or

$$N_{pin} = (s+1) + \frac{1}{r}(p+1), \quad if \quad 0 < r < 1 \tag{4.4}$$

where $(p+1)$ is for the p inputs and one output for a p-LUT and $(s+1)$ is for the s inputs and one output for an s-LUT.

From equations (4.1) and (4.4) we have

$$TotalPins = N_{sup} \times N_{pin} \tag{4.5}$$

4.3 Experimental Results

A total of 40 benchmark circuits from the MCNC logic synthesis benchmark suite were used as the basis for experimentation. Each was synthesized into a heterogeneous FPGA through the procedure described above. We chose p to range from 3 to 7, and s to vary from 2 to p - 1. The ratio r was varied between 0.1 and 10, and was constrained to be either an integer or the reciprocal of an integer. These selections of p, s, and r provide all the practical combinations of (p, s, r). There are a total of 285 combinations of p, s and r and so for 40 circuits the total number of FPGA circuit implementation is 11400.

Figure 4.2 plots the total number of bits versus **r** for several combinations of (p, s), where the number of bits is normalized with respect to the total number of bits of the same circuits implemented with homogenous 4-LUTs as the basic block. We use the 4-LUT as the basis for normalization as it has shown to be the most area-efficient architecture for homogeneous LUT-based FPGAs [Rose90] [Koul92].

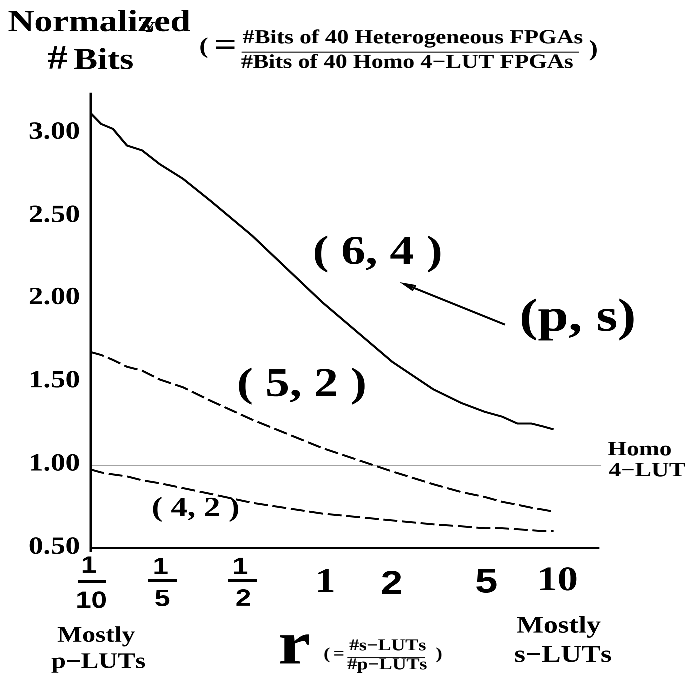


Figure 4.2: Normalized bit count vs r for different (p, s)

Observe that the total number of bits is a decreasing function of r. This is consistent with previous results for homogeneous FPGAs because as r increases, there are more s-LUTs which require significantly fewer bits to implement the same logic function of a circuit. Note that for some combinations of (p, s, r), the heterogeneous FPGAs require fewer bits than the best homogeneous FPGA. The number of LUT bits, however, is not the dominant factor in total area (unless the lookup table size is larger than about 7). The number of pins to be connected is far more important as indicated in [Rose90].

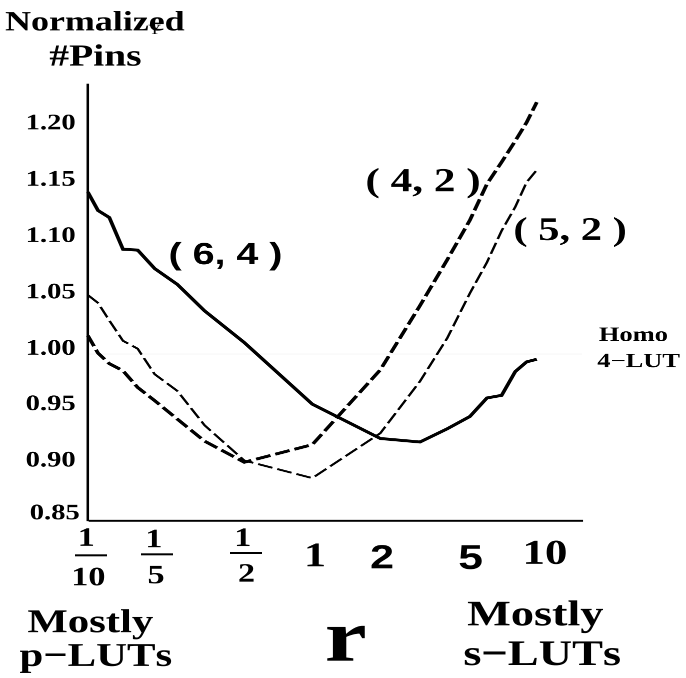


Figure 4.3: Normalized pin count vs r for the same set of (p, s) as in Fig. 4.2

Figure 4.3 illustrates the normalized total number of pins with respect to homogeneous 4-LUTs for the same set of combinations of p and s as in Figure 4.2. These curves have similar shapes for all combinations of p and s. They present several interesting results.

First, observe the shape of the curves in Figure 4.3. They exhibit (as do all the pin count curves of heterogeneous FPGAs) a “U” shape. There is a minimum between the homogeneous extremes, indicating that heterogeneous architectures are always superior to homogeneous architectures in the number of pins used. This shows that, as in the case of the example given in Chapter 1, most circuits benefit from having the choice of two different kinds of blocks.

The “U” shape curves can be explained by using the example of $p = 4$ and $s = 3$. Suppose we employ heterogeneous FPGAs in the following way: first, map the given network into a homogeneous 3-LUT FPGA. Then, introduce a single 4-LUT to the mapped 3-LUT circuit. If we can find a portion of the Boolean network such that one 4-LUT can replace two 3-LUTs (as often occurs), then we will save 3 pins with the number of bits unchanged (because two 3-LUTs have 8 pins and 1 4-LUT has 5 pins). However, as we continue to increase the number of 4-LUTs, this 1 for 2 trade will be less likely to occur. If one 4-LUT can replace only one 3-LUT, then we will use 1 more pin and double the number of bits needed. Therefore, the minimum of the number of pins is usually in between the two extremes of homogeneous 3-LUT and homogeneous 4-LUT.

Secondly, consider the combination of 4-LUT and 2-LUT. The pin curve and bit curve are redrawn together in Figure 4.4. With this combination, for most values of r, we can achieve a reduction in both the number of pins and the number of bits, compared to a homogeneous 4-LUT. At ratio $r = \frac{1}{2}$, for example, with this heterogeneous architecture, the number of bits is reduced by about 20% and the number of pins is reduced by 10%, compared to a homogeneous 4-LUT FPGA.

The above example is one case where a heterogeneous FPGA has superior area measures than a homogeneous FPGA. Table 4.1 gives the minimum value of pin count (normalized to the pin count of homogeneous 4-LUT) and its corresponding bit count (normalized to the homogeneous bit count) for all the combinations of p and s. These are the minimum pin counts over all values of r. Table 4.3 gives the actual values of r that results in the minimum. We observe from this Table that for some of the combinations, we can achieve significant pin and bit reduction. Some combinations, such

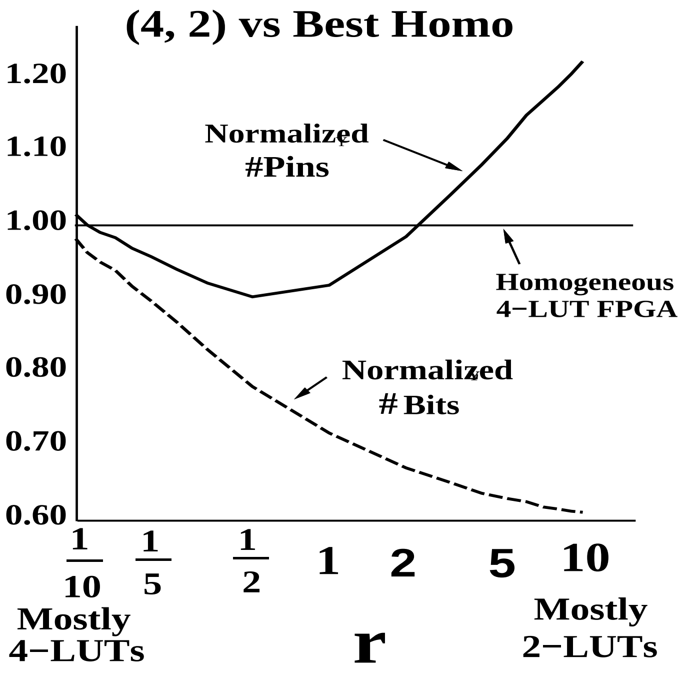


Figure 4.4: Normalized pin and bit count for combination (4, 2)

as (5, 2), has significant pin reduction with slight increase in bits (again we emphasize here that pins are much more important than bits).

These data again illustrate the conclusion that some heterogeneous architectures are more area-efficient than the best homogeneous architecture.

s \ p		**7**	**6**	**5**	**4**	**3**
2	pins	0.92	0.90	0.89	0.90	1.00
	bits	2.78	1.91	1.11	0.78	0.57
3	pins	0.90	0.89	0.91	0.91	
	bits	2.54	1.49	1.13	0.76	
4	pins	0.94	0.92	0.94		
	bits	2.25	1.47	1.12		
5	pins	1.00	1.01			
	bits	2.49	2.01			
6	pins	1.09				
	bits	3.76				

Table 4.1: Normalized minima of pin and corresponding bit, for best value of r

We believe that the heterogeneous combinations such as (4, 2), (4, 3), (5, 2) are superior in area to homogeneous 4-LUT FPGAs because many logic circuits have a significant number of small fanin functions that have fanout greater than one. These can be efficiently implemented by 2 or 3-input LUTs. Note that these results again are based on the comparison of the total number of pins (and the total number of bits) between heterogeneous and homogeneous FPGAs, indicating that different circuits have different weights.

Table 4.1 presents the average improvements on the pin count with each circuit having an equal weight. We first calculated the improvement of the heterogeneous mapping on each individual circuit over the same implementation with homogeneous 4-LUT FPGA and then averaged the improvements over all the circuits. Along with the average improvements in the Table are the standard deviations with respect to these averages. Note that a negative improvement reflects the fact that heterogeneous mapping does worse at that combination than homogeneous mapping of 4-LUT FPGAs.

s \ p		**7**	**6**	**5**	**4**	**3**
2	avg % improve	5.89	9.5	8.7	22.0	8.9
	std. dev	13.2	10.3	9.2	6.0	9.4
3	avg % improve	7.2	9.3	7.3	8.1	
	std. dev	11.9	9.8	6.9	3.5	
4	avg % improve	2.4	4.5	2.8		
	std. dev	10.9	9.6	8.2		
5	avg % improve	-6.0	-5.4			
	std. dev	14.4	12.4			
6	avg % improve	14.2				
	std. dev	19.0				

Table 4.2: Equal weight improvement averages and standard deviations for pin count

The best ratios, r, that produce the smallest value of pin count are different for different combinations of p and s. Table 4.3 gives the value or range (within 1% difference of pin count from the minimum) of r that achieved the minimal total pin count for each combination (p, s), corresponding to Table 4.2. When p and s are both small, the ratios are also small, favoring large p-LUTs since r is the ratio of the number of s-LUTs to the number of p-LUTs. If p and s are large, the best value of r increases, favoring the s-LUTs. The ratio r at minimal pin count in all cases favors the LUT size that is closest to 4, which makes sense since that is the best homogeneous block.

The combination of (6, 4) is also worth noting. In this combination, with ratios 3 and 4, the total numbers of pins are reduced by 8% and 7% with bit number increasing by about 50% and 40%, respectively. The increase in the number of bits roughly offsets the decrease in the number of pins since pins dominate the area. Thus this architecture has nearly equivalent area as homogeneous 4-LUT FPGA. From previous research [Sing92], however, we expect a 6-LUT FPGA to provide roughly 25% less delay, at the system level, than a 4-LUT and so we can expect this combination to exhibit superior speed to the homogeneous 4-LUT FPGA.

s \ p	**7**	**6**	**5**	**4**	**3**
2	$1-2$	1	1	0.5	0.5
3	$2-3$	2	1	$1-2$	
4	$2-4$	$2-4$	$2-4$		
5	$2-6$	$2-5$			
6	$2-6$				

Table 4.3: Ratio **r** with respect to Table 4.1

4.4 Summary

In this chapter, we have described the experimental procedures, the measures of goodness for area-efficiency, and experimental results for an investigation into heterogeneous architecture of FPGAs. A large set of industrial benchmark circuits are mapped into different heterogeneous FPGAs by using the heterogeneous mapper described in Chapter 3, to determine the best heterogeneous architectures. From the experiments, we conclude that:

1. Some heterogeneous FPGAs have superior logic density than the best homogeneous FPGAs.

2. For the (p, s, r) combinations (4, 2, $\frac{1}{2}$) and (4, 3, 1−2), both significant pin and significant bit reduction have been achieved in comparison with the best homogeneous architecture.

3. Some combinations of (p, s, r), such as (5, 2, 1), have reached significant pin reduction with slight bit increase. These combinations are also considered good for area, since pins have a greater effect on area.

4. Certain combinations, such as (6, 4, 2−4), have nearly equivalent area to the best homogeneous 4-LUT FPGA, but are superior in speed.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis has made two major contributions:

1. Developed an algorithm and software tool suitable for *heterogeneous* LUT-based FPGA technology mapping.

   The main features of the mapper are its ability to handle heterogenous logic blocks directly, and the optimal solution to the sub-problem of the selection of mapping solution from a family of fanout-free trees. The new heterogeneous mapper is superior to the heterogeneous mapping from a post-process of a homogenous mapper.

2. Investigated the heterogeneous architectures for area-efficiency, and demonstrated that heterogeneous architectures have superior area-efficiency to homogeneous architectures.

   The thesis investigated the area-efficiency advantages of heterogeneous logic block FPGA architectures in terms of the new mapper. We conclude that certain heterogeneous FPGAs exhibit better area than the most area-efficient homogeneous FPGA. The best ratio r corresponding to these minima differs depending on the size of the lookup tables. We also demonstrated that some heterogeneous mixtures may deliver superior speed with equivalent area to the best homogeneous FPGA.

5.2 Future Work

This work is just a beginning in exploiting the advantages of heterogeneous FPGAs. There are many further issues that need to be investigated. We list a few of them as follows:

1. **To exploit the best speed-area tradeoff**. Speed-area tradeoff is the most important issue of the heterogeneous FPGAs. Previous work has shown that while 4-LUT is best for area, larger (than 4) LUTs have better speed performance since with large LUTs, the number of LUTs in the critical paths of a circuit will be less than that with small LUTs. If we mix larger LUTs, such as 6-LUT, with small LUTs, it may be possible to improve both speed and area. Recall that when we compared our results with homogeneous ones, we compared with the heterogeneous mappings which were optimized for area. However, when speed is considered as the optimization goal. It is usually the case that there will be an increase in area, indicating that the LUTs used will have lower utilization on their used inputs. In such a case, heterogeneous FPGAs may find themselves the best alternatives to the homogeneous FPGAs. For example, we can first map a boolean network into a circuit of homogeneous p-LUT FPGA (e.g. $p = 6$) and optimize it for speed. Then we replace the LUTs in the non-critical paths with area-efficient LUTs and keep the same number of levels of LUTs in the critical paths. In this way, it may be possible to gain improvement on both area and time delay of a mapping.

2. **Issues in heterogeneous FPGA physical design**. Although with heterogeneous FPGAs, one can gain improvement in area and speed, the non-heterogeneity of LUTs will increase the complexity in placement and routing (as well as in technology mapping). More routing resources may be required to make a connection with heterogeneous LUTs than with homogeneous ones. Is there any difference in routing a heterogeneous FPGA and a homogeneous FPGA? If there is any difference, how much is it? What is the best routing architecture for the heterogeneous FPGAs? Investigations into these aspects will be very useful to validate the usability of the heterogeneous FPGAs.

3. **A superior method for post-process-homogeneous mapping of heterogeneous FPGAs**. As we mentioned in Section 3.6, one major reason to cause the post-process-homo to do worse is the transformations between the two LUTs, particularly, when we need a transformation from s-LUTs to p-LUTs. For example, we need three 4-LUTs to make a 5-LUT (refer to Table 2.1). This is usually unnecessary for a given circuit. When we want to replace a p-LUT using a set of s-LUTs, we could instead map the subnetwork inside the p-LUT into a set of homogeneous s-LUTs. It is believed that the number of s-LUTs needed to make a p-LUT this way is fewer than or equal to (at most) the number given in Table 2.1 since the number given in this Table is for the worst case. It should be pointed out that although this re-mapping can improve the quality of post-process-homogeneous mapping, it is a local optimization and it can not improve the quality of transforming p-LUTs into s-LUTs.

Bibliography

[Abou90] P. Abouzeid, L. Bouchet, K. Sakouti, G. Saucier, and P. Sicard, "Lexicographical Expression of Boolean Function for Multilevel Synthesis of High Speed Circuits," *Proc. SASHIMI 90*, Oct. 1990, pp.31-39.

[Ahre90] M. Ahren, A. El Gamal, D. Galbraith, J. Greene, S. Kaptanoglu, K. Djarmarajan, L. Hutchings, S. Ku, P. McGibney, J. McGowan, A. Samie, K. Shaw, N. Stiawalt, T. Whitney, T. Wong, W. Wong, and B. Wu, "An FPGA Family Optimized for High Densities and Reduced Routing Delay," *Proc. 1990 CICC*, May 1990, pp.31.5.1 - 31.5.4.

[Algo89] CAL 1024 Datasheet, Algotronix Ltd. Edinburgh, Scotland, 1989.

[Bray90] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multi-level Logic Synthesis," *Proc. IEEE*, Vol.78, No.2, Feb. 1990, pp.264-300.

[Brit93] B. Britton, D. Hill, W. Oswald, N.-S. Woo, and S. Singh, "Optimized Reconfigurable Cell Array Architecture for High-Performance Field-Programmable Gate Arrays," *Proc. 1993 CICC*, May 1993, pp. 7.2.1-7.2.5.

[Cart86] W. Carter, K. Duong, R. Freeman, H. Hsieh, J. Ja, J. Mahoney, L. Ngo, and S. Sze, "A user Programmable Reconfigurable Gate Array," *Proc. 1986 CICC*, May 1986, pp.233-235.

[Chen92] K. Chen, "Logic Minimization of Lookup-Table Based FPGAs," *Proc. FPGA'92,* Feb. 1992, pp.71-76.

[Cong92] J. Cong, A. Kahng, P. Trajmar, K. Chen, "Graph Based FPGA Technology Mapping for Delay Optimization," *Proc. FPGA'92,* Feb. 1992, pp. 77-81.

[Corn87] D. Corneil and J. Keil, “A Dynamic Programming Approach to the Dominating Set Problem on k-Trees”, *SIAM J. ALG. Disc Meth.*, Vol.8, No.4, Oct.1987, pp. 535-543.

[ElGa89] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, “An Architecture for Electrically Configurable Gate Arrays,” *IEEE J. Solid State Circuits*. Vol. 24, No. 2, April 1989, pp.394-398.

[Fark92] K. Farkas, “Non-homogeneous FPGA Logic Block Architectures”, *Graduate Course Project*, University of Toronto, January, 1992.

[Filo91] D. Filo, J. Yang, F. Mailhot, and G. De Micheli,“Technology Mapping for a Two-Output RAM-based Field Programmable Gate Array”,*Proc. EDAC91,* Feb. 1991,pp. 534-538.

[Fran90] R. Francis, J. Rose, and K. Chung, “Chortle: A Technology Mapping Program for Lookup Table-Based Field-Programmable Gate Arrays,” *Proc. 27th Design Automation Conference,* June 1990, pp. 613-619.

[Fran91a] R. Francis, J. Rose, and Z. Vranesic, “Chortle-crf:Fast Technology Mapping for Lookup Table-Based FPGAs”, *Proc. 28th Design Automation Conference*, June, 1991, pp. 227 - 233.

[Fran91b] R. Francis, J. Rose, and Z. Vranesic, “Technology Mapping of Lookup Table-Based FPGAs for Performance,” *Proc. ICCAD-91,* Nov., 1991.

[Fran92a] R. Francis, “Technology Mapping for Lookup Table-Based Field-Programmable Gate Arrays,” *Ph.D Thesis,* University of Toronto, 1992.

[Fran92b] R. Francis, “A Better Method for Post-Process-Homogeneous Technology Mapping of LUT-Based FPGAs,” *Private Communications,* University of Toronto, 1992.

[He93] J. He and J. Rose, “Advantages of Heterogeneous Logic Blocks in FPGAs,” *Proc. 1993 Custom Integrated Circuits Conference*, May 1993.

[He94] J. He and J. Rose, “Technology Mapping for Heterogeneous FPGAs,” *Proc. 1994 ACM/SIGDA International WorkShop on Field-Programmable Gate Arrays*, Feb. 1994.

[Hill91] D. Hill and N-S Woo, "The Benefits of Flexibility in Look-up Table FPGAs", in **FPGAs**, W. Moore and W. Luk Eds., Abingdon 1991, edited from the *Oxford 1991 International Workshop on Field Programmable Logic and Applications*, pp.127-136.

[Hsie90] H. Hsieh, W. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. TinKey, and R. Kanazawa, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *Proc. 1990 CICC*, May 1990, pp. 31.2.1-31.2.7

[Hutt93] M. Hutton, "Combining Independent Two-Parameter Optimization Subproblems," University of Toronto, Private Communication, 1993.

[Karp91] K. Karplus, "Xmap: A Technology Mapper for Table-Lookup Field- Programmable Gate Arrays",*Proc. 28th Design Automation Conference*,June 1991,pp. 240-243.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching",*Proc. 24th Design Automation Conference*, June 1987, pp. 341-347.

[Koul91] J. Kouloheris and A. El Gamal "FPGA Performance vs. Cell Granularity," *Proc. 1991 CICC*, May 1991, pp. 6.2.1 - 6.2.4.

[Koul92] J. Kouloheris and A. El Gamal, "FPGA Area vesus Cell Granularity - lookup Tables and PLA Cells," *ACM/SIGDA Workshop on FPGAs* (FPGA'92), Feb. 1992, pp.9-14.

[Mizu92] C. Mizuyabu, "Heterogeneous Lookup Table Logic Blocks in Field-Programmable Gate Arrays", *B.Sc. Thesis*, University of Toronto, April, 1992.

[Murg90] R. Murgai, Y. Nishizaki, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," *Proc. 27th Design Automation Conference,* June 1990, pp. 620-625.

[Murg91a] R. Murgai, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD,* 1991.

[Murg91b] R. Murgai, N. Shenay, and R. Brayton, “Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,” *ICCAD*, 1991.

[Murg91c] R. Murgai, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, “Improved Logic Synthesis Algorithms for Table Look Up Architecture”, *Proc. ICCAD*, Nov. 1991.

[Rose90] J. Rose, R.J. Francis, D. Lewis, and P. Chow, “Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency,” *IEEE JSSC*, Vol. 25 No. 5, October 1990, pp. 1217-1225.

[Sava76] J. Savage, “The Complexity of Computing”, *Wiley Inter-science Publication*, 1976.

[Sawk92] P. Sawkar and D. Thomas, “Technology Mapping for Table-Look-Up based Field-Programmable Gate Arrays,” *Proc. FPGA'92,* Feb. 1992, pp. 83-88.

[Sing91] S. Singh, “The Effect of Logic Block Architecture on the Speed of Field-Programmable Gate Arrays,” M.A.Sc. Thesis, University of Toronto, 1991.

[Sing92] S. Singh, J. Rose, P. Chow, D. Lewis, “The Effect of Logic Block Architecture on FPGA Performance,” *IEEE JSSC*, Vol. 27 No. 3, March 1992, pp. 281-287.

[Toua92] H. Touati, N. Shenoy, A. Sangiovanni-Vincentelli, “Re-timing for Table-Lookup Field-Programmable Gate Arrays,” *Proc. FPGA'92,* Feb. 1992, pp. 89-94.

[Wils92] R. Wilson, “Altera Flexes Programmable Logic Muscles,” *Electronic Engineering Times*, Oct.5, 1992.

[Wong89] S. Wong, H. So, J. Ou, and J. Costello, “A 5000-Gate CMOS EPLD with Multiple Logic and Interconnect Arrays,” *Proc. 1989 Custom Integrated Circuits Conference*, May 1989, pp. 5.8.1-5.8.4.

[Woo91] N. Woo, “A Heuristic Method for FPGA Technology Mapping Based on Edge Visibility,” *Proc. 28th Design Automation Conference,* June, 1991,

Appendix A

Appendix

Heterogeneous Mapping Results for Individual Circuits

In this appendix, we list the performance of the heterogeneous mapper on the basis of individual circuits. The list includes a large set of selected tuples of $H = (p, s, r)$, where $H = (4, 2, r), (4, 3, r), (5, 2, r)$, and $(6, 4, r)$, where $r = \frac{1}{10}, \frac{1}{9}, \cdots, 1, 2, \cdots, 9, 10$. For each (p, s, r), the number of supertiles generated by the heterogeneous mapper for a specific circuit is given in a table form. In the vertical direction of the tables are the circuit names. All 40 MCNC benchmark circuits that was used in the experiments are listed. In the horizontal direction is the combination of (p, s, r). Note that from the number of supertiles plus the information of (p, s, r), one can easily calculate the number of pins and the number of bits as in Chapter 4. For the comparison purpose, the number of supertiles from the post-process-homogeneous mapping (PPH) and difference between the heterogeneous and PPH mapping are also presented, in parallel with the heterogeneous results.

Circuit	N_{sup} (4, 2, $\frac{1}{10}$)			N_{sup} (4, 2, $\frac{1}{9}$)			N_{sup} (4, 2, $\frac{1}{8}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	16	16	0.0	17	17	0.0	19	19	0.0
C432	11	11	0.0	12	12	0.0	13	13	0.0
C880	12	12	0.0	13	13	0.0	14	14	0.0
alu2	14	14	0.0	16	16	0.0	17	18	5.9
alu4	24	24	0.0	27	27	0.0	30	30	0.0
apex6	24	24	0.0	26	26	0.0	29	29	0.0
apex7	9	9	0.0	9	9	0.0	10	10	0.0
b9	5	5	0.0	5	5	0.0	6	6	0.0
c8	5	5	0.0	5	5	0.0	5	6	20.0
cht	5	5	0.0	5	5	0.0	6	6	0.0
cm150a	2	2	0.0	2	2	0.0	2	2	0.0
cm151a	1	1	0.0	1	1	0.0	1	1	0.0
cm85a	2	2	0.0	2	2	0.0	2	2	0.0
cmb	2	2	0.0	2	2	0.0	2	2	0.0
count	5	5	0.0	5	5	0.0	6	6	0.0
example2	12	12	0.0	13	13	0.0	14	14	0.0
frg1	4	4	0.0	5	5	0.0	5	5	0.0
frg2	30	30	0.0	33	33	0.0	36	36	0.0
i1	2	2	0.0	2	2	0.0	3	3	0.0
i6	14	14	0.0	15	15	0.0	17	16	-5.9
i7	17	17	0.0	19	19	0.0	21	21	0.0
i8	33	33	0.0	37	37	0.0	41	41	0.0
i9	20	19	-5.0	22	21	-4.5	24	24	0.0
k2	35	35	0.0	38	38	0.0	42	42	0.0
my-adder	6	6	0.0	7	7	0.0	8	8	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	3	3	0.0	4	4	0.0	4	4	0.0
pm1	2	2	0.0	2	2	0.0	2	2	0.0
rot	23	23	0.0	26	26	0.0	28	29	3.6
sct	3	3	0.0	3	3	0.0	3	3	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	5	5	0.0	5	5	0.0	6	6	0.0
ttt2	6	6	0.0	6	6	0.0	7	7	0.0
unreg	4	4	0.0	4	4	0.0	5	4	-20.0
vda	20	20	0.0	22	22	0.0	24	24	0.0
x1	12	12	0.0	13	13	0.0	14	14	0.0
x2	2	2	0.0	2	2	0.0	2	2	0.0
x3	27	27	0.0	29	29	0.0	32	32	0.0
x4	13	13	0.0	14	14	0.0	16	16	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	436	435	-0.2	474	473	-0.2	522	523	0.2

Table A.1: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, $\frac{1}{7}$)			N_{sup} (4, 2, $\frac{1}{6}$)			N_{sup} (4, 2, $\frac{1}{5}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	21	21	0.0	25	24	-4.0	28	28	0.0
C432	15	15	0.0	17	17	0.0	20	20	0.0
C880	16	16	0.0	18	18	0.0	21	21	0.0
alu2	20	20	0.0	22	22	0.0	26	26	0.0
alu4	33	33	0.0	38	38	0.0	44	44	0.0
apex6	33	33	0.0	37	37	0.0	44	43	-2.3
apex7	12	12	0.0	13	13	0.0	15	15	0.0
b9	7	7	0.0	7	7	0.0	9	9	0.0
c8	6	6	0.0	7	7	0.0	8	8	0.0
cht	6	6	0.0	7	7	0.0	8	8	0.0
cm150a	2	2	0.0	3	2	-33.3	3	3	0.0
cm151a	1	1	0.0	2	2	0.0	2	2	0.0
cm85a	2	2	0.0	3	3	0.0	3	3	0.0
cmb	3	3	0.0	3	3	0.0	3	3	0.0
count	6	6	0.0	7	7	0.0	8	8	0.0
example2	16	16	0.0	18	18	0.0	21	21	0.0
frg1	6	6	0.0	7	7	0.0	8	8	0.0
frg2	41	41	0.0	46	46	0.0	54	54	0.0
i1	3	3	0.0	3	3	0.0	4	4	0.0
i6	19	18	-5.3	22	21	-4.5	27	24	-11.1
i7	23	23	0.0	26	26	0.0	31	31	0.0
i8	46	46	0.0	52	52	0.0	61	61	0.0
i9	28	27	-3.6	32	31	-3.1	38	36	-5.3
k2	48	48	0.0	54	54	0.0	63	63	0.0
my-adder	8	8	0.0	10	10	0.0	11	11	0.0
parity	2	2	0.0	3	3	0.0	3	3	0.0
pcler	4	4	0.0	5	5	0.0	6	6	0.0
pm1	3	3	0.0	3	3	0.0	3	3	0.0
rot	32	32	0.0	36	37	2.8	42	43	2.4
sct	4	4	0.0	4	4	0.0	5	5	0.0
t481	2	2	0.0	2	2	0.0	3	3	0.0
term1	7	7	0.0	7	8	14.3	9	9	0.0
ttt2	8	8	0.0	9	9	0.0	10	10	0.0
unreg	5	5	0.0	6	6	0.0	7	7	0.0
vda	27	27	0.0	31	31	0.0	36	36	0.0
x1	16	16	0.0	18	18	0.0	21	21	0.0
x2	3	3	0.0	3	3	0.0	3	3	0.0
x3	36	36	0.0	41	41	0.0	48	48	0.0
x4	18	18	0.0	20	20	0.0	23	23	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	590	588	-0.3	669	667	-0.3	781	776	-0.6

Table A.2: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, $\frac{1}{4}$)			N_{sup} (4, 2, $\frac{1}{3}$)			N_{sup} (4, 2, $\frac{1}{2}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	34	34	0.0	42	42	0.0	56	56	0.0
C432	24	24	0.0	29	30	3.4	40	39	-2.5
C880	25	25	0.0	31	31	0.0	45	44	-2.2
alu2	31	31	0.0	39	39	0.0	54	52	-3.7
alu4	53	53	0.0	66	66	0.0	92	90	-2.2
apex6	52	52	0.0	66	65	-1.5	96	91	-5.2
apex7	18	18	0.0	23	23	0.0	32	30	-6.2
b9	10	10	0.0	13	13	0.0	17	17	0.0
c8	9	10	11.1	12	12	0.0	15	16	6.7
cht	10	10	0.0	13	12	-7.7	19	18	-5.3
cm150a	4	3	-25.0	5	4	-20.0	6	6	0.0
cm151a	2	2	0.0	2	2	0.0	3	3	0.0
cm85a	3	3	0.0	4	4	0.0	5	5	0.0
cmb	4	4	0.0	5	5	0.0	7	7	0.0
count	10	10	0.0	12	12	0.0	16	16	0.0
example2	25	25	0.0	31	31	0.0	41	41	0.0
frg1	9	9	0.0	12	11	-8.3	17	16	-5.9
frg2	65	65	0.0	81	81	0.0	108	108	0.0
i1	4	4	0.0	5	5	0.0	7	7	0.0
i6	33	29	-12.1	43	36	-16.3	62	50	-19.4
i7	37	37	0.0	46	46	0.0	65	64	-1.5
i8	75	73	-2.7	99	94	-5.1	143	132	-7.7
i9	47	44	-6.4	61	57	-6.6	89	79	-11.2
k2	76	76	0.0	95	95	0.0	128	126	-1.6
my-adder	13	13	0.0	16	16	0.0	22	22	0.0
parity	3	3	0.0	4	4	0.0	5	5	0.0
pcler	7	7	0.0	8	8	0.0	11	11	0.0
pm1	4	4	0.0	5	5	0.0	6	6	0.0
rot	51	51	0.0	63	64	1.6	84	85	1.2
sct	6	6	0.0	7	7	0.0	9	9	0.0
t481	3	3	0.0	4	4	0.0	5	5	0.0
term1	10	10	0.0	13	13	0.0	18	17	-5.6
ttt2	12	12	0.0	15	15	0.0	22	20	-9.1
unreg	9	8	-11.1	11	10	-9.1	16	14	-12.5
vda	43	43	0.0	54	54	0.0	72	72	0.0
x1	25	25	0.0	31	31	0.0	42	41	-2.4
x2	4	4	0.0	5	5	0.0	6	6	0.0
x3	58	58	0.0	72	72	0.0	105	100	-4.8
x4	28	28	0.0	35	35	0.0	49	47	-4.1
z4ml	3	3	0.0	3	3	0.0	4	4	0.0
total	939	929	-1.1	1181	1162	-1.6	1639	1577	-3.8

Table A.3: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, 1)			N_{sup} (4, 2, 2)			N_{sup} (4, 2, 3)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	84	84	0.0	56	56	0.0	42	42	0.0
C432	72	65	-9.7	62	51	-17.7	57	43	-24.6
C880	82	72	-12.2	71	56	-21.1	62	47	-24.2
alu2	99	88	-11.1	88	70	-20.5	83	59	-28.9
alu4	168	150	-10.7	148	120	-18.9	139	100	-28.1
apex6	175	156	-10.9	158	125	-20.9	148	104	-29.7
apex7	58	49	-15.5	50	39	-22.0	44	32	-27.3
b9	28	26	-7.1	24	20	-16.7	21	17	-19.0
c8	27	25	-7.4	23	20	-13.0	20	17	-15.0
cht	35	30	-14.3	33	24	-27.3	31	20	-35.5
cm150a	11	10	-9.1	11	8	-27.3	10	7	-30.0
cm151a	6	5	-16.7	5	4	-20.0	5	4	-20.0
cm85a	8	8	0.0	7	6	-14.3	7	5	-28.6
cmb	13	11	-15.4	11	8	-27.3	10	7	-30.0
count	30	26	-13.3	28	20	-28.6	27	16	-40.7
example2	72	65	-9.7	62	51	-17.7	54	42	-22.2
frg1	30	27	-10.0	28	22	-21.4	27	19	-29.6
frg2	197	178	-9.6	169	140	-17.2	152	117	-23.0
i1	10	10	0.0	9	8	-11.1	8	6	-25.0
i6	114	83	-27.2	98	65	-33.7	85	55	-35.3
i7	126	112	-11.1	117	89	-23.9	110	75	-31.8
i8	262	224	-14.5	229	179	-21.8	215	150	-30.2
i9	163	132	-19.0	140	106	-24.3	128	88	-31.2
k2	235	210	-10.6	201	158	-21.4	176	131	-25.6
my-adder	32	32	0.0	28	24	-14.3	24	20	-16.7
parity	8	8	0.0	5	5	0.0	4	4	0.0
pcler	17	17	0.0	15	13	-13.3	13	11	-15.4
pm1	11	10	-9.1	9	8	-11.1	8	6	-25.0
rot	152	138	-9.2	131	105	-19.8	114	88	-22.8
sct	16	15	-6.2	14	11	-21.4	12	9	-25.0
t481	7	7	0.0	5	5	0.0	4	4	0.0
term1	33	29	-12.1	29	23	-20.7	26	19	-26.9
ttt2	40	33	-17.5	34	25	-26.5	30	21	-30.0
unreg	29	23	-20.7	25	17	-32.0	22	14	-36.4
vda	130	118	-9.2	111	89	-19.8	97	74	-23.7
x1	76	69	-9.2	69	56	-18.8	64	46	-28.1
x2	11	10	-9.1	10	8	-20.0	9	7	-22.2
x3	192	168	-12.5	165	134	-18.8	155	112	-27.7
x4	89	78	-12.4	76	59	-22.4	67	48	-28.4
z4ml	6	6	0.0	6	5	-16.7	5	4	-20.0
total	2954	2607	-11.7	2560	2032	-20.6	2315	1690	-27.0

Table A.4: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, 4)			N_{sup} (4, 2, 5)			N_{sup} (4, 2, 6)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	34	34	0.0	28	28	0.0	24	24	0.0
C432	54	37	-31.5	51	32	-37.3	48	29	-39.6
C880	58	40	-31.0	55	35	-36.4	52	31	-40.4
alu2	78	50	-35.9	73	44	-39.7	70	39	-44.3
alu4	131	86	-34.4	123	75	-39.0	117	67	-42.7
apex6	140	89	-36.4	132	78	-40.9	125	70	-44.0
apex7	39	28	-28.2	36	24	-33.3	34	22	-35.3
b9	19	14	-26.3	17	13	-23.5	16	11	-31.2
c8	19	15	-21.1	18	13	-27.8	17	12	-29.4
cht	29	18	-37.9	28	15	-46.4	26	14	-46.2
cm150a	9	6	-33.3	9	6	-33.3	9	5	-44.4
cm151a	5	3	-40.0	4	3	-25.0	4	3	-25.0
cm85a	7	5	-28.6	6	4	-33.3	6	4	-33.3
cmb	9	6	-33.3	8	5	-37.5	7	5	-28.6
count	25	14	-44.0	24	13	-45.8	23	11	-52.2
example2	48	36	-25.0	43	32	-25.6	40	28	-30.0
frg1	25	16	-36.0	24	14	-41.7	23	13	-43.5
frg2	143	100	-30.1	135	88	-34.8	128	78	-39.1
i1	7	6	-14.3	6	5	-16.7	6	4	-33.3
i6	76	48	-36.8	68	43	-36.8	62	39	-37.1
i7	104	64	-38.5	98	57	-41.8	93	52	-44.1
i8	202	128	-36.6	191	112	-41.4	181	100	-44.8
i9	120	76	-36.7	114	66	-42.1	108	59	-45.4
k2	157	113	-28.0	146	99	-32.2	138	88	-36.2
my-adder	22	16	-27.3	20	14	-30.0	18	13	-27.8
parity	3	3	0.0	3	3	0.0	3	3	0.0
pcler	12	9	-25.0	11	8	-27.3	11	7	-36.4
pm1	7	5	-28.6	7	5	-28.6	6	4	-33.3
rot	102	75	-26.5	94	66	-29.8	89	59	-33.7
sct	11	8	-27.3	10	7	-30.0	9	6	-33.3
t481	3	3	0.0	3	3	0.0	3	3	0.0
term1	25	16	-36.0	23	14	-39.1	22	13	-40.9
ttt2	27	18	-33.3	24	16	-33.3	22	14	-36.4
unreg	19	12	-36.8	17	11	-35.3	16	10	-37.5
vda	87	63	-27.6	82	55	-32.9	77	49	-36.4
x1	61	40	-34.4	57	35	-38.6	54	31	-42.6
x2	9	6	-33.3	8	5	-37.5	8	5	-37.5
x3	146	96	-34.2	138	84	-39.1	131	75	-42.7
x4	59	41	-30.5	53	36	-32.1	49	32	-34.7
z4ml	4	3	-25.0	4	3	-25.0	4	3	-25.0
total	2135	1446	-32.3	1991	1269	-36.3	1879	1135	-39.6

Table A.5: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, 7)			N_{sup} (4, 2, 8)			N_{sup} (4, 2, 9)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	21	21	0.0	19	19	0.0	17	17	0.0
C432	46	26	-43.5	44	24	-45.5	42	22	-47.6
C880	49	29	-40.8	47	26	-44.7	45	24	-46.7
alu2	66	35	-47.0	63	32	-49.2	60	30	-50.0
alu4	111	60	-45.9	106	55	-48.1	101	50	-50.5
apex6	119	63	-47.1	113	57	-49.6	108	52	-51.9
apex7	33	20	-39.4	31	18	-41.9	30	16	-46.7
b9	15	10	-33.3	14	9	-35.7	14	9	-35.7
c8	16	10	-37.5	15	10	-33.3	15	9	-40.0
cht	25	12	-52.0	24	11	-54.2	23	10	-56.5
cm150a	8	5	-37.5	8	4	-50.0	7	4	-42.9
cm151a	4	3	-25.0	4	2	-50.0	4	2	-50.0
cm85a	6	3	-50.0	5	3	-40.0	5	3	-40.0
cmb	7	4	-42.9	7	4	-42.9	6	4	-33.3
count	21	10	-52.4	20	10	-50.0	20	9	-55.0
example2	36	26	-27.8	34	23	-32.4	33	21	-36.4
frg1	21	11	-47.6	20	10	-50.0	20	10	-50.0
frg2	122	70	-42.6	116	64	-44.8	111	59	-46.8
i1	6	4	-33.3	6	4	-33.3	5	3	-40.0
i6	57	35	-38.6	53	32	-39.6	49	30	-38.8
i7	88	47	-46.6	84	43	-48.8	80	40	-50.0
i8	172	90	-47.7	164	82	-50.0	156	75	-51.9
i9	102	53	-48.0	98	48	-51.0	93	44	-52.7
k2	131	79	-39.7	125	72	-42.4	119	66	-44.5
my-adder	16	12	-25.0	16	11	-31.2	15	10	-33.3
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	10	7	-30.0	10	6	-40.0	9	6	-33.3
pm1	6	4	-33.3	6	4	-33.3	5	3	-40.0
rot	85	53	-37.6	81	48	-40.7	77	44	-42.9
sct	9	6	-33.3	8	5	-37.5	8	5	-37.5
t481	3	2	-33.3	2	2	0.0	2	2	0.0
term1	21	12	-42.9	20	11	-45.0	19	10	-47.4
ttt2	21	13	-38.1	20	12	-40.0	19	11	-42.1
unreg	15	9	-40.0	15	8	-46.7	14	7	-50.0
vda	74	44	-40.5	70	40	-42.9	67	37	-44.8
x1	52	28	-46.2	49	26	-46.9	47	23	-51.1
x2	7	4	-42.9	7	4	-42.9	7	4	-42.9
x3	124	67	-46.0	118	61	-48.3	113	56	-50.4
x4	46	29	-37.0	44	26	-40.9	42	24	-42.9
z4ml	3	3	0.0	3	2	-33.3	3	2	-33.3
total	1776	1021	-42.5	1691	930	-45.0	1612	855	-47.0

Table A.6: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 2, 10)			N_{sup} (4, 3, $\frac{1}{10}$)			N_{sup} (4, 3, $\frac{1}{9}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	16	16	0.0	16	16	0.0	17	17	0.0
C432	40	20	-50.0	11	11	0.0	12	12	0.0
C880	43	23	-46.5	12	12	0.0	13	13	0.0
alu2	58	27	-53.4	14	14	0.0	16	16	0.0
alu4	97	47	-51.5	24	24	0.0	27	27	0.0
apex6	103	48	-53.4	24	24	0.0	26	26	0.0
apex7	28	15	-46.4	9	9	0.0	9	9	0.0
b9	13	8	-38.5	5	5	0.0	5	5	0.0
c8	14	8	-42.9	5	5	0.0	5	5	0.0
cht	22	10	-54.5	5	5	0.0	5	5	0.0
cm150a	7	4	-42.9	2	2	0.0	2	2	0.0
cm151a	4	2	-50.0	1	1	0.0	1	1	0.0
cm85a	5	3	-40.0	2	2	0.0	2	2	0.0
cmb	6	3	-50.0	2	2	0.0	2	2	0.0
count	19	8	-57.9	5	5	0.0	5	5	0.0
example2	31	20	-35.5	12	12	0.0	13	13	0.0
frg1	19	9	-52.6	4	4	0.0	5	5	0.0
frg2	106	54	-49.1	30	30	0.0	33	33	0.0
i1	5	3	-40.0	2	2	0.0	2	2	0.0
i6	46	28	-39.1	14	14	0.0	15	15	0.0
i7	77	37	-51.9	17	17	0.0	19	19	0.0
i8	149	69	-53.7	33	33	0.0	37	37	0.0
i9	89	41	-53.9	19	19	0.0	21	21	0.0
k2	114	61	-46.5	35	35	0.0	38	38	0.0
my-adder	14	9	-35.7	6	6	0.0	7	7	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	9	5	-44.4	3	3	0.0	4	4	0.0
pm1	5	3	-40.0	2	2	0.0	2	2	0.0
rot	74	41	-44.6	23	23	0.0	26	26	0.0
sct	8	5	-37.5	3	3	0.0	3	3	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	18	9	-50.0	5	5	0.0	5	5	0.0
ttt2	18	10	-44.4	6	6	0.0	6	6	0.0
unreg	13	7	-46.2	4	4	0.0	4	4	0.0
vda	64	34	-46.9	20	20	0.0	22	22	0.0
x1	45	22	-51.1	12	11	-8.3	13	13	0.0
x2	6	4	-33.3	2	2	0.0	2	2	0.0
x3	108	52	-51.9	27	27	0.0	29	29	0.0
x4	40	22	-45.0	13	13	0.0	14	14	0.0
z4ml	3	2	-33.3	2	2	0.0	2	2	0.0
total	1540	793	-48.5	435	434	-0.2	473	473	0.0

Table A.7: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, $\frac{1}{8}$)			N_{sup} (4, 3, $\frac{1}{7}$)			N_{sup} (4, 3, $\frac{1}{6}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	19	19	0.0	21	21	0.0	25	24	-4.0
C432	13	13	0.0	15	15	0.0	17	17	0.0
C880	14	14	0.0	16	16	0.0	18	18	0.0
alu2	17	17	0.0	20	20	0.0	22	22	0.0
alu4	30	30	0.0	33	33	0.0	38	38	0.0
apex6	29	29	0.0	33	33	0.0	37	37	0.0
apex7	10	10	0.0	12	12	0.0	13	13	0.0
b9	6	6	0.0	7	7	0.0	8	7	-12.5
c8	5	5	0.0	6	6	0.0	7	7	0.0
cht	6	6	0.0	6	6	0.0	7	7	0.0
cm150a	2	2	0.0	2	2	0.0	2	2	0.0
cm151a	1	1	0.0	1	1	0.0	2	2	0.0
cm85a	2	2	0.0	2	2	0.0	3	3	0.0
cmb	2	2	0.0	3	3	0.0	3	3	0.0
count	6	6	0.0	6	6	0.0	7	7	0.0
example2	14	14	0.0	16	16	0.0	18	18	0.0
frg1	5	5	0.0	6	6	0.0	7	7	0.0
frg2	36	36	0.0	41	41	0.0	47	46	-2.1
i1	3	3	0.0	3	3	0.0	3	3	0.0
i6	16	16	0.0	18	18	0.0	21	21	0.0
i7	21	21	0.0	23	23	0.0	26	26	0.0
i8	41	41	0.0	46	46	0.0	52	52	0.0
i9	23	23	0.0	26	26	0.0	29	29	0.0
k2	42	42	0.0	48	48	0.0	54	54	0.0
my-adder	8	8	0.0	8	8	0.0	10	10	0.0
parity	2	2	0.0	2	2	0.0	3	3	0.0
pcler	4	4	0.0	4	4	0.0	5	5	0.0
pm1	2	2	0.0	3	3	0.0	3	3	0.0
rot	28	28	0.0	32	32	0.0	36	36	0.0
sct	3	3	0.0	4	4	0.0	4	4	0.0
t481	2	2	0.0	2	2	0.0	3	2	-33.3
term1	6	6	0.0	7	7	0.0	8	7	-12.5
ttt2	7	7	0.0	8	8	0.0	9	9	0.0
unreg	4	4	0.0	5	5	0.0	5	5	0.0
vda	24	24	0.0	27	27	0.0	31	31	0.0
x1	14	14	0.0	16	16	0.0	18	18	0.0
x2	2	2	0.0	3	3	0.0	3	3	0.0
x3	32	32	0.0	36	36	0.0	41	41	0.0
x4	16	16	0.0	18	18	0.0	20	20	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	519	519	0.0	587	587	0.0	667	662	-0.7

Table A.8: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, $\frac{1}{5}$)			N_{sup} (4, 3, $\frac{1}{4}$)			N_{sup} (4, 3, $\frac{1}{3}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	28	28	0.0	34	34	0.0	42	42	0.0
C432	20	20	0.0	24	24	0.0	29	29	0.0
C880	21	21	0.0	25	25	0.0	31	31	0.0
alu2	26	26	0.0	31	31	0.0	39	39	0.0
alu4	44	44	0.0	53	53	0.0	66	66	0.0
apex6	43	43	0.0	52	52	0.0	65	65	0.0
apex7	15	15	0.0	18	18	0.0	23	23	0.0
b9	9	9	0.0	10	10	0.0	13	13	0.0
c8	8	8	0.0	9	9	0.0	12	12	0.0
cht	8	8	0.0	10	10	0.0	12	12	0.0
cm150a	3	3	0.0	3	3	0.0	4	4	0.0
cm151a	2	2	0.0	2	2	0.0	2	2	0.0
cm85a	3	3	0.0	3	3	0.0	4	4	0.0
cmb	3	3	0.0	4	4	0.0	5	5	0.0
count	8	8	0.0	10	10	0.0	12	12	0.0
example2	21	21	0.0	25	25	0.0	31	31	0.0
frg1	8	8	0.0	9	9	0.0	11	11	0.0
frg2	54	54	0.0	65	65	0.0	81	81	0.0
i1	4	4	0.0	4	4	0.0	5	5	0.0
i6	24	24	0.0	29	29	0.0	36	36	0.0
i7	31	31	0.0	37	37	0.0	46	46	0.0
i8	61	61	0.0	73	73	0.0	91	91	0.0
i9	34	34	0.0	41	41	0.0	51	51	0.0
k2	63	63	0.0	76	76	0.0	95	95	0.0
my-adder	11	11	0.0	13	13	0.0	16	16	0.0
parity	3	3	0.0	3	3	0.0	4	4	0.0
pcler	6	6	0.0	7	7	0.0	8	8	0.0
pm1	3	3	0.0	4	4	0.0	5	5	0.0
rot	42	42	0.0	51	51	0.0	63	63	0.0
sct	5	5	0.0	6	6	0.0	7	7	0.0
t481	3	3	0.0	3	3	0.0	4	4	0.0
term1	9	9	0.0	10	10	0.0	13	13	0.0
ttt2	10	10	0.0	12	12	0.0	15	15	0.0
unreg	6	6	0.0	7	7	0.0	9	9	0.0
vda	36	36	0.0	43	43	0.0	54	54	0.0
x1	21	21	0.0	25	25	0.0	31	31	0.0
x2	3	3	0.0	4	4	0.0	5	5	0.0
x3	48	48	0.0	58	58	0.0	72	72	0.0
x4	23	23	0.0	28	28	0.0	35	35	0.0
z4ml	2	2	0.0	3	3	0.0	3	3	0.0
total	772	772	0.0	924	924	0.0	1150	1150	0.0

Table A.9: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, $\frac{1}{2}$)			N_{sup} (4, 3, 1)			N_{sup} (4, 3, 2)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	56	56	0.0	84	84	0.0	56	56	0.0
C432	39	39	0.0	59	58	-1.7	47	41	-12.8
C880	42	42	0.0	62	62	0.0	50	43	-14.0
alu2	51	51	0.0	84	78	-7.1	67	56	-16.4
alu4	88	88	0.0	141	133	-5.7	113	95	-15.9
apex6	86	86	0.0	147	134	-8.8	118	96	-18.6
apex7	30	30	0.0	45	45	0.0	33	31	-6.1
b9	17	17	0.0	25	25	0.0	17	17	0.0
c8	15	15	0.0	23	23	0.0	17	15	-11.8
cht	18	17	-5.6	30	28	-6.7	24	21	-12.5
cm150a	5	5	0.0	9	7	-22.2	7	5	-28.6
cm151a	3	3	0.0	5	4	-20.0	4	3	-25.0
cm85a	5	5	0.0	8	8	0.0	6	5	-16.7
cmb	6	6	0.0	9	9	0.0	7	6	-14.3
count	16	16	0.0	28	24	-14.3	22	16	-27.3
example2	41	41	0.0	61	61	0.0	41	41	0.0
frg1	15	15	0.0	26	23	-11.5	21	17	-19.0
frg2	108	108	0.0	161	161	0.0	127	113	-11.0
i1	7	7	0.0	10	10	0.0	7	7	0.0
i6	48	48	0.0	72	72	0.0	48	48	0.0
i7	64	61	-4.7	111	92	-17.1	89	66	-25.8
i8	121	121	0.0	207	183	-11.6	165	131	-20.6
i9	68	68	0.0	117	107	-8.5	94	77	-18.1
k2	126	126	0.0	189	189	0.0	136	126	-7.4
my-adder	22	22	0.0	32	32	0.0	22	22	0.0
parity	5	5	0.0	8	8	0.0	5	5	0.0
pcler	11	11	0.0	16	16	0.0	12	11	-8.3
pm1	6	6	0.0	9	9	0.0	6	6	0.0
rot	84	84	0.0	126	126	0.0	90	85	-5.6
sct	9	9	0.0	13	13	0.0	9	9	0.0
t481	5	5	0.0	7	7	0.0	5	5	0.0
term1	17	17	0.0	26	25	-3.8	21	18	-14.3
ttt2	20	20	0.0	30	30	0.0	21	20	-4.8
unreg	12	12	0.0	17	17	0.0	14	13	-7.1
vda	72	72	0.0	108	108	0.0	78	73	-6.4
x1	41	41	0.0	66	62	-6.1	53	44	-17.0
x2	6	6	0.0	9	9	0.0	8	7	-12.5
x3	96	96	0.0	155	144	-7.1	124	102	-17.7
x4	46	46	0.0	69	69	0.0	47	46	-2.1
z4ml	4	4	0.0	6	6	0.0	4	4	0.0
total	1531	1527	-0.3	2410	2301	-4.5	1835	1602	-12.7

Table A.10: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, 3)			N_{sup} (4, 3, 4)			N_{sup} (4, 3, 5)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	42	42	0.0	34	34	0.0	28	28	0.0
C432	39	32	-17.9	34	26	-23.5	30	22	-26.7
C880	41	33	-19.5	36	27	-25.0	31	23	-25.8
alu2	56	44	-21.4	48	36	-25.0	42	31	-26.2
alu4	94	74	-21.3	81	61	-24.7	71	52	-26.8
apex6	98	77	-21.4	84	64	-23.8	74	55	-25.7
apex7	28	24	-14.3	24	20	-16.7	21	17	-19.0
b9	14	13	-7.1	12	11	-8.3	11	9	-18.2
c8	14	12	-14.3	12	10	-16.7	11	8	-27.3
cht	20	17	-15.0	18	14	-22.2	15	12	-20.0
cm150a	6	4	-33.3	5	3	-40.0	5	3	-40.0
cm151a	3	2	-33.3	3	2	-33.3	3	2	-33.3
cm85a	5	4	-20.0	5	4	-20.0	4	3	-25.0
cmb	6	5	-16.7	5	4	-20.0	4	4	0.0
count	19	13	-31.6	16	11	-31.2	14	9	-35.7
example2	32	31	-3.1	28	25	-10.7	24	22	-8.3
frg1	18	13	-27.8	15	11	-26.7	13	9	-30.8
frg2	106	88	-17.0	91	73	-19.8	79	63	-20.3
i1	6	5	-16.7	5	5	0.0	4	4	0.0
i6	39	37	-5.1	34	31	-8.8	29	26	-10.3
i7	74	51	-31.1	64	42	-34.4	56	36	-35.7
i8	138	102	-26.1	118	84	-28.8	104	71	-31.7
i9	78	60	-23.1	67	49	-26.9	59	42	-28.8
k2	114	98	-14.0	97	80	-17.5	85	68	-20.0
my-adder	16	16	0.0	14	14	0.0	12	12	0.0
parity	4	4	0.0	3	3	0.0	3	3	0.0
pcler	10	9	-10.0	8	7	-12.5	7	6	-14.3
pm1	5	5	0.0	5	4	-20.0	4	4	0.0
rot	75	66	-12.0	64	54	-15.6	56	47	-16.1
sct	8	7	-12.5	7	6	-14.3	6	5	-16.7
t481	4	4	0.0	3	3	0.0	3	3	0.0
term1	18	14	-22.2	15	12	-20.0	13	10	-23.1
ttt2	18	16	-11.1	15	13	-13.3	13	12	-7.7
unreg	11	10	-9.1	10	9	-10.0	9	8	-11.1
vda	65	57	-12.3	56	47	-16.1	49	40	-18.4
x1	44	35	-20.5	38	28	-26.3	33	24	-27.3
x2	6	5	-16.7	6	4	-33.3	5	4	-20.0
x3	104	81	-22.1	89	68	-23.6	78	58	-25.6
x4	39	37	-5.1	34	31	-8.8	30	26	-13.3
z4ml	3	3	0.0	3	3	0.0	3	3	0.0
total	1520	1250	-17.8	1306	1033	-20.9	1141	884	-22.5

Table A.11: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, 6)			N_{sup} (4, 3, 7)			N_{sup} (4, 3, 8)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	24	24	0.0	21	21	0.0	19	19	0.0
C432	26	19	-26.9	24	17	-29.2	22	15	-31.8
C880	28	20	-28.6	25	18	-28.0	23	16	-30.4
alu2	37	27	-27.0	34	24	-29.4	31	21	-32.3
alu4	63	45	-28.6	57	40	-29.8	52	36	-30.8
apex6	66	48	-27.3	59	43	-27.1	54	39	-27.8
apex7	19	15	-21.1	17	14	-17.6	15	12	-20.0
b9	9	8	-11.1	9	7	-22.2	8	7	-12.5
c8	10	7	-30.0	9	7	-22.2	8	6	-25.0
cht	14	11	-21.4	12	10	-16.7	11	9	-18.2
cm150a	4	3	-25.0	4	2	-50.0	4	2	-50.0
cm151a	2	2	0.0	2	1	-50.0	2	1	-50.0
cm85a	4	3	-25.0	3	3	0.0	3	2	-33.3
cmb	4	3	-25.0	4	3	-25.0	3	3	0.0
count	13	8	-38.5	11	7	-36.4	10	7	-30.0
example2	22	19	-13.6	20	17	-15.0	18	15	-16.7
frg1	12	8	-33.3	11	7	-36.4	10	7	-30.0
frg2	71	55	-22.5	64	49	-23.4	58	44	-24.1
i1	4	3	-25.0	4	3	-25.0	3	3	0.0
i6	26	23	-11.5	24	21	-12.5	22	19	-13.6
i7	50	31	-38.0	45	28	-37.8	41	25	-39.0
i8	92	62	-32.6	83	55	-33.7	75	50	-33.3
i9	52	36	-30.8	47	32	-31.9	43	29	-32.6
k2	76	59	-22.4	68	53	-22.1	62	47	-24.2
my-adder	11	10	-9.1	10	9	-10.0	9	8	-11.1
parity	3	3	0.0	2	2	0.0	2	2	0.0
pcler	7	5	-28.6	6	5	-16.7	6	4	-33.3
pm1	4	3	-25.0	3	3	0.0	3	3	0.0
rot	50	41	-18.0	45	36	-20.0	41	33	-19.5
sct	5	5	0.0	5	4	-20.0	4	4	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	12	9	-25.0	11	8	-27.3	10	7	-30.0
ttt2	12	10	-16.7	11	9	-18.2	10	8	-20.0
unreg	8	7	-12.5	7	6	-14.3	6	5	-16.7
vda	43	34	-20.9	39	30	-23.1	36	27	-25.0
x1	30	21	-30.0	27	19	-29.6	24	17	-29.2
x2	4	3	-25.0	4	3	-25.0	4	3	-25.0
x3	69	51	-26.1	62	45	-27.4	57	41	-28.1
x4	26	23	-11.5	24	21	-12.5	22	19	-13.6
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	1016	768	-24.4	917	686	-25.2	835	619	-25.9

Table A.12: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (4, 3, 9)			N_{sup} (4, 3, 10)			N_{sup} (4, 3, $\frac{1}{10}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	17	17	0.0	16	16	0.0	15	15	0.0
C432	20	14	-30.0	18	13	-27.8	10	10	0.0
C880	21	15	-28.6	19	14	-26.3	11	11	0.0
alu2	28	19	-32.1	26	18	-30.8	12	12	0.0
alu4	47	33	-29.8	44	30	-31.8	21	21	0.0
apex6	49	35	-28.6	46	32	-30.4	22	22	0.0
apex7	14	11	-21.4	13	10	-23.1	7	7	0.0
b9	7	6	-14.3	7	6	-14.3	5	5	0.0
c8	7	5	-28.6	7	5	-28.6	4	4	0.0
cht	10	8	-20.0	10	7	-30.0	5	5	0.0
cm150a	3	2	-33.3	3	2	-33.3	1	1	0.0
cm151a	2	1	-50.0	2	1	-50.0	1	1	0.0
cm85a	3	2	-33.3	3	2	-33.3	2	2	0.0
cmb	3	2	-33.3	3	2	-33.3	2	2	0.0
count	10	6	-40.0	9	6	-33.3	5	5	0.0
example2	16	14	-12.5	15	13	-13.3	10	10	0.0
frg1	9	6	-33.3	8	5	-37.5	4	4	0.0
frg2	53	40	-24.5	49	37	-24.5	26	26	0.0
i1	3	3	0.0	3	2	-33.3	2	2	0.0
i6	20	17	-15.0	18	16	-11.1	11	11	0.0
i7	37	23	-37.8	34	21	-38.2	14	14	0.0
i8	69	45	-34.8	64	42	-34.4	27	27	0.0
i9	39	26	-33.3	36	24	-33.3	14	14	0.0
k2	57	43	-24.6	53	40	-24.5	31	31	0.0
my-adder	8	8	0.0	8	7	-12.5	6	6	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	5	4	-20.0	5	4	-20.0	3	3	0.0
pm1	3	3	0.0	3	2	-33.3	2	2	0.0
rot	38	30	-21.1	35	27	-22.9	21	21	0.0
sct	4	4	0.0	4	3	-25.0	3	3	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	9	7	-22.2	8	6	-25.0	4	4	0.0
ttt2	9	8	-11.1	8	7	-12.5	5	5	0.0
unreg	6	5	-16.7	6	5	-16.7	4	4	0.0
vda	33	25	-24.2	30	23	-23.3	18	18	0.0
x1	22	15	-31.8	21	14	-33.3	10	10	0.0
x2	3	3	0.0	3	2	-33.3	2	2	0.0
x3	52	37	-28.8	48	34	-29.2	22	22	0.0
x4	20	17	-15.0	18	16	-11.1	11	11	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	762	565	-25.9	709	520	-26.7	379	379	0.0

Table A.13: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, $\frac{1}{9}$)			N_{sup} (5, 2, $\frac{1}{8}$)			N_{sup} (5, 2, $\frac{1}{7}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	17	17	0.0	19	19	0.0	21	21	0.0
C432	11	11	0.0	12	12	0.0	13	13	0.0
C880	12	12	0.0	13	13	0.0	14	14	0.0
alu2	13	14	7.7	15	15	0.0	16	17	6.2
alu4	23	23	0.0	25	26	4.0	29	29	0.0
apex6	24	24	0.0	26	26	0.0	30	30	0.0
apex7	8	8	0.0	9	9	0.0	10	10	0.0
b9	5	5	0.0	5	6	20.0	6	6	0.0
c8	4	4	0.0	5	5	0.0	5	5	0.0
cht	5	5	0.0	6	6	0.0	6	6	0.0
cm150a	2	2	0.0	2	2	0.0	2	2	0.0
cm151a	1	1	0.0	1	1	0.0	1	1	0.0
cm85a	2	2	0.0	2	2	0.0	2	2	0.0
cmb	2	2	0.0	2	2	0.0	2	2	0.0
count	5	5	0.0	6	6	0.0	6	6	0.0
example2	11	11	0.0	12	12	0.0	14	14	0.0
frg1	4	4	0.0	4	4	0.0	5	5	0.0
frg2	29	29	0.0	32	32	0.0	36	36	0.0
i1	2	2	0.0	2	2	0.0	3	3	0.0
i6	12	12	0.0	13	13	0.0	15	15	0.0
i7	15	15	0.0	17	17	0.0	19	19	0.0
i8	30	29	-3.3	33	32	-3.0	38	36	-5.3
i9	15	15	0.0	17	17	0.0	19	19	0.0
k2	34	34	0.0	38	38	0.0	43	43	0.0
my-adder	7	7	0.0	8	8	0.0	8	8	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	4	4	0.0	4	4	0.0	4	4	0.0
pm1	2	2	0.0	2	2	0.0	2	2	0.0
rot	23	24	4.3	26	26	0.0	29	29	0.0
sct	3	3	0.0	3	3	0.0	4	4	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	5	5	0.0	5	5	0.0	6	6	0.0
ttt2	6	6	0.0	6	6	0.0	7	7	0.0
unreg	4	4	0.0	5	4	-20.0	5	5	0.0
vda	20	20	0.0	22	22	0.0	25	25	0.0
x1	11	11	0.0	12	12	0.0	13	13	0.0
x2	2	2	0.0	2	2	0.0	2	2	0.0
x3	24	24	0.0	27	27	0.0	30	30	0.0
x4	12	12	0.0	14	14	0.0	15	15	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	415	416	0.2	458	458	0.0	511	510	-0.2

Table A.14: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, $\frac{1}{6}$)			N_{sup} (5, 2, $\frac{1}{5}$)			N_{sup} (5, 2, $\frac{1}{4}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	24	24	0.0	28	28	0.0	33	33	0.0
C432	15	15	0.0	17	17	0.0	21	21	0.0
C880	16	16	0.0	19	19	0.0	23	23	0.0
alu2	19	19	0.0	22	22	0.0	26	27	3.8
alu4	33	33	0.0	38	38	0.0	45	46	2.2
apex6	34	34	0.0	39	39	0.0	47	47	0.0
apex7	11	11	0.0	13	13	0.0	15	15	0.0
b9	7	7	0.0	8	8	0.0	9	10	11.1
c8	6	6	0.0	7	7	0.0	8	8	0.0
cht	7	7	0.0	8	8	0.0	10	10	0.0
cm150a	2	2	0.0	2	2	0.0	3	3	0.0
cm151a	1	1	0.0	1	1	0.0	2	2	0.0
cm85a	2	2	0.0	2	2	0.0	3	3	0.0
cmb	3	3	0.0	3	3	0.0	4	4	0.0
count	7	7	0.0	8	8	0.0	10	10	0.0
example2	16	16	0.0	18	18	0.0	22	22	0.0
frg1	5	5	0.0	6	6	0.0	7	7	0.0
frg2	41	41	0.0	48	48	0.0	57	57	0.0
i1	3	3	0.0	3	3	0.0	4	4	0.0
i6	17	17	0.0	20	20	0.0	23	23	0.0
i7	22	22	0.0	26	26	0.0	32	31	-3.1
i8	44	42	-4.5	52	48	-7.7	65	58	-10.8
i9	22	22	0.0	27	26	-3.7	33	32	-3.0
k2	49	49	0.0	57	57	0.0	68	68	0.0
my-adder	10	10	0.0	11	11	0.0	13	13	0.0
parity	3	3	0.0	3	3	0.0	3	3	0.0
pcler	5	5	0.0	6	6	0.0	7	7	0.0
pm1	3	3	0.0	3	3	0.0	4	4	0.0
rot	33	33	0.0	39	39	0.0	46	47	2.2
sct	4	4	0.0	5	5	0.0	5	5	0.0
t481	2	2	0.0	2	2	0.0	3	3	0.0
term1	7	7	0.0	8	8	0.0	9	9	0.0
ttt2	8	8	0.0	9	9	0.0	11	11	0.0
unreg	6	5	-16.7	7	6	-14.3	9	7	-22.2
vda	28	28	0.0	33	33	0.0	39	39	0.0
x1	15	15	0.0	18	18	0.0	21	21	0.0
x2	3	3	0.0	3	3	0.0	4	4	0.0
x3	34	34	0.0	40	40	0.0	48	48	0.0
x4	18	18	0.0	21	20	-4.8	24	24	0.0
z4ml	2	2	0.0	2	2	0.0	3	3	0.0
total	587	584	-0.5	682	675	-1.0	819	812	-0.9

Table A.15: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, $\frac{1}{3}$)			N_{sup} (5, 2, $\frac{1}{2}$)			N_{sup} (5, 2, 1)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	41	41	0.0	55	55	0.0	82	82	0.0
C432	26	26	0.0	34	34	0.0	59	55	-6.8
C880	28	28	0.0	39	38	-2.6	72	63	-12.5
alu2	32	33	3.1	44	44	0.0	80	72	-10.0
alu4	57	57	0.0	75	76	1.3	138	124	-10.1
apex6	61	59	-3.3	88	80	-9.1	161	136	-15.5
apex7	19	19	0.0	26	26	0.0	48	42	-12.5
b9	12	12	0.0	15	16	6.7	24	24	0.0
c8	10	10	0.0	14	13	-7.1	25	23	-8.0
cht	13	12	-7.7	19	17	-10.5	35	30	-14.3
cm150a	4	4	0.0	5	5	0.0	10	9	-10.0
cm151a	2	2	0.0	3	3	0.0	5	5	0.0
cm85a	3	3	0.0	4	4	0.0	8	7	-12.5
cmb	5	4	-20.0	7	6	-14.3	12	10	-16.7
count	12	12	0.0	16	16	0.0	23	23	0.0
example2	27	27	0.0	36	36	0.0	64	57	-10.9
frg1	9	9	0.0	13	12	-7.7	24	21	-12.5
frg2	71	71	0.0	95	95	0.0	171	150	-12.3
i1	5	5	0.0	6	6	0.0	10	10	0.0
i6	29	29	0.0	39	39	0.0	67	64	-4.5
i7	42	40	-4.8	61	56	-8.2	112	95	-15.2
i8	85	75	-11.8	123	105	-14.6	225	181	-19.6
i9	43	41	-4.7	63	59	-6.3	123	106	-13.8
k2	85	85	0.0	113	113	0.0	207	187	-9.7
my-adder	16	16	0.0	22	22	0.0	32	32	0.0
parity	4	4	0.0	5	5	0.0	8	8	0.0
pcler	8	8	0.0	11	11	0.0	16	16	0.0
pm1	4	4	0.0	6	6	0.0	10	9	-10.0
rot	58	58	0.0	77	77	0.0	135	124	-8.1
sct	7	7	0.0	9	9	0.0	15	14	-6.7
t481	3	3	0.0	4	4	0.0	6	6	0.0
term1	11	11	0.0	15	15	0.0	28	24	-14.3
ttt2	14	14	0.0	18	18	0.0	33	30	-9.1
unreg	11	9	-18.2	16	12	-25.0	29	18	-37.9
vda	49	49	0.0	65	65	0.0	114	105	-7.9
x1	26	26	0.0	35	35	0.0	63	57	-9.5
x2	4	4	0.0	6	6	0.0	10	9	-10.0
x3	62	61	-1.6	89	85	-4.5	163	145	-11.0
x4	30	30	0.0	43	40	-7.0	78	63	-19.2
z4ml	3	3	0.0	4	4	0.0	6	6	0.0
total	1031	1011	-1.9	1418	1368	-3.5	2531	2242	-11.4

Table A.16: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, 2)			N_{sup} (5, 2, 3)			N_{sup} (5, 2, 4)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	55	55	0.0	41	41	0.0	33	33	0.0
C432	51	42	-17.6	45	36	-20.0	42	32	-23.8
C880	61	49	-19.7	54	42	-22.2	48	37	-22.9
alu2	72	58	-19.4	68	50	-26.5	64	44	-31.2
alu4	118	99	-16.1	111	84	-24.3	105	74	-29.5
apex6	140	109	-22.1	132	91	-31.1	124	79	-36.3
apex7	41	34	-17.1	39	28	-28.2	37	24	-35.1
b9	21	18	-14.3	18	15	-16.7	16	13	-18.8
c8	21	18	-14.3	19	15	-21.1	17	13	-23.5
cht	33	24	-27.3	31	20	-35.5	29	17	-41.4
cm150a	9	7	-22.2	9	6	-33.3	8	5	-37.5
cm151a	5	4	-20.0	5	3	-40.0	4	3	-25.0
cm85a	7	6	-14.3	7	5	-28.6	6	4	-33.3
cmb	10	7	-30.0	9	6	-33.3	8	5	-37.5
count	16	16	0.0	16	14	-12.5	15	13	-13.3
example2	55	45	-18.2	48	38	-20.8	43	32	-25.6
frg1	22	18	-18.2	21	15	-28.6	21	13	-38.1
frg2	148	120	-18.9	139	100	-28.1	131	86	-34.4
i1	9	7	-22.2	8	6	-25.0	7	5	-28.6
i6	65	55	-15.4	63	48	-23.8	61	43	-29.5
i7	100	76	-24.0	93	64	-31.2	88	57	-35.2
i8	198	149	-24.7	186	128	-31.2	175	112	-36.0
i9	119	88	-26.1	115	76	-33.9	111	66	-40.5
k2	177	141	-20.3	155	116	-25.2	138	99	-28.3
my-adder	28	24	-14.3	24	20	-16.7	22	16	-27.3
parity	5	5	0.0	4	4	0.0	3	3	0.0
pcler	13	12	-7.7	11	9	-18.2	10	8	-20.0
pm1	8	7	-12.5	7	6	-14.3	7	5	-28.6
rot	116	94	-19.0	102	79	-22.5	90	68	-24.4
sct	13	10	-23.1	12	9	-25.0	10	8	-20.0
t481	4	4	0.0	3	3	0.0	3	3	0.0
term1	24	19	-20.8	22	16	-27.3	21	14	-33.3
ttt2	29	22	-24.1	25	19	-24.0	23	16	-30.4
unreg	25	14	-44.0	22	12	-45.5	19	11	-42.1
vda	97	79	-18.6	85	65	-23.5	76	56	-26.3
x1	56	46	-17.9	53	39	-26.4	50	34	-32.0
x2	8	7	-12.5	7	6	-14.3	7	6	-14.3
x3	150	116	-22.7	141	97	-31.2	133	83	-37.6
x4	67	48	-28.4	59	40	-32.2	52	35	-32.7
z4ml	6	5	-16.7	5	4	-20.0	4	3	-25.0
total	2202	1757	-20.2	2014	1475	-26.8	1861	1278	-31.3

Table A.17: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, 5)			N_{sup} (5, 2, 6)			N_{sup} (5, 2, 7)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	28	28	0.0	24	24	0.0	21	21	0.0
C432	40	28	-30.0	38	25	-34.2	37	23	-37.8
C880	46	33	-28.3	43	30	-30.2	41	27	-34.1
alu2	61	39	-36.1	60	35	-41.7	58	32	-44.8
alu4	99	66	-33.3	94	59	-37.2	91	54	-40.7
apex6	117	70	-40.2	111	63	-43.2	105	57	-45.7
apex7	35	22	-37.1	33	20	-39.4	31	18	-41.9
b9	15	11	-26.7	14	10	-28.6	14	9	-35.7
c8	17	12	-29.4	16	10	-37.5	15	10	-33.3
cht	28	15	-46.4	26	14	-46.2	25	12	-52.0
cm150a	8	5	-37.5	7	4	-42.9	7	4	-42.9
cm151a	4	3	-25.0	4	2	-50.0	4	2	-50.0
cm85a	6	4	-33.3	6	3	-50.0	5	3	-40.0
cmb	7	5	-28.6	7	4	-42.9	6	4	-33.3
count	15	11	-26.7	15	10	-33.3	14	10	-28.6
example2	39	28	-28.2	36	25	-30.6	34	23	-32.4
frg1	20	12	-40.0	20	11	-45.0	19	10	-47.4
frg2	124	76	-38.7	117	68	-41.9	111	62	-44.1
i1	6	5	-16.7	6	4	-33.3	5	4	-20.0
i6	59	39	-33.9	57	35	-38.6	56	32	-42.9
i7	83	52	-37.3	79	47	-40.5	75	43	-42.7
i8	165	100	-39.4	156	90	-42.3	149	82	-45.0
i9	108	59	-45.4	105	53	-49.5	102	48	-52.9
k2	128	88	-31.2	122	79	-35.2	116	72	-37.9
my-adder	20	14	-30.0	18	13	-27.8	16	12	-25.0
parity	3	3	0.0	3	3	0.0	2	2	0.0
pcler	9	7	-22.2	8	7	-12.5	8	6	-25.0
pm1	6	4	-33.3	6	4	-33.3	6	4	-33.3
rot	85	59	-30.6	81	53	-34.6	77	49	-36.4
sct	9	7	-22.2	9	6	-33.3	8	5	-37.5
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	20	13	-35.0	19	12	-36.8	18	11	-38.9
ttt2	22	14	-36.4	21	13	-38.1	20	12	-40.0
unreg	17	10	-41.2	16	9	-43.8	15	8	-46.7
vda	71	49	-31.0	67	44	-34.3	64	40	-37.5
x1	47	31	-34.0	45	28	-37.8	42	25	-40.5
x2	7	5	-28.6	6	5	-16.7	6	4	-33.3
x3	125	74	-40.8	119	67	-43.7	113	61	-46.0
x4	49	32	-34.7	46	28	-39.1	44	26	-40.9
z4ml	4	3	-25.0	4	3	-25.0	3	3	0.0
total	1754	1138	-35.1	1666	1022	-38.7	1585	932	-41.2

Table A.18: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (5, 2, 8)			N_{sup} (5, 2, 9)			N_{sup} (5, 2, 10)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	19	19	0.0	17	17	0.0	15	15	0.0
C432	36	21	-41.7	35	20	-42.9	35	18	-48.6
C880	39	25	-35.9	38	23	-39.5	37	21	-43.2
alu2	56	29	-48.2	55	27	-50.9	54	25	-53.7
alu4	89	49	-44.9	86	46	-46.5	84	42	-50.0
apex6	100	53	-47.0	96	49	-49.0	92	45	-51.1
apex7	30	16	-46.7	28	15	-46.4	27	14	-48.1
b9	13	8	-38.5	12	8	-33.3	12	7	-41.7
c8	14	9	-35.7	14	8	-42.9	13	8	-38.5
cht	24	11	-54.2	23	10	-56.5	22	10	-54.5
cm150a	7	4	-42.9	6	4	-33.3	6	3	-50.0
cm151a	4	2	-50.0	3	2	-33.3	3	2	-33.3
cm85a	5	3	-40.0	5	3	-40.0	5	3	-40.0
cmb	6	4	-33.3	6	3	-50.0	5	3	-40.0
count	14	9	-35.7	13	8	-38.5	13	8	-38.5
example2	32	21	-34.4	31	20	-35.5	30	18	-40.0
frg1	18	9	-50.0	18	8	-55.6	18	8	-55.6
frg2	106	57	-46.2	101	53	-47.5	97	49	-49.5
i1	5	3	-40.0	5	3	-40.0	5	3	-40.0
i6	54	30	-44.4	53	28	-47.2	52	26	-50.0
i7	71	40	-43.7	68	37	-45.6	66	35	-47.0
i8	142	75	-47.2	137	69	-49.6	133	64	-51.9
i9	99	44	-55.6	97	41	-57.7	94	38	-59.6
k2	110	66	-40.0	105	61	-41.9	101	57	-43.6
my-adder	16	11	-31.2	15	10	-33.3	14	9	-35.7
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	8	6	-25.0	8	5	-37.5	8	5	-37.5
pm1	5	3	-40.0	5	3	-40.0	5	3	-40.0
rot	73	45	-38.4	70	41	-41.4	67	38	-43.3
sct	8	5	-37.5	7	5	-28.6	7	4	-42.9
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	17	10	-41.2	16	9	-43.8	16	8	-50.0
ttt2	19	11	-42.1	18	10	-44.4	17	9	-47.1
unreg	15	7	-53.3	14	7	-50.0	13	6	-53.8
vda	61	37	-39.3	58	34	-41.4	55	32	-41.8
x1	41	23	-43.9	40	21	-47.5	39	20	-48.7
x2	6	4	-33.3	6	4	-33.3	6	3	-50.0
x3	108	56	-48.1	103	52	-49.5	98	48	-51.0
x4	42	24	-42.9	40	22	-45.0	38	20	-47.4
z4ml	3	2	-33.3	3	2	-33.3	3	2	-33.3
total	1519	855	-43.7	1459	792	-45.7	1409	733	-48.0

Table A.19: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, $\frac{1}{10}$)			N_{sup} (6, 4, $\frac{1}{9}$)			N_{sup} (6, 4, $\frac{1}{8}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	15	15	0.0	17	17	0.0	19	19	0.0
C432	9	9	0.0	10	10	0.0	11	11	0.0
C880	10	10	0.0	11	10	-9.1	12	12	0.0
alu2	11	11	0.0	12	12	0.0	13	13	0.0
alu4	19	19	0.0	21	21	0.0	23	23	0.0
apex6	19	19	0.0	21	21	0.0	23	23	0.0
apex7	7	7	0.0	7	7	0.0	8	8	0.0
b9	4	4	0.0	4	4	0.0	5	5	0.0
c8	4	4	0.0	4	4	0.0	5	5	0.0
cht	5	5	0.0	5	5	0.0	6	6	0.0
cm150a	1	1	0.0	1	1	0.0	1	1	0.0
cm151a	1	1	0.0	1	1	0.0	1	1	0.0
cm85a	2	2	0.0	2	2	0.0	2	2	0.0
cmb	2	2	0.0	2	2	0.0	2	2	0.0
count	3	3	0.0	4	4	0.0	4	4	0.0
example2	10	10	0.0	11	11	0.0	12	12	0.0
frg1	3	3	0.0	3	3	0.0	4	4	0.0
frg2	24	24	0.0	26	26	0.0	29	29	0.0
i1	2	2	0.0	2	2	0.0	2	2	0.0
i6	8	7	-12.5	8	8	0.0	9	9	0.0
i7	14	14	0.0	15	15	0.0	16	16	0.0
i8	23	23	0.0	26	25	-3.8	28	28	0.0
i9	13	13	0.0	15	15	0.0	16	16	0.0
k2	29	29	0.0	32	32	0.0	35	35	0.0
my-adder	6	6	0.0	7	7	0.0	8	8	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	3	3	0.0	3	3	0.0	4	4	0.0
pm1	2	2	0.0	2	2	0.0	2	2	0.0
rot	20	20	0.0	22	22	0.0	24	24	0.0
sct	3	3	0.0	3	3	0.0	3	3	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	4	4	0.0	4	4	0.0	5	5	0.0
ttt2	5	5	0.0	5	5	0.0	6	6	0.0
unreg	2	2	0.0	2	2	0.0	2	2	0.0
vda	17	17	0.0	19	19	0.0	21	21	0.0
x1	9	9	0.0	10	10	0.0	11	11	0.0
x2	2	2	0.0	2	2	0.0	2	2	0.0
x3	21	21	0.0	23	23	0.0	26	26	0.0
x4	9	9	0.0	10	10	0.0	11	11	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	347	346	-0.3	378	376	-0.5	417	417	0.0

Table A.20: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, $\frac{1}{7}$)			N_{sup} (6, 4, $\frac{1}{6}$)			N_{sup} (6, 4, $\frac{1}{5}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	21	21	0.0	24	24	0.0	28	28	0.0
C432	12	12	0.0	14	14	0.0	16	16	0.0
C880	13	13	0.0	15	15	0.0	17	17	0.0
alu2	14	14	0.0	16	16	0.0	19	19	0.0
alu4	26	26	0.0	29	29	0.0	34	34	0.0
apex6	26	26	0.0	30	30	0.0	35	35	0.0
apex7	9	9	0.0	10	10	0.0	12	12	0.0
b9	5	5	0.0	6	6	0.0	7	7	0.0
c8	5	5	0.0	6	6	0.0	7	7	0.0
cht	6	6	0.0	7	7	0.0	8	8	0.0
cm150a	1	1	0.0	1	1	0.0	1	1	0.0
cm151a	1	1	0.0	1	1	0.0	1	1	0.0
cm85a	2	2	0.0	2	2	0.0	2	2	0.0
cmb	2	2	0.0	2	2	0.0	3	3	0.0
count	4	4	0.0	5	5	0.0	6	6	0.0
example2	13	13	0.0	15	15	0.0	17	17	0.0
frg1	4	4	0.0	5	5	0.0	5	5	0.0
frg2	32	32	0.0	37	37	0.0	43	43	0.0
i1	3	3	0.0	3	3	0.0	3	3	0.0
i6	10	10	0.0	12	12	0.0	14	14	0.0
i7	18	18	0.0	21	21	0.0	24	24	0.0
i8	32	32	0.0	36	36	0.0	42	42	0.0
i9	18	18	0.0	21	21	0.0	24	24	0.0
k2	39	39	0.0	45	45	0.0	52	52	0.0
my-adder	8	8	0.0	10	10	0.0	11	11	0.0
parity	2	2	0.0	3	3	0.0	3	3	0.0
pcler	4	4	0.0	4	4	0.0	5	5	0.0
pm1	2	2	0.0	3	3	0.0	3	3	0.0
rot	27	27	0.0	31	31	0.0	36	36	0.0
sct	3	3	0.0	4	4	0.0	4	4	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	5	5	0.0	6	6	0.0	7	7	0.0
ttt2	6	6	0.0	7	7	0.0	8	8	0.0
unreg	3	3	0.0	3	3	0.0	4	4	0.0
vda	23	23	0.0	26	26	0.0	31	31	0.0
x1	12	12	0.0	14	14	0.0	16	16	0.0
x2	2	2	0.0	2	2	0.0	3	3	0.0
x3	29	29	0.0	33	33	0.0	38	38	0.0
x4	13	13	0.0	14	14	0.0	17	17	0.0
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	459	459	0.0	527	527	0.0	610	610	0.0

Table A.21: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, $\frac{1}{4}$)			N_{sup} (6, 4, $\frac{1}{3}$)			N_{sup} (6, 4, $\frac{1}{2}$)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	33	33	0.0	41	41	0.0	55	55	0.0
C432	19	19	0.0	24	24	0.0	32	32	0.0
C880	20	20	0.0	25	25	0.0	34	34	0.0
alu2	23	23	0.0	28	28	0.0	38	38	0.0
alu4	41	41	0.0	51	51	0.0	67	67	0.0
apex6	42	42	0.0	52	52	0.0	69	69	0.0
apex7	14	14	0.0	18	18	0.0	23	23	0.0
b9	8	8	0.0	10	10	0.0	14	14	0.0
c8	8	8	0.0	10	10	0.0	13	13	0.0
cht	10	10	0.0	12	12	0.0	16	16	0.0
cm150a	2	2	0.0	2	2	0.0	3	3	0.0
cm151a	1	1	0.0	1	1	0.0	2	2	0.0
cm85a	3	3	0.0	3	3	0.0	4	4	0.0
cmb	3	3	0.0	4	4	0.0	5	5	0.0
count	7	7	0.0	8	8	0.0	11	11	0.0
example2	21	21	0.0	26	26	0.0	34	34	0.0
frg1	6	6	0.0	8	8	0.0	10	10	0.0
frg2	51	51	0.0	64	64	0.0	85	85	0.0
i1	4	4	0.0	5	5	0.0	6	6	0.0
i6	17	16	-5.9	22	21	-4.5	31	29	-6.5
i7	29	29	0.0	36	36	0.0	48	48	0.0
i8	50	50	0.0	63	63	0.0	84	84	0.0
i9	29	29	0.0	36	36	0.0	48	48	0.0
k2	63	63	0.0	78	78	0.0	104	104	0.0
my-adder	13	13	0.0	16	16	0.0	22	22	0.0
parity	3	3	0.0	4	4	0.0	5	5	0.0
pcler	6	6	0.0	7	7	0.0	10	10	0.0
pm1	4	4	0.0	4	4	0.0	6	6	0.0
rot	43	43	0.0	54	54	0.0	71	71	0.0
sct	5	5	0.0	6	6	0.0	8	8	0.0
t481	3	3	0.0	3	3	0.0	4	4	0.0
term1	8	8	0.0	10	10	0.0	13	13	0.0
ttt2	10	10	0.0	12	12	0.0	16	16	0.0
unreg	4	4	0.0	6	5	-16.7	8	7	-12.5
vda	37	37	0.0	46	46	0.0	61	61	0.0
x1	19	19	0.0	24	24	0.0	31	31	0.0
x2	3	3	0.0	4	4	0.0	5	5	0.0
x3	46	46	0.0	57	57	0.0	76	76	0.0
x4	20	20	0.0	25	25	0.0	33	33	0.0
z4ml	3	3	0.0	3	3	0.0	4	4	0.0
total	731	730	-0.1	908	906	-0.2	1209	1206	-0.2

Table A.22: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, 1)			N_{sup} (6, 4, 2)			N_{sup} (6, 4, 3)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	82	82	0.0	55	55	0.0	41	41	0.0
C432	47	47	0.0	33	32	-3.0	27	24	-11.1
C880	50	50	0.0	35	34	-2.9	29	26	-10.3
alu2	57	56	-1.8	46	40	-13.0	41	31	-24.4
alu4	101	101	0.0	73	68	-6.8	66	53	-19.7
apex6	103	103	0.0	69	69	0.0	57	53	-7.0
apex7	35	35	0.0	23	23	0.0	19	18	-5.3
b9	20	20	0.0	14	14	0.0	11	10	-9.1
c8	19	19	0.0	13	13	0.0	10	10	0.0
cht	23	23	0.0	16	16	0.0	12	12	0.0
cm150a	5	4	-20.0	4	3	-25.0	4	3	-25.0
cm151a	2	2	0.0	2	2	0.0	2	2	0.0
cm85a	6	6	0.0	4	4	0.0	3	3	0.0
cmb	7	7	0.0	5	5	0.0	4	4	0.0
count	16	16	0.0	14	12	-14.3	13	10	-23.1
example2	51	51	0.0	34	34	0.0	26	26	0.0
frg1	16	16	0.0	14	12	-14.3	12	9	-25.0
frg2	128	128	0.0	85	85	0.0	67	65	-3.0
i1	9	9	0.0	6	6	0.0	5	5	0.0
i6	53	48	-9.4	43	36	-16.3	37	29	-21.6
i7	72	72	0.0	56	48	-14.3	46	37	-19.6
i8	146	126	-13.7	117	91	-22.2	98	73	-25.5
i9	71	72	1.4	61	51	-16.4	55	41	-25.5
k2	156	156	0.0	104	104	0.0	79	78	-1.3
my-adder	32	32	0.0	22	22	0.0	16	16	0.0
parity	8	8	0.0	5	5	0.0	4	4	0.0
pcler	14	14	0.0	10	10	0.0	8	7	-12.5
pm1	8	8	0.0	6	6	0.0	4	4	0.0
rot	107	107	0.0	71	71	0.0	54	54	0.0
sct	12	12	0.0	8	8	0.0	6	6	0.0
t481	6	6	0.0	4	4	0.0	3	3	0.0
term1	19	19	0.0	13	13	0.0	11	10	-9.1
ttt2	24	24	0.0	16	16	0.0	13	12	-7.7
unreg	15	12	-20.0	13	9	-30.8	12	7	-41.7
vda	91	91	0.0	61	61	0.0	46	46	0.0
x1	47	47	0.0	34	32	-5.9	29	25	-13.8
x2	7	7	0.0	5	5	0.0	5	4	-20.0
x3	114	114	0.0	76	76	0.0	59	58	-1.7
x4	49	49	0.0	36	35	-2.8	30	28	-6.7
z4ml	6	6	0.0	4	4	0.0	3	3	0.0
total	1834	1805	-1.6	1310	1234	-5.8	1067	950	-11.0

Table A.23: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, 4)			N_{sup} (6, 4, 5)			N_{sup} (6, 4, 6)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	33	33	0.0	28	28	0.0	24	24	0.0
C432	25	20	-20.0	23	17	-26.1	21	15	-28.6
C880	26	21	-19.2	24	18	-25.0	22	16	-27.3
alu2	37	26	-29.7	34	22	-35.3	31	19	-38.7
alu4	60	44	-26.7	55	38	-30.9	51	33	-35.3
apex6	49	44	-10.2	43	37	-14.0	39	32	-17.9
apex7	16	15	-6.2	14	13	-7.1	13	11	-15.4
b9	9	9	0.0	8	7	-12.5	8	7	-12.5
c8	9	8	-11.1	8	7	-12.5	8	6	-25.0
cht	10	10	0.0	8	8	0.0	7	7	0.0
cm150a	4	2	-50.0	3	2	-33.3	3	2	-33.3
cm151a	2	2	0.0	2	1	-50.0	2	1	-50.0
cm85a	3	3	0.0	3	3	0.0	2	2	0.0
cmb	3	3	0.0	3	3	0.0	3	3	0.0
count	12	8	-33.3	11	7	-36.4	10	6	-40.0
example2	22	21	-4.5	19	18	-5.3	17	16	-5.9
frg1	11	8	-27.3	10	7	-30.0	10	6	-40.0
frg2	58	54	-6.9	53	46	-13.2	49	41	-16.3
i1	4	4	0.0	3	3	0.0	3	3	0.0
i6	33	24	-27.3	31	21	-32.3	28	18	-35.7
i7	40	31	-22.5	36	26	-27.8	33	23	-30.3
i8	84	61	-27.4	74	52	-29.7	69	46	-33.3
i9	50	34	-32.0	46	29	-37.0	42	26	-38.1
k2	68	64	-5.9	62	54	-12.9	57	48	-15.8
my-adder	13	13	0.0	11	11	0.0	10	10	0.0
parity	3	3	0.0	3	3	0.0	3	3	0.0
pcler	7	6	-14.3	6	5	-16.7	6	4	-33.3
pm1	4	4	0.0	3	3	0.0	3	3	0.0
rot	45	43	-4.4	39	37	-5.1	35	32	-8.6
sct	5	5	0.0	4	4	0.0	4	4	0.0
t481	3	3	0.0	2	2	0.0	2	2	0.0
term1	9	8	-11.1	9	7	-22.2	8	6	-25.0
ttt2	11	10	-9.1	10	9	-10.0	9	8	-11.1
unreg	11	6	-45.5	10	5	-50.0	9	5	-44.4
vda	40	37	-7.5	35	32	-8.6	32	27	-15.6
x1	27	20	-25.9	25	18	-28.0	23	15	-34.8
x2	4	3	-25.0	4	3	-25.0	4	3	-25.0
x3	51	48	-5.9	44	41	-6.8	40	36	-10.0
x4	26	23	-11.5	22	20	-9.1	21	18	-14.3
z4ml	3	3	0.0	2	2	0.0	2	2	0.0
total	930	784	-15.7	830	669	-19.4	763	589	-22.8

Table A.24: Comparison of Hetero-Mapper & Post-Process-Homo

Circuit	N_{sup} (6, 4, 7)			N_{sup} (6, 4, 8)			N_{sup} (6, 4, 9)		
Names	PPH	Het	Dif %	PPH	Het	Dif %	PPH	Het	Dif %
C1355	21	21	0.0	19	19	0.0	17	17	0.0
C432	20	13	-35.0	18	12	-33.3	17	11	-35.3
C880	20	14	-30.0	19	13	-31.6	18	12	-33.3
alu2	29	17	-41.4	27	16	-40.7	26	14	-46.2
alu4	47	29	-38.3	44	26	-40.9	41	24	-41.5
apex6	36	29	-19.4	34	26	-23.5	32	23	-28.1
apex7	12	10	-16.7	11	9	-18.2	10	8	-20.0
b9	7	6	-14.3	7	5	-28.6	6	5	-16.7
c8	7	5	-28.6	7	5	-28.6	6	5	-16.7
cht	6	6	0.0	6	6	0.0	5	5	0.0
cm150a	3	2	-33.3	3	2	-33.3	3	2	-33.3
cm151a	2	1	-50.0	2	1	-50.0	1	1	0.0
cm85a	2	2	0.0	2	2	0.0	2	2	0.0
cmb	3	2	-33.3	3	2	-33.3	2	2	0.0
count	9	6	-33.3	9	5	-44.4	8	5	-37.5
example2	15	14	-6.7	14	13	-7.1	13	12	-7.7
frg1	9	5	-44.4	8	5	-37.5	8	4	-50.0
frg2	46	36	-21.7	43	33	-23.3	40	30	-25.0
i1	3	3	0.0	3	2	-33.3	2	2	0.0
i6	26	16	-38.5	25	15	-40.0	23	14	-39.1
i7	31	21	-32.3	29	19	-34.5	27	17	-37.0
i8	64	41	-35.9	60	37	-38.3	56	33	-41.1
i9	39	23	-41.0	37	21	-43.2	34	19	-44.1
k2	53	42	-20.8	50	38	-24.0	47	35	-25.5
my-adder	8	8	0.0	8	8	0.0	7	7	0.0
parity	2	2	0.0	2	2	0.0	2	2	0.0
pcler	6	4	-33.3	5	4	-20.0	5	3	-40.0
pm1	2	2	0.0	2	2	0.0	2	2	0.0
rot	32	28	-12.5	29	26	-10.3	28	23	-17.9
sct	3	3	0.0	3	3	0.0	3	3	0.0
t481	2	2	0.0	2	2	0.0	2	2	0.0
term1	7	6	-14.3	7	5	-28.6	7	5	-28.6
ttt2	8	7	-12.5	8	6	-25.0	7	6	-14.3
unreg	9	4	-55.6	8	4	-50.0	8	4	-50.0
vda	29	24	-17.2	28	22	-21.4	26	20	-23.1
x1	21	14	-33.3	20	12	-40.0	19	11	-42.1
x2	3	2	-33.3	3	2	-33.3	3	2	-33.3
x3	36	32	-11.1	32	29	-9.4	30	26	-13.3
x4	19	16	-15.8	18	14	-22.2	17	13	-23.5
z4ml	2	2	0.0	2	2	0.0	2	2	0.0
total	699	520	-25.6	657	475	-27.7	612	433	-29.2

Table A.25: Comparison of Hetero-Mapper & Post-Process-Homo