LEBANESE AMERICAN UNIVERSITY

ENHANCED TECHNOLOGY MAPPING FOR FPGAs WITH EXPLORATION OF CELL CONFIGURATIONS

By

GRACE JOSEPH ZGHEIB

A thesis
Submitted in partial fulfillment of the requirements
for the Degree of Master of Science in Computer Engineering

School of Engineering
August 2011

LEBANESE AMERICAN UNIVERSITY

Thesis approval Form (Annex III)

Student Name: Grace Zgheib I.D. #: 200401077

Thesis Title:

Enhanced Mapping Technique for FPGAs with Exploration of Cell Configurations

Program : Masters of Science in Computer Engineering

Division/Dept: Electrical and Computer Engineering

School: School of Engineering

Approved by:

Signatures Redacted

Dr. Iyad Ouaiss

Signatures Redacted

Dr. Zahi Nakad

Signatures Redacted

Dr. Wissam Fawaz

Date: 02/08/11

THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY

By signing and submitting this license, I (the author(s) or copyright owner) grant the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute my submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. I agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. I also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. I represent that the submission is my original work, and that I have the right to grant the rights contained in this license. I also represent that my submission does not, to the best of my knowledge, infringe upon anyone's copyright. If the submission contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, I REPRESENT THAT I HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify my name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to my submission.

Name: Grace Zgheib

Signature: Signatures Redacted

Date: August 2, 2011

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that we have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Grace Zgheib

Signature Signatures Redacted

Date: August 2, 2011

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor Dr. Iyad Ouaiss who kept a watchful eye on my work progress and helped me find the path within the un-explored fields of research through the fogs of uncertainty and confusion that any inexperienced researcher faces.

I am also thankful for Dr. Zahi Nakad and Dr. Wissam Fawaz who will be assessing this work as members in my thesis committee.

But above all, I owe this work and this achievement to those who stood by me throughout the unstable path of a student/researcher's life…to my friends and family!

Enhanced Technology Mapping for FPGAs with Exploration of Cell Configurations

Grace Joseph Zgheib

Abstract

In the state of the art Field-Programmable Gate Arrays (FPGAs), logic circuits are synthesized and mapped on clusters of look-up tables. However, when additions need to be performed, an adder along with a carry-chain is used to ensure a fast execution of such an arithmetic operation. This carry-chain is a dedicated wire available in the architecture of the FPGA and is as such independent of the external programmable routing resources.

The proposed idea introduces variable-structure Boolean matching as well as decomposition of mapped functions in order to take advantage of the carry-chains when they are not used for addition operations. Previously synthesized and mapped logic functions are adapted so that their outputs are routed using the dedicated carry-chains instead of the external programmable interconnects. Mapping onto these chains yields a reduction in the overall external routing resources as well as the general routing congestion.

Moreover, a generic software platform was developed allowing users to identify and test various basic-unit structures and compare their performances on particular logic circuits depending on criteria specified by the user. Such structures may vary from currently available FPGA architectures to customized theoretical structures well-suited for a specific design(s). This tool can also propose particular cell structures to map logic circuits while respecting the user's constraints and insuring the optimization of specific parameters.

Keywords: Field Programmable Gate Array, Technology Mapping, Boolean Matching, Decomposition, Carry Chain, Cell Configuration.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Chapter One

Introduction

Field Programmable Gate Arrays (FPGAs) are known to have an advantage over Application-Specific Integrated Circuits (ASICs) with their ability to implement a wider range of applications and designs. Such flexibility is achieved through the presence of a large set of programmable interconnects that ensures the connectivity of all FPGA cells. However, ensuring such full grid connectivity comes at a compromise of speed, power consumption and size, as well as the density of the FPGA's logic cells. On one hand, programmable interconnects introduce delays slowing down critical paths while being the main contributors in the total static and dynamic power dissipation. On the other hand, the majority of the FPGA's area is dedicated to its routing resources, which reduces the amount of logic components that can fit on that FPGA. As such, increasing the FPGA's size is constrained due to the area consumed by its programmable routing connections [1], [2]. It is also frequent to face scenarios where a design that does not use all the logic cells of a particular FPGA fails to be implemented due to lack of available routing resources.

As such, reducing a design's utilization of routing resources became a necessity in large circuits, small devices or even power constrained applications. As such, compilation

tools try to optimize the use of interconnects through various mapping, placement and routing algorithms.

Moreover, minor modifications on the logic cells of the current main FPGAs, such as Altera and Xilinx FPGAs, might introduce some important reductions in the power consumption, used routing resources, delay, etc. Exploring such possible modifications in the current state-of-the-art FPGAs requires the presence of an analytical tool that would perform the necessary computations. Such a tool is needed to map main design circuits on some modified FPGA cell structures while reporting relevant statistics and comparative information.

Furthermore, designers are limited with the compilation tools of current FPGA companies which support their manufactured -in the market- FPGAs. However, designers wishing to perform modifications to the internal architecture of the FPGAs' logic cells cannot use such compilation tools to explore the benefits of the new architecture on their particular applications. So, such designers find themselves forced to implement their own tool trying to simulate the changes in the architecture and their respective effects; or to compromise and implement their application on one of the available architectures at possible costs of area, power, delay, etc.

This proposed work comes as a solution to these issues by presenting the designers with such an analytical tool that can be customized depending on the user's requirements. It also proposes an enhanced mapping technique that would use some available dedicated routing resources to route FPGA cells while reducing the used external routing resources, and as such their congestion.

The presentation of this work starts by covering the related research in this field, as well as any required material for a further understanding of the proposed approaches, in a literature review presented in Chapter 2. Then Chapter 3 explains the proposed and used approaches such as the Boolean Matching technique and the flexible-structure Boolean Matching technique used to map on general structures and particularly on carry-chains of Altera Stratix FPGAs. It also proposes some decomposition techniques used to enhance whichever used mapping approach. Furthermore, an elaboration and clear explanation of an enhanced mapping technique that uses the previously proposed approaches and aims on the reduction of routing congestion is proposed in Chapter 4. The analytical tool that allows designers to explore modified and new logic cell structures is presented in Chapter 5. Then, in order to properly highlight the efficiency and benefits of these approaches, Chapter 6 summarizes the major experiments conducted and their respective results. Finally, Chapter 7 provides general conclusions on the overall work and results, while it also lists possible future work that would improve the current techniques.

Chapter Two

Literature Review

2.1 Structure of the Altera Stratix FPGAs

The latest Altera Stratix II, III and IV Field Programmable Gate Arrays (FPGAs) are composed of Logic Array Blocks (LABs) where each LAB is built using several Adaptive Logic Modules (ALMs) [3], [4], [5]. These ALMs are designed using Look-Up Tables (LUTs), multiplexers, adders and registers. Each ALM has 8 inputs and two outputs. Its LUTs can be distributed to form two Adaptive LUTs (ALUTs) where each ALUT can independently implement a logic function and drive it on one of the outputs. The overall ALM can implement any up-to-6-input functions and a subset of 7-input functions as well.

ALMs can operate in four different modes: Normal mode, Extended-LUT mode, Arithmetic mode and Shared-Arithmetic mode. The main modes of interest are the Normal mode and the Arithmetic mode.

When in Normal mode, the ALM is used to implement combinational logic functions using the LUTs. However, the ALM’s Arithmetic mode is used whenever arithmetic

operations need to be performed (such as additions), so it uses the dedicated adders along with the carry-chain as shown in Figure 1.

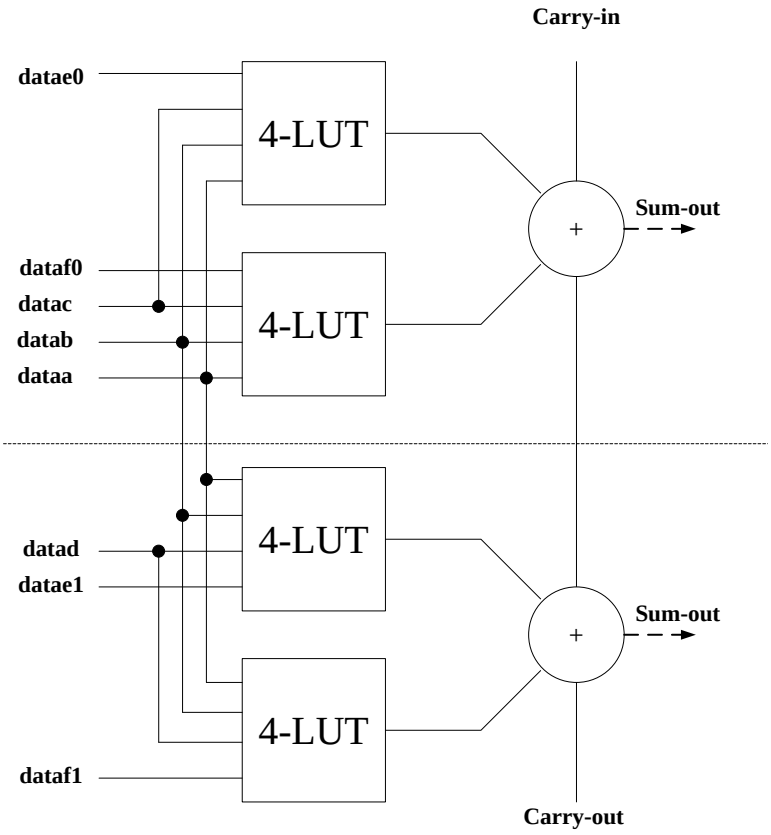

Figure 1: Adaptive Logic Module (ALM) in Arithmetic mode

The used carry-chain is a dedicated wire whose main role is to pass the carry-out of a previous addition as a carry-in to the current addition without major delays [6]. Using this wire allows a fast propagation of the carry without necessitating the use of any external programmable routing resources which are known to be rather slower than a hard-wired path.

2.2 Altera Quartus Tool

Altera Quartus II is a design tool produced by the Altera Incorporation. This software allows designers to synthesize and analyze designs, simulate and conduct some timing analysis as well as configuring the device at hand (such as an FPGA or a CPLD) to implement a particular circuit or application [7].

Once Quartus performs synthesis and mapping, an option called Verilog Quartus Mapping (VQM) Writer can be used to generate an atom-based netlist of logic functions, in Verilog.

This tool takes a digital circuit either as a Verilog/VHDL file or as a Block Diagram file, and performs synthesis, mapping, placement and routing. If needed, Quartus can generate the mapping netlist of the design in question through the VQM writer option. This writer simply lists all the nets generated for a particular design after synthesis and mapping, in a text file with the ".vqm" extension. Each net can be expressed in the following format (assuming a Stratix II FPGA) [7]:

```
Stratixii_lcell_comb <lcell_name>
(
        .dataa (<data_a source>),
        .datab (<data_b source>),
        .datac (<data_c source>),
        .datad (<data_d source>),
        .datae (<data_e source>),
        .dataf (<data_f source>),
        .datag (<data_g source>),
        .cin (<carry in source>),
        .sharedin (<shared function input source>),

        .combout (<combinational output>),
        .sumout(<arithmetic sum output>),
        .cout(<carry output>),
        .sharedout(<shared function output>)
);
defparam <lcell_name>.lut_mask = <lut mask>;
defparam <lcell_name>.shared_arith = <on, off>;
defparam <lcell_name>.extended_lut = <on, off>;
```

When the logic cell is used in normal mode, one should expect the following format:

```
Stratixii_lcell_comb <lcell_name>
(
        .dataa (<data_a source>),
        .datab (<data_b source>),
        .datac (<data_c source>),
        .datad (<data_d source>),
        .datae (<data_e source>),
        .dataf (<data_f source>),
        .datag (<data_g source>),

        .combout (<combinational output>),
);
defparam <lcell_name>.lut_mask = <lut mask>;
defparam <lcell_name>.shared_arith = <on, off>;
defparam <lcell_name>.extended_lut = <on, off>;
```

However, when the logic cell is used in Arithmetic mode, one should expect the following format:

```
Stratixii_lcell_comb <lcell_name>
(
        .dataa (<data_a source>),
        .datab (<data_b source>),
        .datac (<data_c source>),
        .datad (<data_d source>),
        .datae (<data_e source>),
        .dataf (<data_f source>),
        .datag (<data_g source>),
         .cin (<carry in source>),

        .sumout(<arithmetic sum output>),
        .cout(<carry output>),
);
defparam <lcell_name>.lut_mask = <lut mask>;
defparam <lcell_name>.shared_arith = <on, off>;
defparam <lcell_name>.extended_lut = <on, off>;
```

Quartus users can actually perform a reverse process where knowing the format of such netlists, a Verilog file can be created to design a particular circuit using such logic cell expressions. The Verilog file can be fed back to Quartus as a new design file where Quartus would perform the usual flow and implement the designed circuit.

2.3 Related Work

Throughout the years, researchers have developed numerous algorithms tackling the mapping problem and trying to optimize it [8], [9] depending on certain parameters such as power minimization [10], [11], area and delay [12]. The area optimization problem was proven to be NP-hard when mapping is applied on 4-or-more-input Look-Up Tables (LUTs), and as such heuristics needed to be developed in order to properly address such an optimization [13], [14].

Ideas in the same context of the proposed approach, trying to use Boolean Matching techniques to optimize the mapping process have been presented. Some proposed a Boolean Matching technique for mapping on networks of Programmable Logic Blocks (PLBs) [15], expressing it as a Boolean Satisfiability problem; however their approach remained algorithmic since no manipulations on the structure were tried while performing the Boolean Matching technique. Others tried to add to the algorithmic power some intelligence derived from observed architectural symmetries of the structure used [16]; however, this approach did not venture into configuring the structure at hand in order to achieve better mappability.

Some researchers tried to optimize the Boolean Matching technique in general by exploring the flexibility added using Don’t Cares which increases the probability of finding the function at hand [17]. A similar idea will be partially explored in one of the proposed approaches.

Various logic decomposition techniques with respect to technology mapping have been also researched trying to optimize the mapping process for either area or delay [18], [19]. However, such techniques are generating new algorithms to improve mapping of logical circuits over current FPGA's logic cells and with its present normal mode architecture.

One attempt of mapping logic functions on the FPGAs' carry-chain has been tried [20]; however the approach uses the carry-select chains which have been replace by the ripple carry-chain in modern FPGAs; and as such, the technique became obsolete. Another approach used the fast carry chains in order to design compressor trees [21].

A simplified version of this work has been first proposed as part of a mapping technique and a 6-input function decomposition approach, combined with a chaining algorithm, trying to map logic functions onto FPGAs' carry-chains in order to optimize the use of routing resources [22]. It has also been used as a simulation approach to generate data for a theoretical proposition of generic logic chains, trying also to reduce the congestion of the external routing resources [23].

Chapter Three

Generic Boolean Matching Technique

3.1 Cell Configurations

The Adaptive Logic Module (ALM) of the Stratix II-V FPGAs uses the fast carry-chain whenever it is operating in arithmetic mode, as shown in its general structure of Figure 1. Knowing that during the mapping process, each logic function will be typically mapped to a half-ALM. This half-ALM consists of two 4 input-LUTs and a multiplexer when the ALM is in normal mode as opposed to two 4 input-LUTs and an adder when the ALM is in arithmetic mode.

This general arithmetic structure of the ALM can be configured in various ways; however, logically, each adder must be used to implement one function. As such, even in the arithmetic mode, each function is mapped onto one set of two LUTs and an adder. For simplicity reasons, each half-ALM will be referred to as a logic cell. From Figure 1, it is clear that each set of two LUTs can have up to five external inputs while respecting the overall constraint of eight inputs per ALM. So in this section, various configurations will be explored and the most important and relevant structures will be listed. Each structure is a possible configuration of one or multiple logic cells, depending on the

structure. The structures are named, for each logic cell, in an X-Y-Z format where X is the number of inputs used by the logic cell's first LUT, Y is the number of inputs used by the logic cell's second LUT and Z is the number of shared inputs between these two LUTs.

3.1.1 - Structure 3-3-2

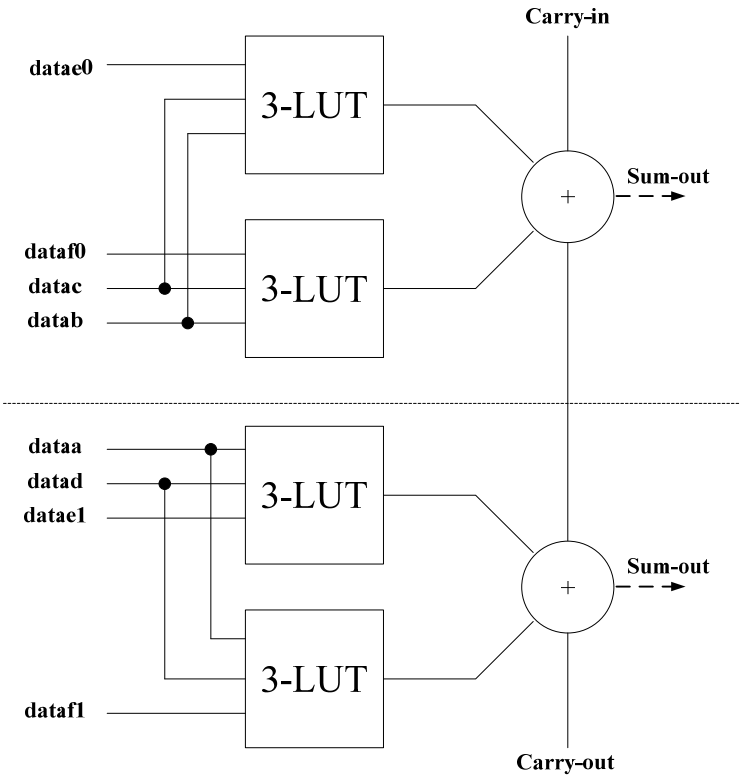

Figure 2: ALM in a 3-3-2 structure

This structure can be configured by forcing the first logic cell to be independent of input a (dataa) and the second logic cell of b (datab), as shown in Figure 2. This way, the two

logic cells do not share any inputs and as such each one depends on a totally different and independent set of five inputs: datab, datac, dataf0 and datae0 for the first logic cell and dataa, datad, datae1 and dataf1 for the second logic cell. Therefore, in this configuration, both logic cells have a 3-3-2 structure but for simplicity reasons it is called the 3-3-2 structure instead of the 3-3-2_3-3-2 structure.

3.1.2 - Structure 4-4-3

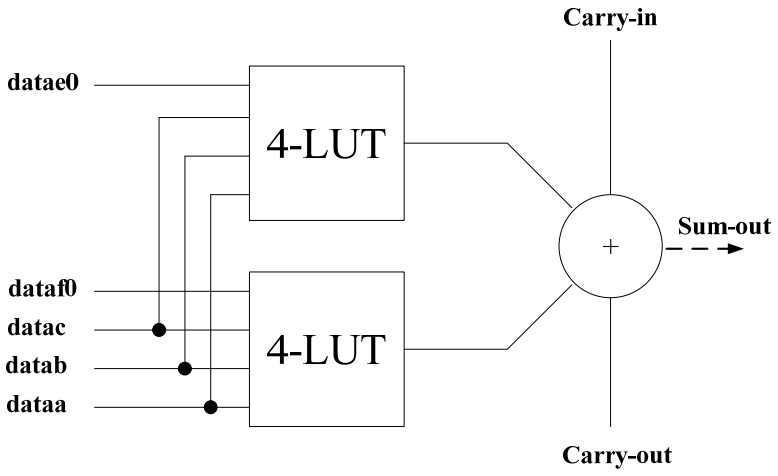

Figure 3: Logic cell in a 4-4-3 structure

Figure 3 represents this particular configuration, where inputs a and b (dataa and datab) are both dedicated to the same logic cell. As such, the LUTs have an overall of five inputs and each one of these LUTs have four inputs with three shared among each other. One of the main advantages of this structure, represented in Figure 3, is its ability to map up to 6-input functions if the carry-in of the adder is considered as an additional input.

However, looking at the two logic cells of the ALM, having a 4-4-3 structure on the first logic cell leaves only 3 independent inputs to the second logic cell. In order to properly benefit from the ALM and not waste the remaining available resources, the second logic cell can be used with a 2-2-1 configuration to map a subset of up-to-4-input functions. This 2-2-1 configuration will be presented in details in section 3.1.4.

3.1.3 - Structure 4-4-4

The 4-4-4 structure is very similar to the 4-4-3 structure and can be directly derived from it. This configuration can be achieved by having all four inputs of the LUTs of the logic cell shared among each other. One can notice from the general structure that three inputs of the first LUT are already shared with the second LUT, while each LUT reserves one independent input for itself. So in order to derive the 4-4-4 structure, pins e0 and f0 (datae0 and dataf0) must be assigned to the same input or variable and as such it would be considered as if these two pins are actually connected, as shown in Figure 4. For example, one possible implementation of a function f(v,w,x,y,z) on the 4-4-4 structure can be achieved by:

- assigning ‘v’ to datae0
- assigning ‘v’ also to dataf0
- assigning ‘w’ to datac
- assigning ‘x’ to datab
- assigning ‘y’ to data
- assigning ‘z’ to the Carry-in

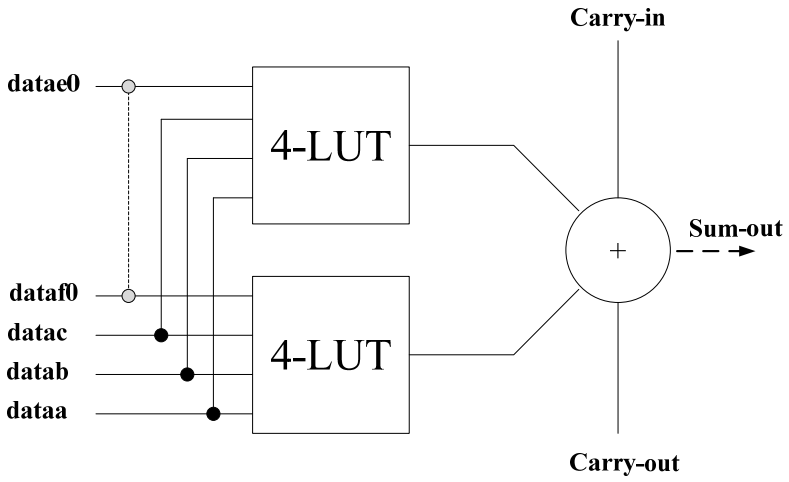

Figure 4: Half-ALM in a 4-4-4 Structure

This 4-4-4 structure can map up to 5-input functions. Using this configuration, the ALM's resources are mainly dedicated to one logic cell while the other cell is limited to only three inputs. So, an efficient way of using these remaining resources would be to force the second logic cell to a 2-2-1 configuration. This 2-2-1 configuration will be discussed in details in section 3.1.4.

3.1.4 - Structure 2-2-1

This is a very specific structure that is used whenever the logic cell is limited to only three available independent inputs for the LUTs. So in this 2-2-1 configuration, each LUT has two inputs, one shared and one independent, as shown in Figure 5.

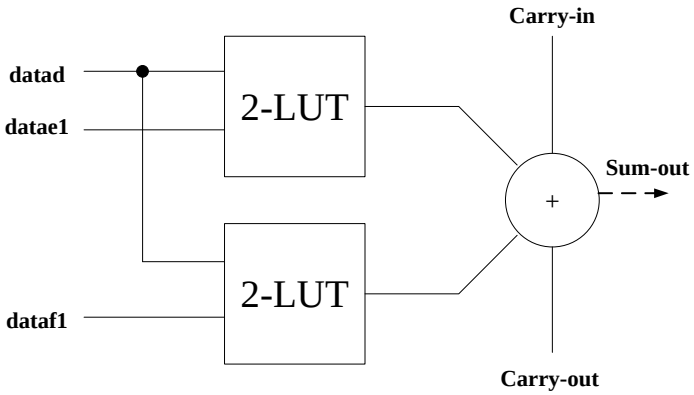

Figure 5: Half-ALM in a 2-2-1 Structure

This configuration is typically needed for the second logic cell of the ALM when the first logic cell is configured as a 4-4-3 or a 4-4-4 structure.

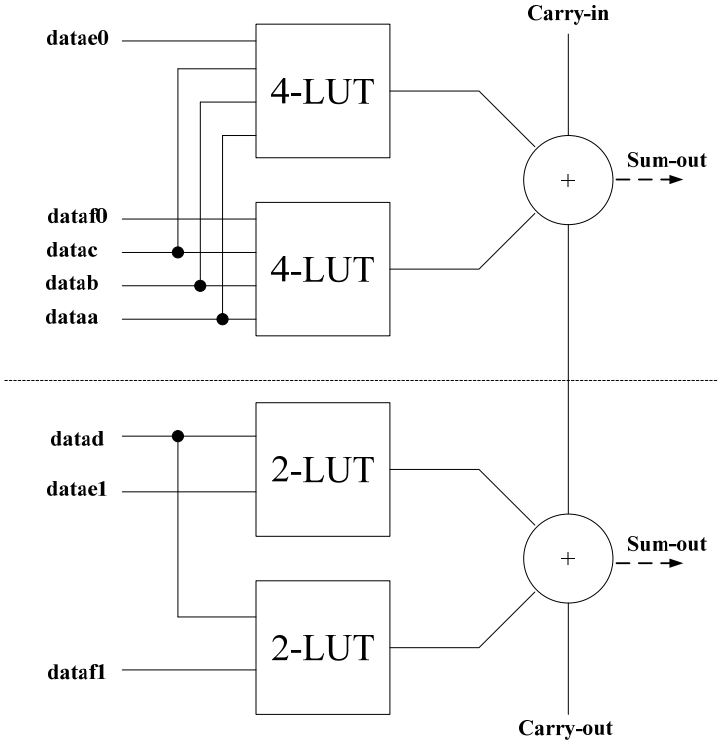

Figure 6: ALM in a 4-4-3_2-2-1 structure

In the first case, represented in Figure 6, after using a 4-4-3 structure on the first half ALM to map a 6-input function, the ALM is left with three inputs from the LUTs side (four if the carry-in is considered). Therefore, in order to properly benefit from the second logic cell, a 2-2-1 configuration is used to possibly map up to four-input functions.

Similarly, the second case is when the first logic cell is configured using a 4-4-4 structure where the same scenario occurs, and forcing a 2-2-1 structure on the second logic cell allows efficient use of the remaining resources available in the ALM. Otherwise, the remaining LUTs and adder are useless and the ALM is partially used.

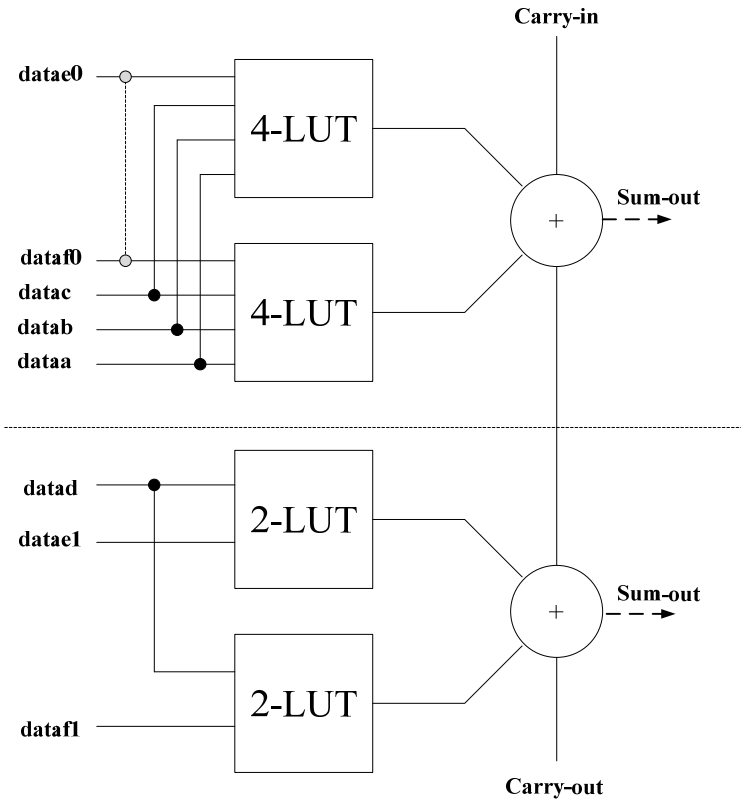

Figure 7: ALM in a 4-4-4_2-2-1 structure

3.2 Boolean Matching Technique

The main idea in this work is to try to efficiently use the already available FPGA's dedicated routing resources such as the carry-chain wire in order to pass output values and connect functions. As such, using the carry-chain would optimize and reduce the use of the external and programmable routing resources. However, carry-chains are only used when the ALMs operate in arithmetic mode. So the ALMs need to be used in arithmetic mode, to implement non-arithmetic operations. And by that, the necessity to map on the adders arises. Placing the output of one function on the carry-chain through the adder's carry-out, allows the next function to use it as an input through its carry-in, as shown in Figure 8.

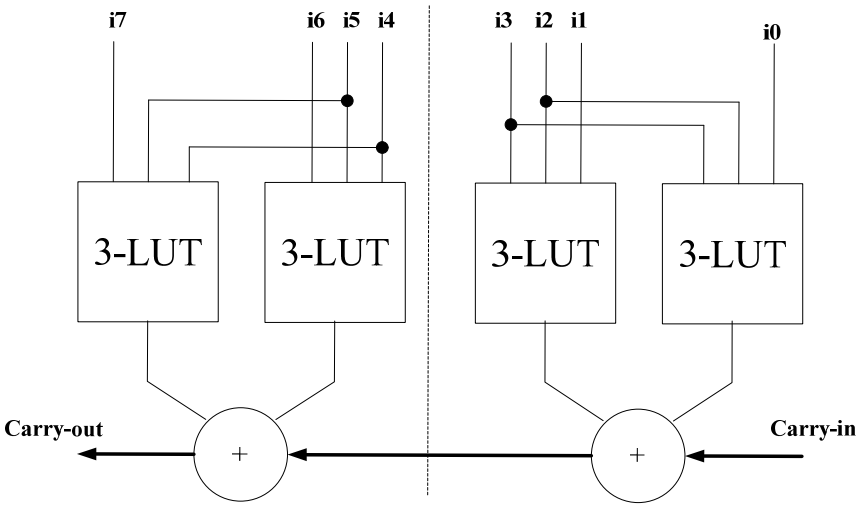

Figure 8: Carry propagation in arithmetic mode

However, a limited number of non-arithmetic logic functions can be mapped on the logic cell of the ALM's arithmetic mode. Therefore, the Boolean Matching technique

was developed in order to identify the non-arithmetic logic functions that can be mapped on such architecture. This section explains in details the approach that was adopted and the technique used to specify if a specific function can be mapped on a particular configuration of the logic cell in arithmetic mode.

For simplicity reasons, the technique will be explained on one particular structure, which is the 3-3-2 structure; however, the same technique can be applied on the remaining structures in a similar way.

Taking into consideration the carry-in as an additional input, a logic cell can be looked at as a 5-input structure. Since the output of each cell needs to be placed on the fast carry-chain, the Sum-out of the adder will be disregarded and the carry-out will be the main output of the macro cell, as seen in Figure 9.

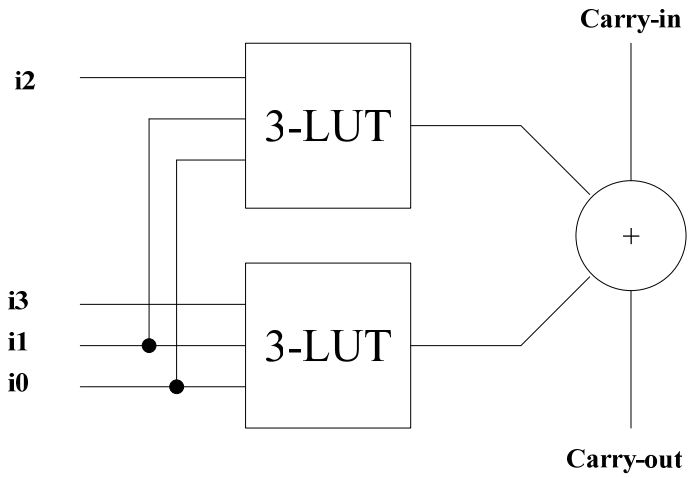

Figure 9: Logic cell in a 3-3-2 structure

The presence of the adder in the structure of these macro cells, as well as the characteristics of the LUTs (size and shared inputs) limits the number of functions that can be mapped to it. A small library was generated in order to identify which functions

can be mapped on such a structure and therefore be placed on the fast carry-chain. Coming up with this library requires careful analysis of the behavior of the adder and the LUTs separately.

The behavior of the adder is well known; however, one can express the carry-out of this adder, which is the output of the Logic Cell, in terms of the adder's inputs as shown in Table 1. If these inputs are equal to each others, the carry-out will be also equal to these inputs. However, if the inputs are not equal (i.e. one is the opposite of the other), the carry-out will be equal to the value of the carry-in.

A	B	Carry-in	Carry-out
0	0	0	A
0	0	1	A
0	1	0	Cin
0	1	1	Cin
1	0	0	Cin
1	0	1	Cin
1	1	0	A
1	1	1	A

Table 1: Behavior of the Carry-out of the adder

On the other hand, each of the two LUTs, separately, can implement any function. However, the shared two inputs limit their possible outputs, since the output of one LUT might depend on the output of the other. So exploring the dependencies between the outputs of the LUTs with respect to the four different LUTs' inputs is an essential step in identifying the range of functions that can be mapped onto the structure of the logic cell in arithmetic mode. Table 2 lists the states of the LUTs outputs with respect to all possible combinations of the inputs. The output of the first LUT (LUT_1) depends on the inputs i_0, i_1 and i_2 while the output of the second LUT (LUT_2) depends on the inputs i_0, i_1

and i_3. So for example, the state a_0 is the output of LUT_1 when its inputs are 000, b_1 is the output of LUT_2 when its inputs are 001, etc. Analyzing Table 2, one can notice that it can be divided into four different sections where the outputs' states in each section are totally independent of the outputs' states in any of the other three sections. For example, the first section includes states a_0, a_1, b_0 and b_1. Furthermore, comparing all four sections, it can be also noticed that they have identical state patterns, which means that analyzing one section of the list provides sufficient information on all four sections. So now, the truth table is reduced to only one section, composed of four variables (the four different states of the LUTs outputs).

i_0	i_1	i_2	i_3	LUT_1	LUT_2
0	0	0	0	a_0	b_0
0	0	0	1	a_0	b_1
0	0	1	0	a_1	b_0
0	0	1	1	a_1	b_1
0	1	0	0	a_2	b_2
0	1	0	1	a_2	b_3
0	1	1	0	a_3	b_2
0	1	1	1	a_3	b_3
1	0	0	0	a_4	b_4
1	0	0	1	a_4	b_5
1	0	1	0	a_5	b_4
1	0	1	1	a_5	b_5
1	1	0	0	a_6	b_6
1	1	0	1	a_6	b_7
1	1	1	0	a_7	b_6
1	1	1	1	a_7	b_7

Table 2: Relation between the LUTs outputs

Knowing the possible dependencies between the outputs of the LUTs, the last step consists of listing the possible values of the carry-out which results from processing the outputs of the LUTs along with the carry-in through the adder. Therefore, only the

subsequences of the functions that can be mapped to the macro cell structure in arithmetic mode are saved. The list of these 2^4 =16 subsequences is stored in a library where each subsequence has a length of 8-bit. These subsequences can be generated by listing all possible states' combinations and passing it, along with the carry-in, through the adder by following the adder's behavior stated in Table 2.

LUT_1	LUT_2	C_{in}	Possible output subsequences					
a_0	b_0	0	0	0	0	0	0	…
a_0	b_0	1	0	0	1	1	0	…
a_0	b_1	0	0	0	0	0	0	…
a_0	b_1	1	0	1	0	1	0	…
a_1	b_0	0	0	0	0	0	0	…
a_1	b_0	1	0	0	1	1	1	…
a_1	b_1	0	0	0	0	0	0	…
a_1	b_1	1	0	1	0	1	1	…

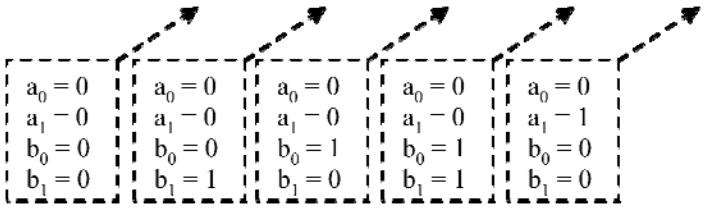

Table 3: Generation of possible output subequences

So now Table 2 can be updated where each state of the LUTs outputs can be replaced by actual values of 0 or 1. Thus, for each combination of these state values, along with the carry-in, a possible output subsequence can be generated on the carry-out. So whenever the outputs of the two LUTs are identical (whether equal to 0 or to 1) the carry-out of the adder has the same value as the output of the LUTs; however, if the outputs of the LUTs are opposite, the carry-out will be equal to the carry-in. Following this logic, all output subsequences can be generated by simply considering all the input combinations. There are a total of 16 output subsequences having a length of 8 bits each. Table 3 shows the

generation of some subsequences for particular values of the LUT outputs or in other words, particular values of the adder's inputs.

Having any function at hand, this newly created library helps identifying whether or not it can be mapped to the half ALM in arithmetic mode and with that, the placement of its output on the carry-chain. In order to test the eligibility of any function, its mask (or the function's truth table) must be divided into four parts, where each part must be checked against the generated library:

- If all parts of this function exist in the library, then the function can be mapped to the macro cell
- If at least one part of this function cannot be found in the library, then the function cannot be mapped to the structure.

Moreover, it is known that the output sequence of any truth table can change when considering a different order of the inputs. So having the truth table of a particular function, the probability of mapping that function to the structure might be improved by permuting the inputs' order of this function since it might lead to a new output sequence. Applying the Boolean Matching Technique to this new output sequence might allow the function to be mapped on the considered logic cell.

One of the main advantages of this library is that it does not necessitate storing all 32 bit long output combinations. Instead, only a small set of sequences with sufficient

information is saved, which reduces the memory required for the storage of such libraries.

Finally, as stated earlier, this technique was generalized so that I can support any of the configurations listed earlier.

This Boolean Matching technique was first presented in [22] combined with an effective chain-selection algorithm to optimize the use of routing resources. However, at that time, the technique was limited to the 3-3-2 structure and could not support various configurations. Furthermore, section 3.3 will present an enhancement on this Boolean Matching technique so that it supports not only various configurations of the Stratix ALMs in arithmetic mode, but also some theoretical structures that has not been designed or manufactured yet. This will introduce a higher flexibility and the possibility of researching and exploring potential performance improvement through new logic cell architectures.

3.3 Flexible Structure Boolean Matching Technique

After coming up with the Boolean Matching Technique to identify the functions mappable on particular structures, a generalization of this technique was explored trying to cover all possible configurations, realistic or hypothetical. This technique was enhanced by allowing a variety of modifications on the used logic cell, while respecting the following terms:

- A general architecture composed of two LUTs whose outputs are feeding a three-input function has to be respected.
- Each LUT can have a variable number of inputs independently of the other LUT. No upper-bound is set for the LUT's inputs. For example, assuming that LUT_1 has 5 inputs, LUT_2 might have 8 inputs.
- The total number of shared inputs between the two LUTs is variable and can take any value.
- The adder can be replaced by any three-input function as long as its behavior is known. The carry-in can simply be considered as an additional input.

As long as the new hypothetical structure does not cover and map all functions, the enhanced technique uses the same general method to generate a library of mappable subsequences, and identifies mappable functions by testing its existence in this newly generated library. The testing approach is similar to the previously explained one, except that some parameters will differ:

- The library's sequences will change depending on the function used to replace the adder. i.e. for the same LUTs parameter, changing the carry-out function by another one will change the library's sequences.
- The length of the library's sequences will vary depending on the LUTs number of inputs as well as the number of shared inputs
- The number of partitions, which is the number of times a function is divided to generated testable sequences (on a 3-3-2 structure the number of partitions is 4) might change depending on the total number of inputs and the length of the library's sequences.
- The size of the library (i.e. the number of available sequences) would change with all parameters, such as the function replacing the adder, the number of LUTs inputs and number of shared inputs.

This enhanced mapping technique was embedded in a tool that allows the designer to compare various configurations and structures generating relevant information for particular specifications and preferences. This tool can also propose a particular structure depending on the user's design and priorities for some predefined parameters. This tool is presented in details in Chapter 5.

On a side note, this technique is limited by the structure at hand. This means that if the structure can support a maximum of 5 inputs, the Boolean Matching technique cannot map 6-or-more-input functions. Furthermore, if a 5-input function cannot be mapped

using the Boolean Matching technique, this function is labeled as ‘unmappable’ and nothing is further done to try and map it.

For this purpose, decomposition techniques were proposed so that unmappable functions (rejected by the Boolean Matching Technique or having a structure-unsupported number of inputs) can be decomposed into two functions that might be mappable. These decomposition techniques will be discussed in details in section 3.4.

3.4 Decomposition Techniques

Decomposition techniques became a necessity once statistics were generated over common benchmarks and where a substantial number of logic functions were found to be unmappable on the various configurations of the ALMs' arithmetic mode of the Altera's Stratix FPGA.

So if the function cannot be mapped, as is, then decomposing it into two functions, while maintaining its overall logical behavior, might actually map it. Two different decomposition techniques were adopted depending on the nature of the function: the decomposition of 6-input functions and the decomposition of 5-and-less-input functions.

3.4.1 - Decomposition of 6-input functions

As seen in the section 3.1, 6-input functions can only be mapped on a 4-4-3 structure when mapping is performed on ALMs in arithmetic mode. However, statistics showed that for the used MCNC benchmarks, a small set of these 6-input functions can be actually mapped on the 4-4-3 configuration. So a decomposition technique was developed, trying to map as many 6-input functions as possible.

This technique takes a 6-input function and tries to decompose it into two 5-or-less-input functions while maintaining the same overall logical behavior. Assuming that the function to be decomposed is f, the idea is to find two functions f1 and f2 such that f1

would be mapped on the first logic cell of the ALM with its output placed on the carry-chain through the carry-out. f2 is placed on the second logic cell, taking f1 as an additional input through the carry-in while its output is placed on the carry-chain, through the carry-out of the second logic cell. Choosing f1 and f2 should abide to the following conditions:

- f1 should be independent of one of the function's 6 inputs.
- f1 should be mappable on a logic cell with a 3-3-2 configuration.
- f2 should be independent of two inputs:
    - The input that f1 is using as carry-in
    - One of the function's 6 inputs, as long as it is not the same input f1 is independent of.
- f2 should be mappable on one 3-3-2 structured logic cell with f1 being the input placed on the carry-chain.

Assuming that the 6-input function is $f = F(u, v, w, x, y, z)$ then one possible set of $\{f1, f2\}$ could be:

$$\begin{cases} f1 = F(u, v, w, x, y) \\ f2 = F(u, v, w, z, f1) \end{cases}$$

The decomposition of the 6-input functions and the process of finding f1 and f2 is represented in details in the diagram of Figure 10. Note that f_ind_i and f_ind_j are functions derived from f and independent of the inputs i and j respectively. Since these independent functions are derived from a 6-input function, they depend on a maximum

of 5 inputs. Before ORing the two functions, common minterms between functions f1 and f2 are set as Don't Cares in function f2 since these minterms exist in both functions which will be later on ORed. So changing these minterms to DC introduces more flexibility to function f2 and, as such, increases its probability of being mapped.

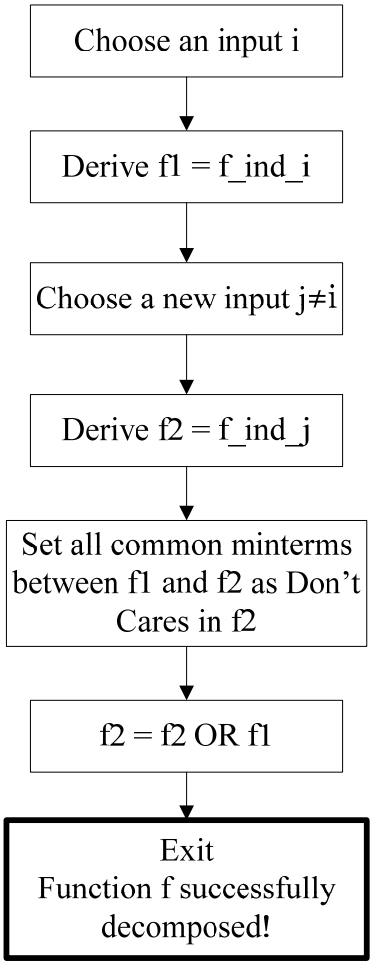

Figure 10: 6-input function decomposition process

Now, adding the conditions of having f1 and f2 mappable on the 3-3-2 structure, the decomposition and mapping process can be represented as shown in Figure 11. If one possible decomposition (i.e. one set of {f1, f2}) turned out to be unmappable, new decompositions are tried through exploration of more {f1, f2} sets by changing the inputs from which the independent functions are derived. Figure 11 explains this process in details.

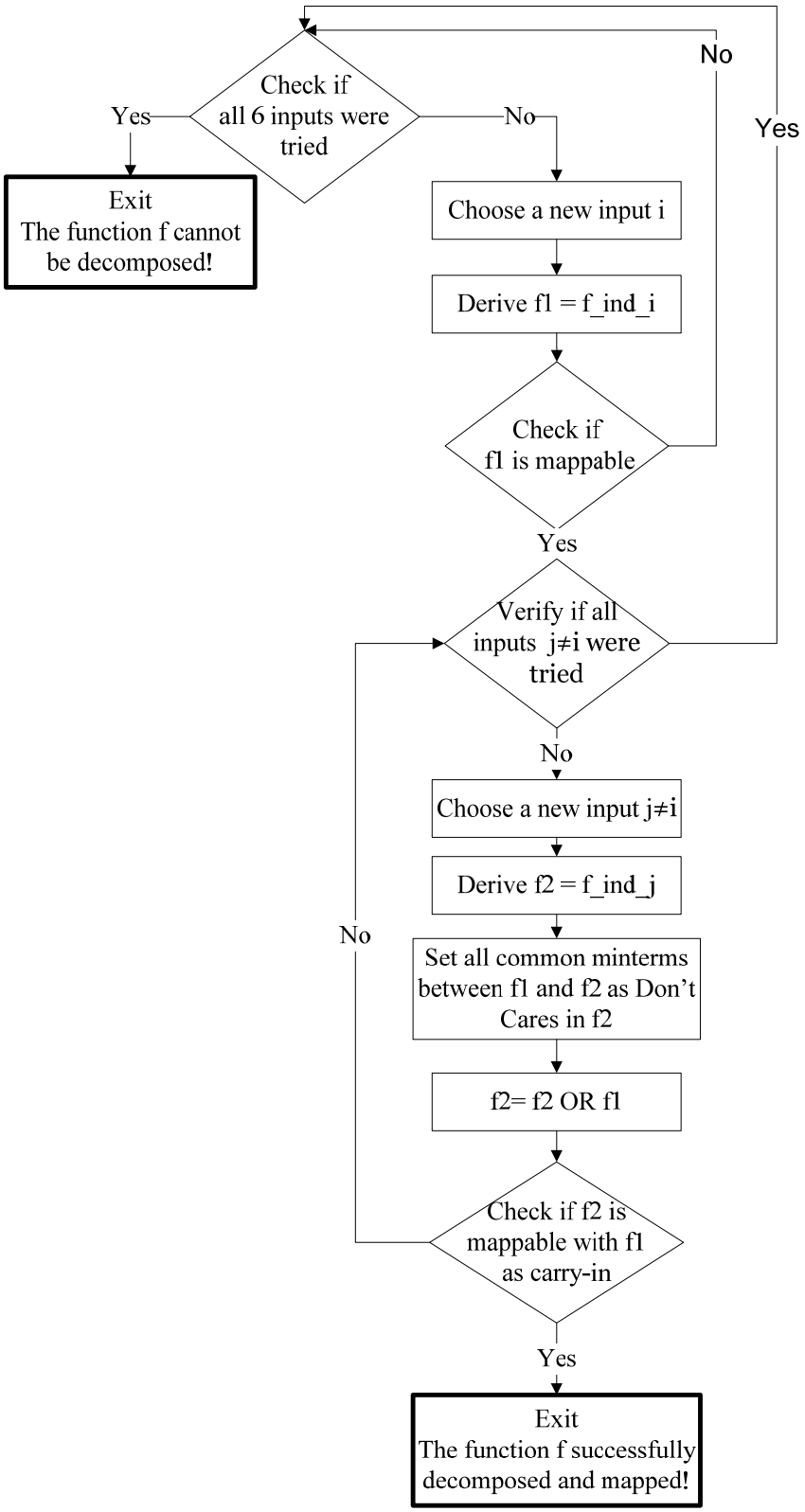

Figure 11: 6-input function decomposition and mapping process

3.4.2 - Decomposition of 5-and-less-input functions

Several possible configurations have up to 5 inputs (such as 3-3-2, 4-4-4 and 4-2-2 configurations) and as such can implement a large set of 5-and-less input functions. However, some functions still cannot be mapped on any of these structures. So a decomposition technique was developed as a way to map such unmappable functions.

In this section, all encounters of the term ‘function’ would be a reference to the 5-and-less-input functions since this decomposition technique is only used on such functions.

So this decomposition approach divides any function f into two functions f1 and f2, in such a way that:

- f1 is a 5-or-less-input function.
- f2 is also a 5-or-less-input function.
- If the initial function f is an X-input function, then f2 should have up to X-1 inputs out of the X-inputs, where the omitted input should be the one placed on the carry-in in f1. One additional input of f2 should be f1.

Assuming that the function to be decomposed is $f = F(v, w, x, y, z)$ and that z was the input placed on the carry-in of the function f1, then one possible set of $\{f1, f2\}$ could be:

$$\begin{cases} f1 = F(v, w, x, y, z) \\ f2 = F(v, w, x, y, f1) \end{cases}$$

Now this function f needs to be mapped on the ALM in arithmetic mode. So, each of f1 and f2 should be mapped on one logic cell; while the output of f1 is passed onto f2 through the carry-out carry-in connection. To be able to successfully decompose f and map both f1 and f2, four additional conditions need to be satisfied:

- f1 should be mappable on the logic cell with the 3-3-2 configuration.
- f2 should also be mappable on the logic cell with the 3-3-2 configuration.
- f2 should be mappable with f1 being the input read from the carry-in.
- f2 should be independent of the input that f1 uses as carry-in.

Adding some intelligence to the decomposition, this technique starts from an initially known-to-be-mappable function f1. Using this f1, it tries to derive f2 while insuring the overall logical behavior of the function f. To do so, f2 has to be a function of f1and up to four other inputs of f. Figure 12 lists in details the steps to be followed in order to successfully decompose and map such functions.

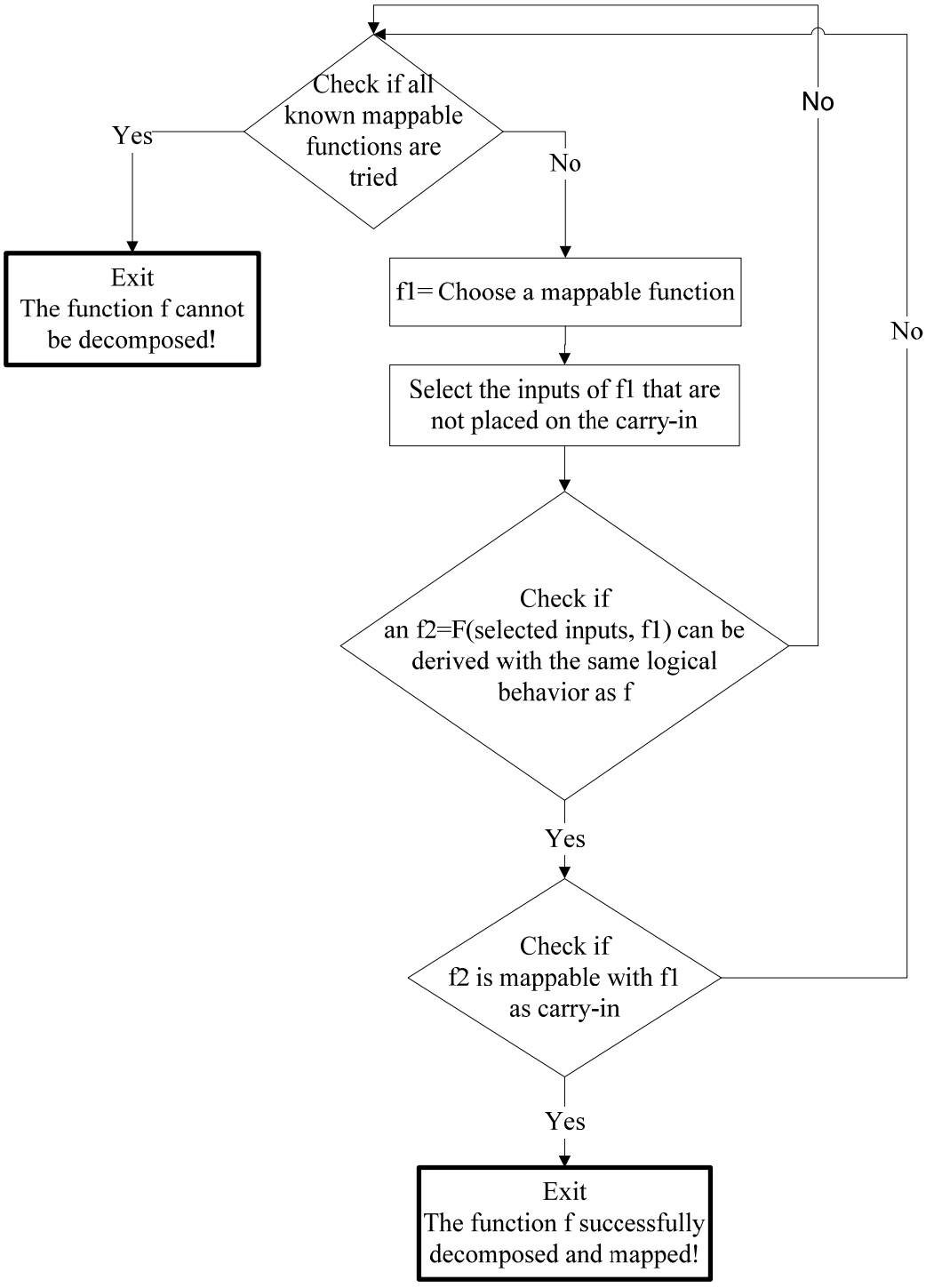

Figure 12: 5-and-less-input functions decomposition and mapping process

Furthermore, the complexity and performance of this decomposition technique can be enhanced using some previous knowledge of the library used to perform Boolean matching. Instead of decomposing the function and then checking whether the newly generated functions can be mapped or not, Boolean matching is somehow embedded inside the decomposition technique so that it would only decompose if that ensures mappability.

Starting from an initially known function f1, the technique should exhaustively try all possible mappable functions as f1. That is a complex and time consuming procedure. So the adopted alternative is the improved approach of Figure 13. In this approach, the function is first partitioned into segments as if it is being subjected to the Boolean matching technique. Then for each partition/segment, decomposition is applied using the library's subsequences as the first sub-functions on which decomposition is based. This approach can be further explained as follows:

- The function f can be divided into four partitions similar to the process used during the Boolean Matching technique. Each partition or sequence of f shall be called P_f.
- For every P_f the library is searched:
    - Every sequence in the matrix is considered as a partition of f1 (known as P_f1) and the process tries to derive a partition of f2 (P_f2) using P_f1.
    - P_f2 is derived as a partial function that has the same logical behavior as the partial function P_f. The inputs of P_f2 can be all the inputs of f except the one used as carry-in in f1. P_f2 also uses P_f1 as an additional

input. The derivation of P_f2 might not always be feasible since it might depend on unused inputs (such as f1's input which was used as carry-in).

- If P_f2 was successfully derived, the technique will then verify its mappability on the the 3-3-2 configuration by performing the normal Boolean Matching technique over one partition. In other words, the library is searched for the existence of P_f2 since finding it in the library signals its mappability.
- If one P_f2 was successfully derived and mapped, the technique would have found, for this partition, at least one set of {P_f1, P_f2} that generates P_f. As such, it starts applying the same process on the remaining partitions.

- If for one P_f the technique was not able to find any set of {P_f1, P_f2} that would generate P_f, then the entire function f cannot be mapped. This is due to the fact that even if the entire f1 and f2 are derived, the Boolean Matching technique will not be able to map it on the 3-3-2 structure since, for at least one partition, their respective sequences were found to be unmappable.
- If, for each partition, the technique was able to find at least one set of {P_f1, P_f2} that would generate P_f, then the function f can be successfully mapped on two logic cells with 3-3-2 structures. So now f is mapped through its decompositions {f1, f2} where f1 is mapped on the first logic cell with its output generated through the carry-out, while f2 is mapped on the second logic cell using f1 as an input through the carry-in.

- In order to generate f2, a reverse process must be performed, where the successfully derived sequences of P_f2, for all partitions, are joined to form the required f2. As similar process is applied to derive f1.

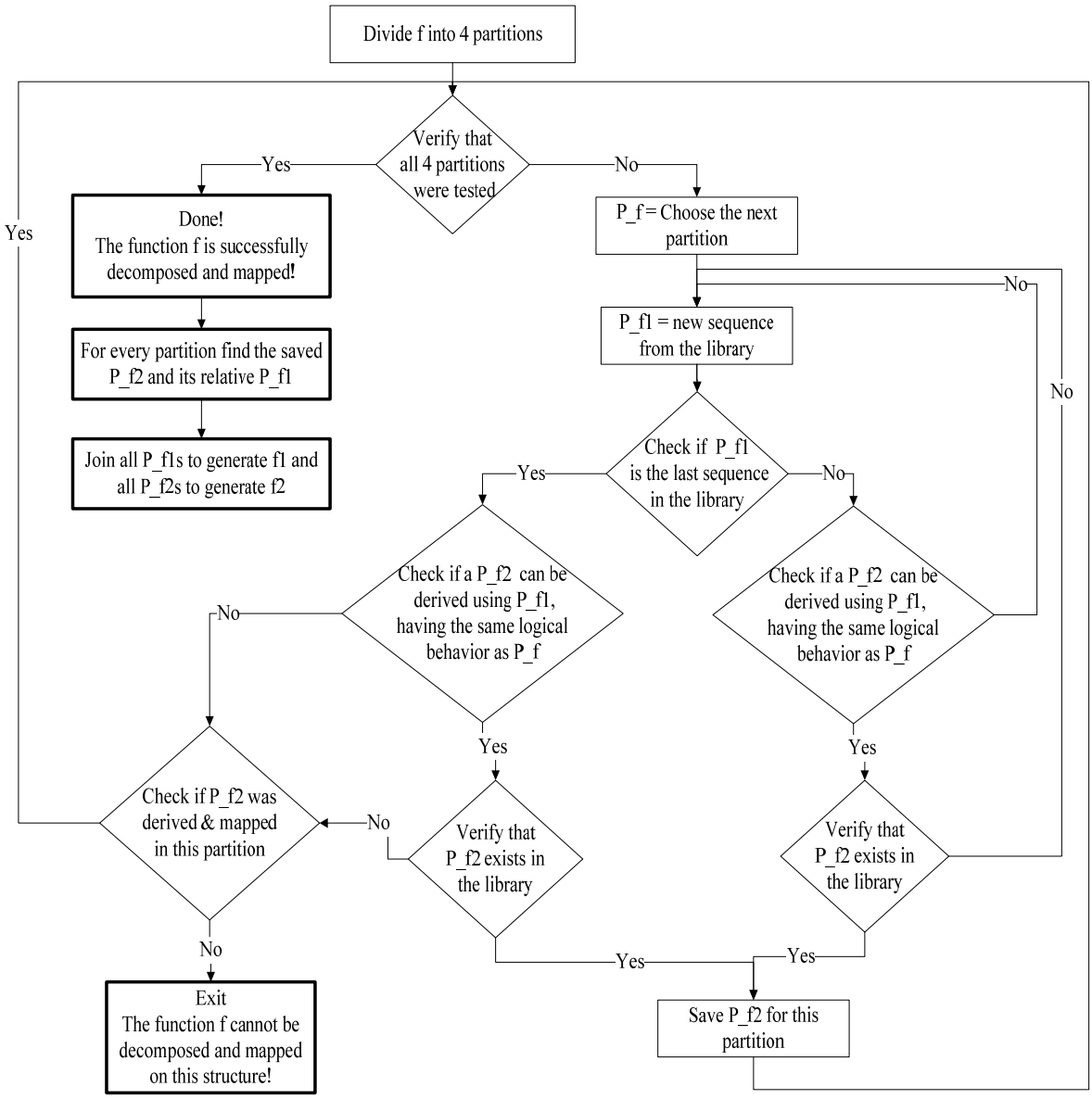

Figure 13: Improved decomposition of 5-and-less-input functions

Chapter Four

Enhanced Mapping Technique using the Generic Boolean Matching

This chapter proposes an enhanced mapping approach for the Altera Stratix Field Programmable Gate Arrays (FPGAs), using the so-called Generic Boolean Matching technique which performs mapping on flexible structure configurations while using the decomposition processes.

This technique aims on mapping logic non-arithmetic functions on FPGAs while using the Adaptive Logic Modules (ALMs) in arithmetic mode instead of the typical normal mode. This approach tries to propagate non-arithmetic outputs and connect logic cells using the hard-wired carry-chain which is already implemented in the ALM but only used while performing arithmetic operations,. Mapping non-arithmetic operations on the carry-chains routes functions without using the external routing resources, which reduces the congestion on the programmable interconnects, power dissipation, etc.

Altera uses the Quartus II tool to perform synthesis, mapping, placement and routing of any design, before implementing it on the FPGA. However, since only the mapping

process of synthesized functions is of interest and a comparison between the proposed approach and the currently used mapping technique is needed, Quartus will be used to perform the synthesis, placement and routing only, as shown in the diagram of Figure 14.

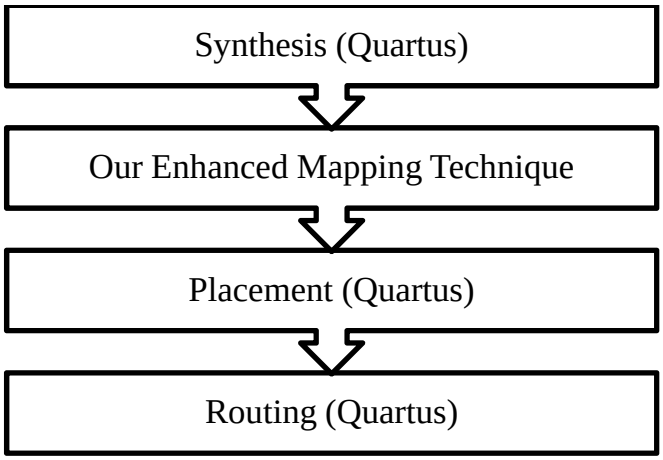

Figure 14: Modified flow of compilation phases

However, the proposed mapping technique does not directly map synthesized functions but relies on Quartus' mapping instead. In other words, this technique requires Quartus to perform synthesis and mapping, after which it parses the mapped cells and tries to remap it using the new approach. So the previous diagram can somehow be updated as shown in Figure 15.

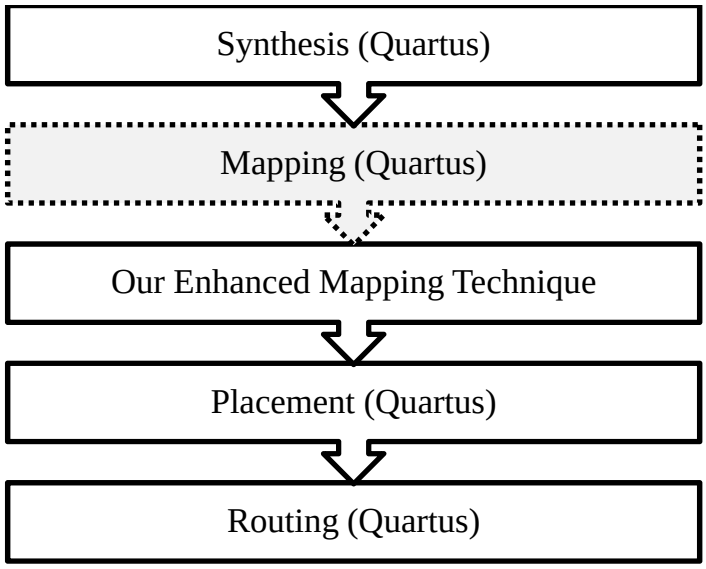

Figure 15: Updated tool flow

So this approach starts by parsing Quartus' mapping netlist and locating the benchmark's functions. Then for each function, the technique tries to map it on the one half-ALM in arithmetic mode. It might even try to map the selected function on several configurations of the logic cell depending on the function's nature and number of used inputs. Once decisions have been reached for all available functions in the design, a new mapping netlist is generated and returned back to Quartus to perform placement and routing.

The detailed flow diagram can be found in Figure 16.

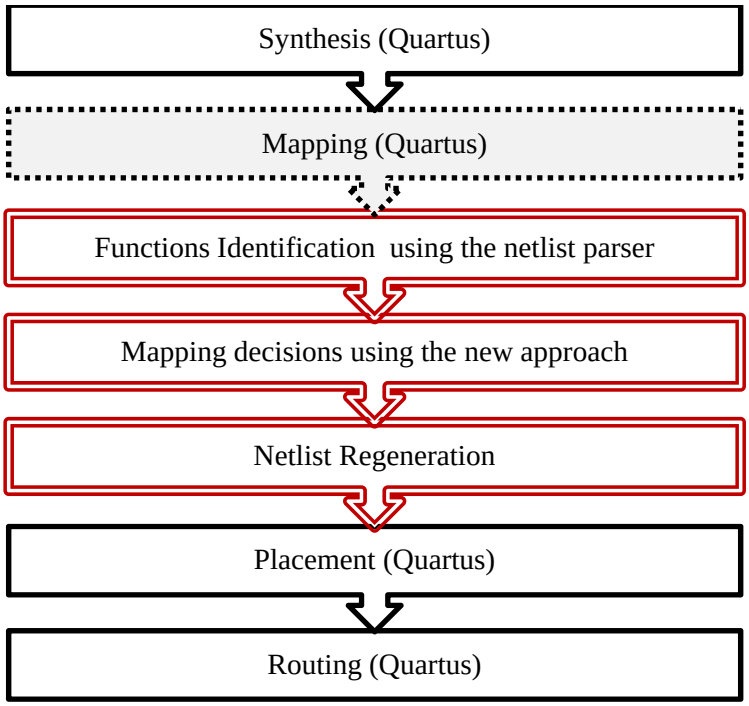

Figure 16: Detailed flow diagram

4.1 Netlist Parser (VQM Parser)

Once Quartus performs synthesis and mapping, an option called Verilog Quartus Mapping (VQM) Writer can be used to generate an atom-based netlist of logic functions, in Verilog.

The approach starts by parsing the vqm file generated through Quartus after synthesis and mapping. This parser creates a list of extracted functions along with their respective inputs and truth table. Each extracted function can have between two inputs and seven inputs. Once the functions are listed, a Data Flow Graph (DFG) is created, keeping track

of the function's successors and predecessors. Successors to a current operation are simply functions using the current output as an input. As such, the current operation becomes a predecessor to those functions.

4.2 The Enhanced Mapping Process

Traversing the DFG, every function is extracted separately and subjected to the mapping process. The Flexible-Structure Boolean matching technique that was presented in Chapter 3 tries to map each function on various structures depending on the function's number of inputs and truth table. If the function cannot be mapped on some selected structures, decomposition is performed.

In order to study this mapping technique in details, it must be divided into three sections, each presenting one approach depending on the function's number of inputs. These sections were selected knowing that the number of inputs of the DFG's functions range between 2 and 7 inputs.

4.2.1 - 7-input functions approach

These 7-input functions are rather rare to encounter and are usually mapped using the 'Extended-LUT' mode. As such, the impact of these functions on the generated statistics

and overall improvement is rather negligible. So at this stage, the proposed mapping technique will not map such functions on the ALM's arithmetic mode but instead, it will simply keep it as is and map it eventually on the ALM in extended-LUT mode.

4.2.2 - 6-input functions approach

Since the flexible-structure mapping technique is used, functions can be mapped on any of the realistic configurations of the logic cells of the ALM in arithmetic mode, as listed in Chapter 3. However, the only configuration that supports 6-input functions is the 4-4-3 structure of Figure 3.

So the approach tries to map a selected 6-input function f on the 4-4-3 configuration using the flexible-structure mapping technique. If f is not mappable, then the 6-input function decomposition technique is used to decompose it into two 5-or-less-input functions. If both techniques fail to map the selected function, then the approach renounces on trying to map it on the ALM in arithmetic mode and thus the typical normal mode is used.

However, in the case where f is mappable on the 4-4-3 configuration then the approach explores its successors trying to find one that can be successfully mapped on a 2-2-1 configuration while using f through the carry-in pin. If such a successor is found, the ALM will be used in a 4-4-3_2-2-1 configuration reducing the number of logic cells needed to map these two functions. That is due to the fact that, whenever in normal

mode, mapping a 6-input function uses 2 logic cells (an entire ALM), while now only one logic cell (half-ALM) is needed for mapping that same function. On the other hand, even if no successor is successfully mapped on the 2-2-1 configuration, the function f will still be mapped on the 4-4-3 structure. However, in this case, the ALM is underused and its second logic cell will be only passing the carry value without performing any useful operation.

The diagram in Figure 17 illustrates the steps to be followed when applying this approach.

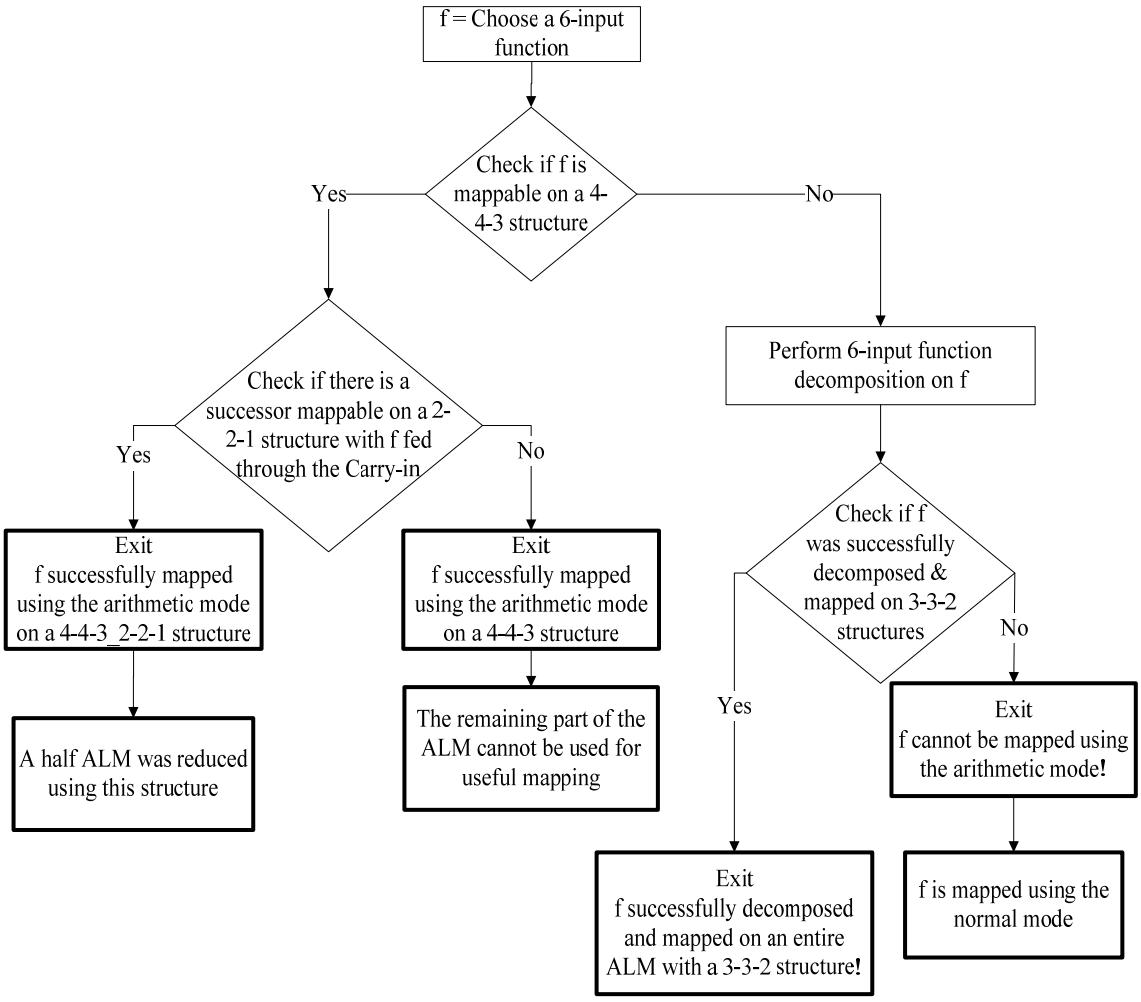

Figure 17: 6-input functions mapping process

4.2.3 - 5-and-less-input functions approach

This approach covers all functions having a number of inputs between 2 and 5. It tries to map the function f on configurations supporting five independent inputs. Several structures are tried during the mapping process of each function, depending on the configuration's priority. These priorities are pre-specified depending on the cost each configuration introduces, in such a way that a configurations' priority decreases as its cost increases.

Two structure configurations, as well as the decomposition technique, will be tried depending on the following priorities:

I. The 3-3-2 structure: This structure has the highest priority since it allows mapping two up-to-5-input functions, each on one logic cell without any overhead or loss. If a function can be mapped using this configuration, no other trials are necessary.

II. The 4-4-4 structure: Experiments proved that this configuration can map a much higher percentage of functions than any other structure. However, such mappability comes at the cost of having the second logic cell forced to a 2-2-1 structure which can map a rather small subset of up-to-4-input functions. When no functions are found to be mapped on the 2-2-1 structure, the 4-4-4 configuration consumes the entire ALM, introducing a loss of one half-ALM or logic cell.

III. Decomposition: The decomposition technique is the final option that is only used when mapping on the two previous configurations fails. This method has the least priority since it always introduces an overhead of an increase in the used logic cells since for every decomposed function, and entire ALM is needed (two logic cells, one for each of f1 and f2).

Therefore, initially, the technique tries to map the selected function f on a 3-3-2 structure. If it is not successfully mapped, the 4-4-4 structure is tried next. If f can be mapped on this 4-4-4 configuration, then successors are searched for one that can be mapped on a 2-2-1 with f being the input on its carry-in, in order to form a 4-4-4_2-2-1 ALM configuration.

However, if the selected function f cannot be mapped on any of the previous configurations, decomposition is performed using the 5-or-less-input function decomposition technique.

When all these steps fail, the approach stops trying to map f using the arithmetic mode, and as such it remains mapped using the normal mode.

The diagram of Figure 18 properly visualizes the steps followed in this approach.

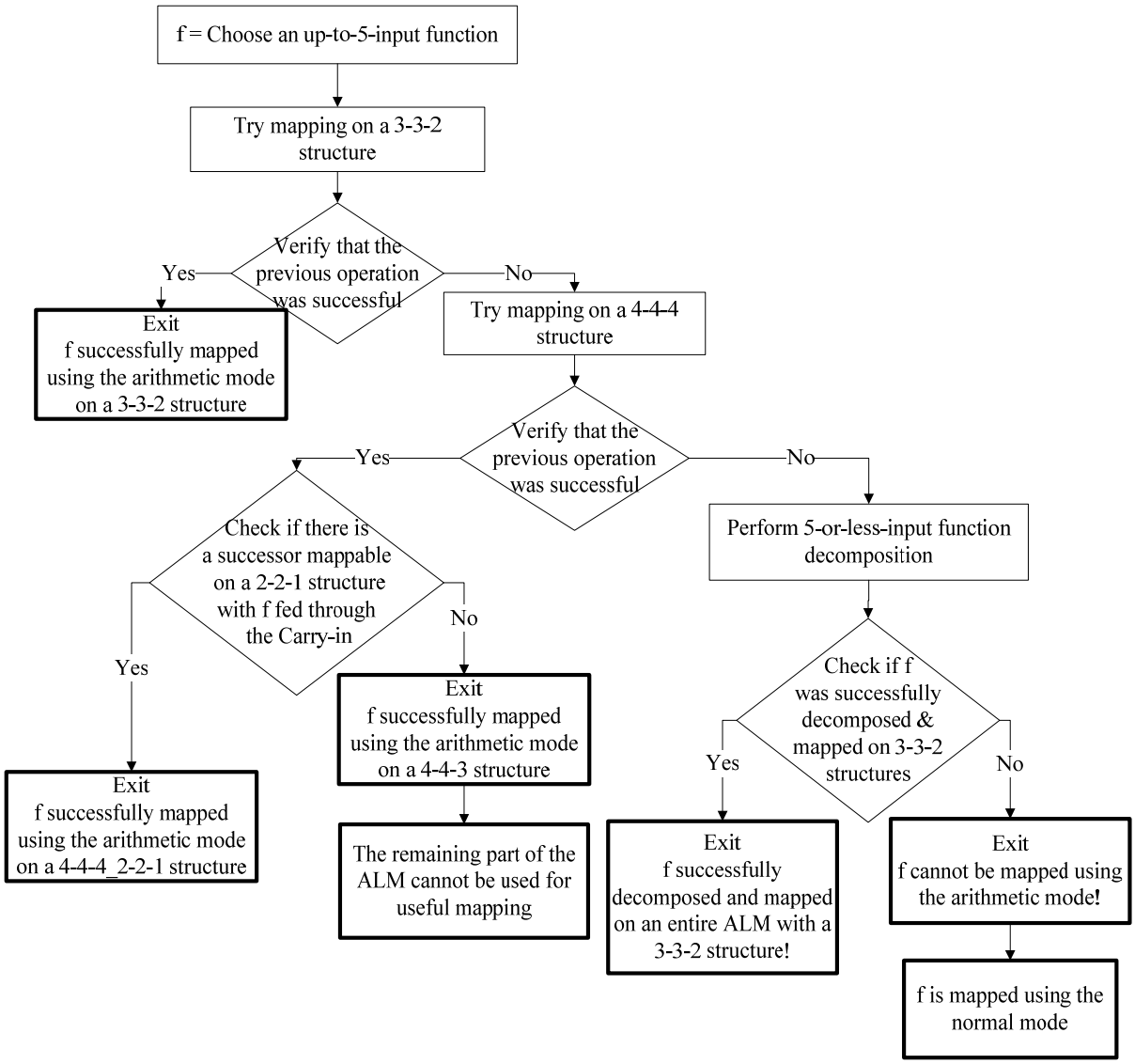

Figure 18: 5-or-less-input functions mapping process

4.3 Regeneration of the mapping netlist

Once mapping decisions have been reached for all functions of the DFG, a new mapping netlist is generated in order to reflect the new changes and mapping preferences of every function.

The generation of the new mapping netlist (or vqm file) must respect the following guidelines:

- The general interface should remain identical to the initial vqm file, such as entity declaration, inputs, outputs, wires, etc.
- All 7-input functions and functions that were not successfully mapped must be declared using the normal mode, as in the initial vqm.
- The declaration of the functions successfully mapped using the arithmetic mode must be changed so that:
    - The selected input is placed on the carry-in.
    - The output is generated on the carry-out.
    - The mask is changed to properly represent the function at hand.
    - The ALM pins used to convey the inputs must be precised depending on the configuration used to map the function.
- If decomposition is used, the following modifications should be performed:
    - Update the function's declaration so that it becomes the declaration of the first sub-function (f1), by changing the mask and the used inputs.

- A new cell declaration must be added to represent the second sub-function (f2).
    - The carry-in is the output of f1.
    - The carry-out is a new wire added to the file.
    - The mask reflects the logical behavior of f2.
    - The inputs are the ones used for f2.
- All successors of the decomposed function f must be updated so that they use the output of f2 instead of f.

It might be relevant to mention that in this mapping technique, all available functions are tested for potential mappability using the arithmetic mode of the ALMs. Whenever such mappability is achieved, the function is mapped onto the carry-chain. This process is done opportunistically so that every chance of mappability is seized. However, that might not be the optimal way of mapping onto carry-chains if the overall objective is to maximize the reduction of used routing resources while maintaining the area overhead under control. In such cases, it might be more efficient to use some algorithmic intelligence to choose the functions that will be placed onto the chains, since, as the experimental results proved, routing and placement might add some more area overhead to the original overhead resulting from the mapping process.

Chapter Five

A Generic Software Platform

Combining all the previously proposed techniques and approaches one final tool was generated exploiting the flexibility of the structures on which these techniques are applied. This generic software platform can analyze, test various basic-unit structures and compare their performances on particular logic circuits depending on criteria specified by the user. Such structures may vary from currently available FPGA architectures to customized theoretical structures well-suited for a specific design(s). This tool can also propose particular cell structures to map logic circuits while respecting the user's constraints and insuring the optimization of specific parameters.

5.1 The need for such platform

It is previewed that this tool will be useful for two main parties:

I. Current FPGA companies' designers, especially Altera's since the main idea was proposed as an enhancement on its Stratix FPGAs. This tool will allow these designers to explore either minor or major modifications onto the current logic cells architectures and compare these new architectures to the already

implemented ones looking for improvements in mappability, area, routing or any other preferred parameters.

II. Designers searching for logic cells' architectures that would best implement their designs or applications while optimizing some personally defined parameters. The search for such architectures does not have to be limited to the logic cells of devices currently available in the market but can also be extended to hypothetical logic cells with user-specified architectures.

5.2 The tool's capabilities

This tool is based on one main structure shown in Figure 19, to which all flexibility measures were added allowing it to cover a wide range of possible configurations and modified/new architectures.

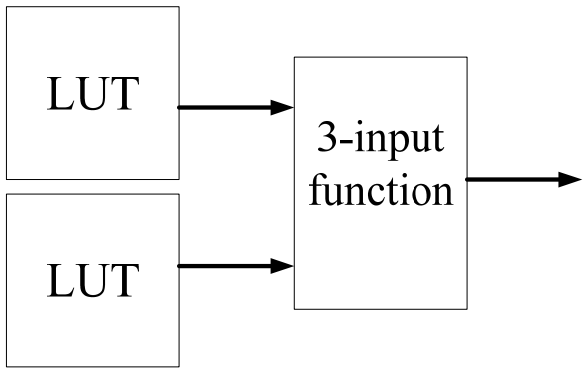

Figure 19: Tool's base configuration

The added flexibility to this base configuration can be performed at different levels where users can either select one flexibility level, multiple or even all levels during a single use. These levels can be quantified as follows:

1. The LUTs can have a variable and unlimited number of inputs as illustrated in Figure 20.

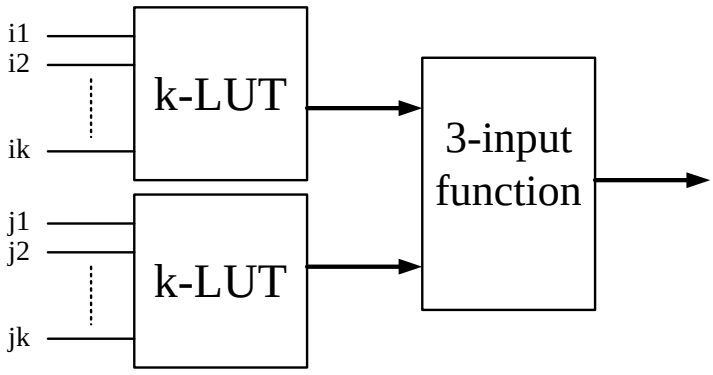

Figure 20: Configuration with k-input LUTs

2. The sizes of the LUT can be totally uncorrelated, where each LUT can have a different size (or number of inputs) independently of the second LUT, as shown in Figure 21.

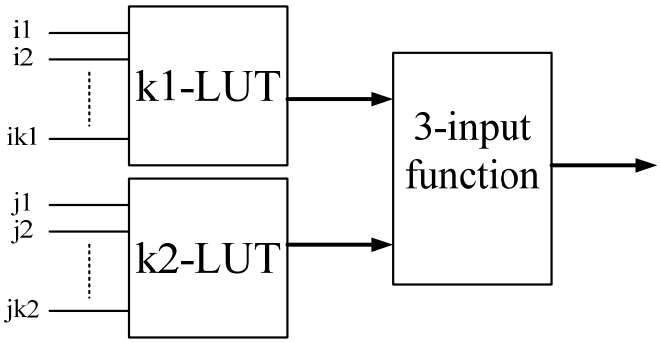

Figure 21: Configuration with variable-input LUTs (k1, k2)

3. The two LUTs can share inputs. The number of shared inputs can vary depending on the configuration, as long as it does not exceed the logical maximum, which is the minimum number of inputs the LUTs are using. This flexibility level is represented using Figure 22.

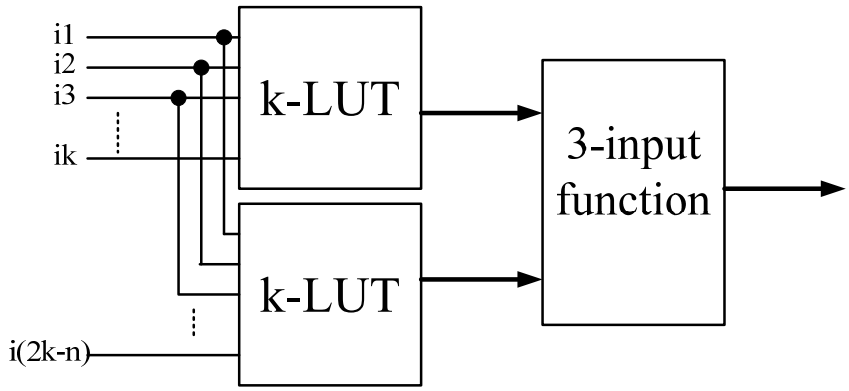

Figure 22: Configuration with k-input LUTs and n shared inputs

4. The 3-input function can perform any 3-input logical operation. The user is allowed to specify the behavior of this component by simply providing its truth table.

So if all flexibility levels are combined together, one would end up with a configuration of unbounded variable-input LUTs, with the possibility of having ‘n’ shared inputs, and any three input combinational function taking as inputs the outputs of the LUTs and an additional external input. This configuration is illustrated in Figure 23.

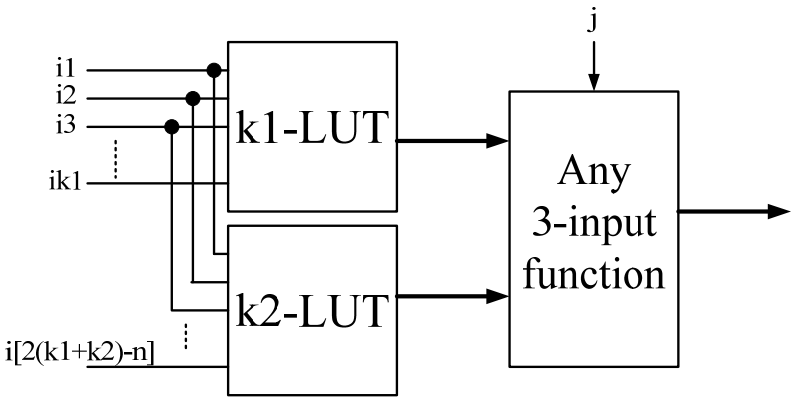

Figure 23: Fully flexible configuration

5.3 User's options

This tool allows to the user to perform a wide range of operations depending on his/her goals and objectives. However, the user is required to specify the configuration(s) on which the designs need to be tested, through a set of parameters.

5.3.1 - User-specified Parameters

Depending on the purpose of the conducted experiments, the user has to specify some parameters that would identify the objective and/or configuration(s) required. These parameters are:

- The first LUT's number of inputs 'k1'.

- The second LUT's number of inputs 'k2'.
- The number of shared inputs 'n'.
- The logical behavior of the three input function used (by providing its truth table).
- The objective(s) of the conducted experiment (one or more can be chosen):
    - Verifying mappability of functions on the chosen basic-unit structure.
    - Measuring the amount of used routing resources and its related congestion.
    - Quantifying the area needed to map a particular design on a particular structure.

5.3.2 - Modes of operation

Once the parameters have been chosen, users have to choose a mode of operation depending on their objectives. The chosen mode of operation will reflect the designer's end goal, as well as the analysis/conclusions that need to be derived from the experiment. The available modes of operation can be listed as follows:

- Mode 1: In this mode, the used structure has fixed overall configuration. In other terms, the LUTs' characteristics k1, k2 and n are chosen by the user as well as the 3-input function's behavior. This mode can be selected whenever the designer wishes to map a specific circuit on a particular architecture. Statistics can be generated, as always, for the chosen objectives, such as percent

mappability, area used by the circuit, etc. This mode can help designers assess the efficiency and performance of newly designed basic-unit structures.

- Mode 2: In this mode, the structure used is partially fixed in a way that the LUTs' characteristics k1, k2 and n must be predetermined and specified by the user. However, the tool tries all 256 possible 3-input functions ($2^{2^3} = 2^8 = 256$) and then reports statistics depending on the user selected objectives. This mode is useful when designers wish to perform some modifications on the currently available structures, trying to improve their performance.

- Mode 3: The structure, on which functions are mapped in this mode, has full LUTs flexibility while the 3-input function is predetermined by the user. So for a specific logical behavior at the output of the LUTs, the tool will try all possible LUT configurations by varying k1, k2 and n. Then it will report the configuration that best suits the designer's preferences. However, trying to reduce the runtime of the experiments, it is preferred to set a range from which k1, k2 and n can take their values. This mode was mainly used when exploring various configurations of the Stratix FPGAs logic cells, such as the 3-3-2, 4-4-3 and 2-2-1 structures while applying the enhanced mapping technique.

- Mode 4: This is a combination of modes 2 and 3, where the structure preserves it full flexibility. All 256 possible 3-input functions are tried at the outputs of the LUTs, while for every 3-input function selected, all possible LUT configurations

are explored by varying the values of k1, k2 and n (again with a specified range of possible values). It is important to note that this mode is very exhaustive and has a rather high runtime. However, this mode would reveal most useful when designers desire to widen the scope of their research and explore various possible modifications to available structures in FPGAs or even new out of the scope configurations.

- Mode 5: This mode takes two specific structures with particular configurations and compares these configurations according to some user-specified parameters. This mode can even map particular designs on both structures and then choose which structure is optimal for the user's application depending again on the user's priority and preferred optimality parameters. This particular mode enables designers to compare either two FPGA available structures trying to choose the one that better suits the application at hand, or to compare a totally theoretical structure with one that already exists for possible improvement in the current FPGA's architecture; or even, to compare two totally new hypothetical configurations, trying to choose which one maps the desired circuit with the most optimized designer-specific parameters.

Going back to the importance of such a tool to the potential users, the modes can be distributed in terms of the respective interest of each of the two main parties listed earlier, as follows:

I. Current FPGA companies' designers would be mainly interested in modes 1, 2, 3 and 5 where the tool allows them, through these modes, to explore minor

modifications on the logic cells used in their current respective FPGAs. It also helps them collect data and analyze statistics of high priorities, such as the routing resources congestion, total area used, etc. The tool also enables them to compare the performance of their currently used structures and the newly modified ones, helping them assess the importance of any modification.

II. Designers searching for logic cells' architectures that would best implement their circuits would probably be interested in all modes but mainly in modes 4 and 5. On one hand, mode 4 allows such designers to search a wide configuration space for a basic-unit structure that suits their preferences and specific applications with a minimized cost. Such a wide exploration might cover available FPGA's logic cells as well as hypothetical and purely theoretical structures. On the other hand, mode 5 allows those users to specifically choose two architectures and compare them according to some personally defined parameters. The two comparable architectures can be of two currently available FPGA cells, or two hypothetical basic-unit structures or even a combination of both.

All of these options provide the users with a high flexibility software platform that directly answers to their individual needs and preferences.

Chapter Six

Experimental Results

Several sets of experiments were conducted to properly reflect the benefits and improvement achieved through the various techniques and approaches proposed in this work. First, an overview of the benchmarks used will the presented highlighting each benchmark's characteristics. Then results of the mapping techniques applied will be listed along with data collected and analysis performed using the software platform.

6.1 Used Benchmarks

Experiments were conducted using the MCNC benchmarks, and more precisely, the Big 20 benchmarks. These benchmarks are known as the most used for conducting research experiments on new FPGA architectures and technologies.

As it is called, this set of benchmarks consists of 20 circuits distributed between combinational and sequential logic files. However since the proposed enhanced mapping technique and software platform support only combinational functions, experiments will be conducted using only combinational benchmarks.

Table 4 lists the used combinational benchmarks out of the big 20 and provides, for every benchmark the total number of used Adaptive Logic Modules (ALMs) after compiling it using the Altera Quartus II tool on a Stratix II FPGA. It also provides, for each benchmark the total number of interconnects used after Quartus' synthesis, mapping, placement and routing process.

Benchmark	Number of used ALMs	Number of used Interconnects
ex5p	197	3173
alu4	311	2978
apex4	362	4872
apex2	401	4042
seq	428	4713
ex1010	479	5084
spla	938	11303
pdc	968	11194

Table 4: Benchmarks' characteristics

The benchmarks in all statistics and experimental results will be listed in ascending order of the number of used ALMs after Quartus' compilation, as shown in Table 4.

Once the benchmark has been compiled using Quartus, the mapping netlist files can be parsed looking for synthesized logic function that would be mapped on logic cells. Such functions can have variable inputs ranging between two and seven. Table 5 provides the distribution of such functions for every benchmark. This means that, again for every

benchmark used, Table 5 lists the number of X-input functions available where X varies between two and seven.

Benchmark	Total functions	7-input functions	6-input functions	5-input functions	4-input functions	3-input functions	2-input functions
ex5p	344	1	44	128	68	59	44
alu4	542	5	74	229	76	91	67
apex4	606	1	120	237	101	101	46
apex2	668	2	128	251	111	103	73
seq	718	1	128	289	125	108	67
ex1010	733	5	208	288	122	67	43
spla	1524	6	285	561	253	196	223
pdc	1535	16	319	593	263	180	164
TOTAL	6670	37	1306	2576	1119	905	727

Table 5: Distribution of functions in every benchmark

6.2 Conducted Experiments

Several experiments have been conducted to simulate various cell configurations and test the enhanced mapping technique that was proposed. Some experiments were also conducted, using the proposed tool, for the sake of exploring wide ranges of cell configurations, new and previously available, where a comparison has been developed depending on some comparable parameters.

6.2.1 - Enhanced Mapping Technique's experimental results

In order to measure the performance of the new enhanced mapping technique and properly compare it to the currently used mapping process, the following experiments were preformed in order to generate respective data and statistics. These experiments were conducted using the previously specified MCNC benchmarks.

For each experiment and each benchmark, Quartus performs synthesis and mapping then the logic cell functions are extracted from the mapping netlist using the vqm parser. After that, the mapping technique is applied; a new netlist file is generated and fed back to quartus to perform placement and routing. Statistical data are extracted from both the mapping technique and Quartus.

Under this section, two different experiments were conducted following the same procedure; however differing in the specifications of the enhanced mapping technique.

I. The enhanced mapping technique is used with all its options as stated in Chapter 4; however, the decomposition of 5-or-less-input functions is not performed. In other words, if a 5-or-less-input function cannot be mapped on the 3-3-2 structure or on the 4-4-4 structure, then the function will not be mapped on the logic cell in arithmetic mode, but instead it will be mapped using the normal mode.

II. The enhanced mapping technique is used with all its options including the decomposition of 5-or-less-input functions.

Results of the enhanced mapping technique-experiment I (EMT-I) are presented in Table 6 and Table 7. Its respective comparative charts are illustrated in Figure 24 and Figure 25.

Benchmark	-Normal Mode- Number of used interconnects	-Arithmetic mode- Number of used interconnects	Percent improvement (decrease in used interconnects)
ex5p	3173	1989	37.31%
alu4	2978	2604	12.56%
apex4	4872	3677	24.53%
apex2	4042	3921	2.99%
seq	4713	4254	9.74%
ex1010	5084	4628	8.97%
spla	11303	10576	6.43%
pdc	11194	10531	5.92%
Maximum percent improvement			37.31%
Average percent improvement			13.56%
Minimum percent improvement			2.99%

Table 6: Experimental results of used interconnects after applying EMT-I

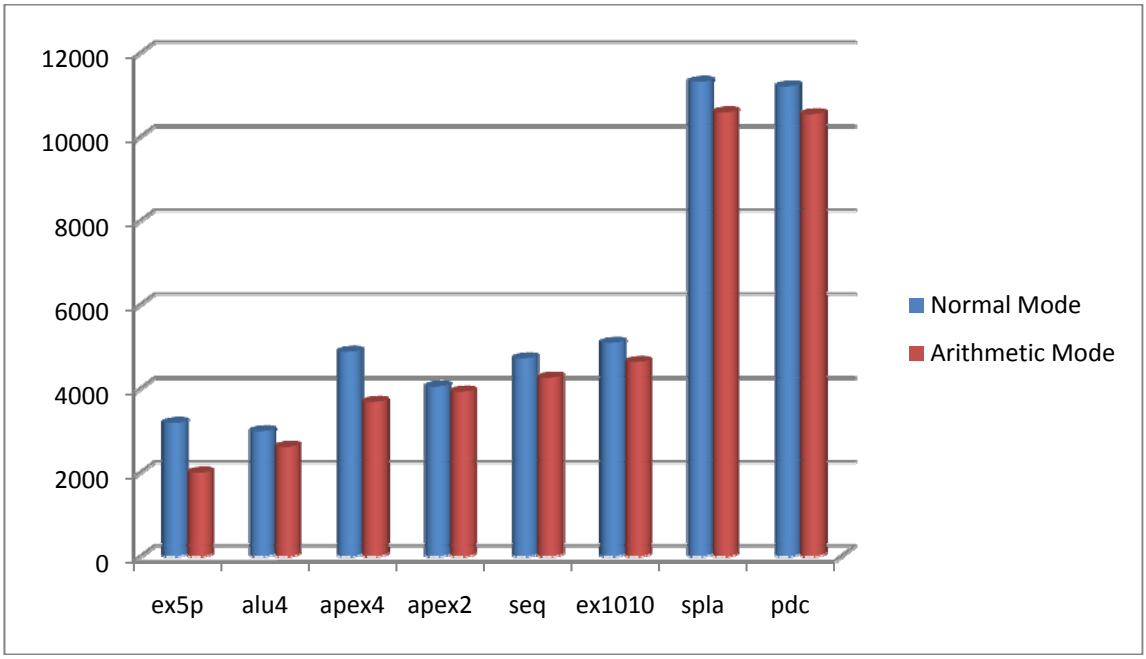

Figure 24: Representation of the improvement in used interconnects for EMTI

Benchmark	-Normal Mode- Number of used ALMs	-Arithmetic mode- Number of used ALMs	Percent overhead (increase in used ALMs)
ex5p	197	320	62.44%
alu4	311	378	21.54%
apex4	362	424	17.13%
apex2	401	540	34.66%
seq	428	523	22.2%
ex1010	479	550	14.82%
spla	938	1446	54.16%
pdc	968	1496	54.55%
Maximum percent overhead			62.44%
Average percent overhead			35.19%
Minimum percent overhead			14.82%

Table 7: Experimental results of used ALMs after applying EMT-I

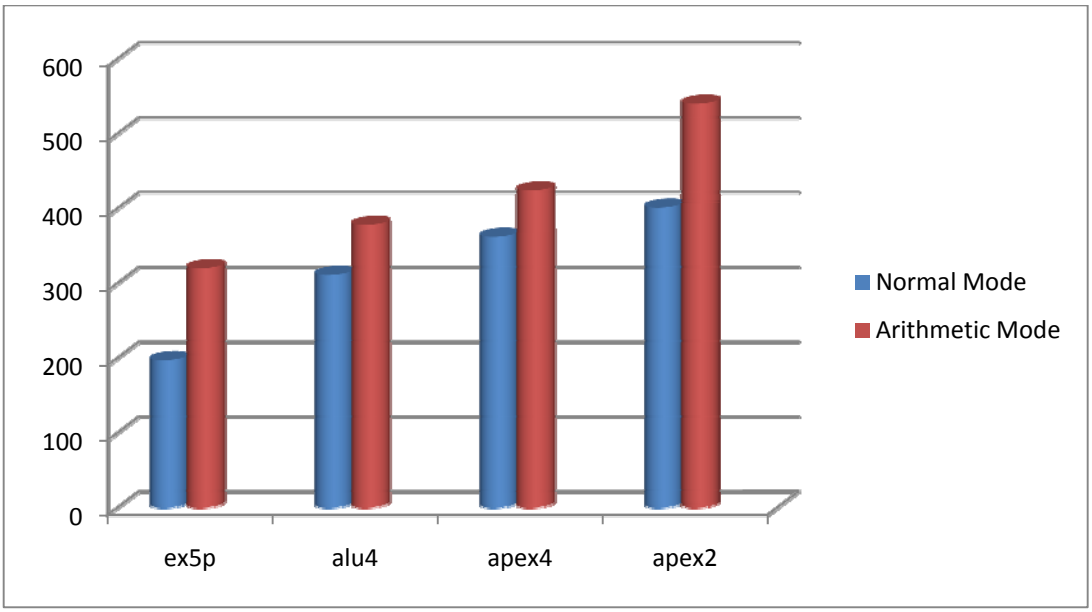

Figure 25: Representation of the overhead in used ALMs for EMTI

Since the enhanced mapping technique aims on reducing the number of used routing resources and the overall routing congestion by mapping using the arithmetic mode, the experiment measures the total number of used interconnects and compares it with the interconnects used while mapping using the normal mode.

This enhanced mapping technique (experiment I) achieves a reduction in the used interconnects with an average of 13.56% and a maximum of 37.31%, by mapping mom-arithmetic functions on the arithmetic mode instead of the normal mode.

However, and as expected, this reduction in the used routing resources is achieved at the expense of an increase in the number of used ALMs. Such an increase is acceptable since the area dedicated for routing resources is much larger than the one dedicated for logic components, so a decrease in overall used routing resources means a large area reduction. Nevertheless, one should keep in mind that the reported number of used ALMs is not highly accurate since these ALMs might be only partially used.

Results of the enhanced mapping technique-experiment II (EMT-II) are presented in Table 8 and Table 9. Their respective comparative charts are illustrated in Figure 26 and Figure 27.

Benchmark	-Normal Mode- Number of used interconnects	-Arithmetic mode- Number of used interconnects	Percent improvement (decrease in used interconnects)
ex5p	3173	2502	21.15%
alu4	2978	3161	- 6.15%
apex4	4872	3785	22.31%
apex2	4042	4297	-6.31%
seq	4713	4734	-0.45%
ex1010	5084	5359	-5.41%
spla	11303	6111	45.93%
pdc	11194	5435	51.45%
Maximum percent improvement			54.45%
Average percent improvement			15.32%
Minimum percent improvement			-6.31%
Maximum percent improvement over benchmarks with improvement			54.45%
Average percent improvement over benchmarks with improvement			31.25%
Minimum percent improvement over benchmarks with improvement			21.15%

Table 8: Experimental results of used interconnects after applying EMT-II

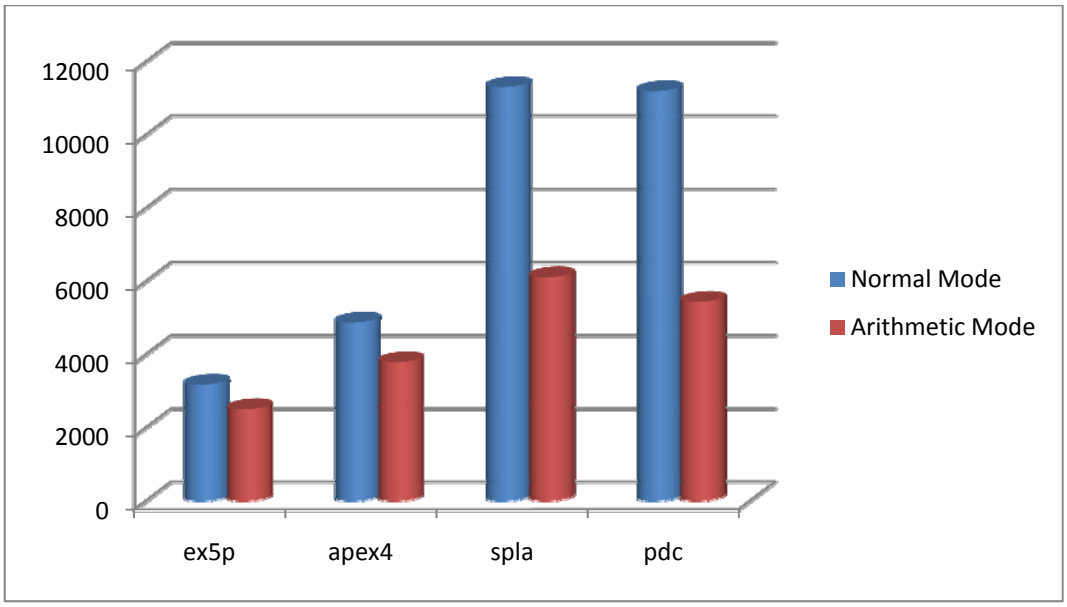

Figure 26: Representation of the improvement in used interconnects for EMTII

The results of this experiment are rather different that the ones of experiment I.

As shown in Table 8, the effect of decomposition on the overall used routing resources was not systematic or predictable. For some benchmarks the percent reduction in the used interconnects was the highest seen so far; however, for other benchmarks, the number of used interconnects increased after decomposition. This un-expected increase is to a certain extend the result of doubling the number of used logic cells per function during decomposition. Delivering inputs on all logic cells can be one contributor to the increase in the used external routing resources. Another reason for such an increase is due to Quartus' added logic cells to pass on the carry-out (onto the carry-chain) whenever a chain of functions has been interrupted.

Studying the reasons for these increases and understanding it might help as a future work in improving the performance of decomposition.

When only improvement cases are considered, the decrease of used routing resources is rather high and one major observation is that the improvement was mainly achieve on large benchmarks.

Such improvement, as before, introduces an overhead in the number of (partially or fully) used ALMs as shown in Table 8.

Benchmark	-Normal Mode- Number of used ALMs	-Arithmetic mode- Number of used ALMs	Percent overhead (increase in used ALMs)
ex5p	197	574	191.37%
alu4	311	646	107.72%
apex4	362	774	113.81%
apex2	401	816	103.49%
seq	428	939	119.39%
ex1010	479	959	100.21%
spla	938	1571	67.48%
pdc	968	1385	43.08%
Maximum percent overhead			191.37%
Average percent overhead			105.82%
Minimum percent overhead			43.08%
Maximum percent overhead over benchmarks with improvement			54.45%
Average percent overhead over benchmarks with improvement			103.94%
Minimum percent overhead over benchmarks with improvement			21.15%

Table 9: Experimental results of used ALMs after applying EMT-II

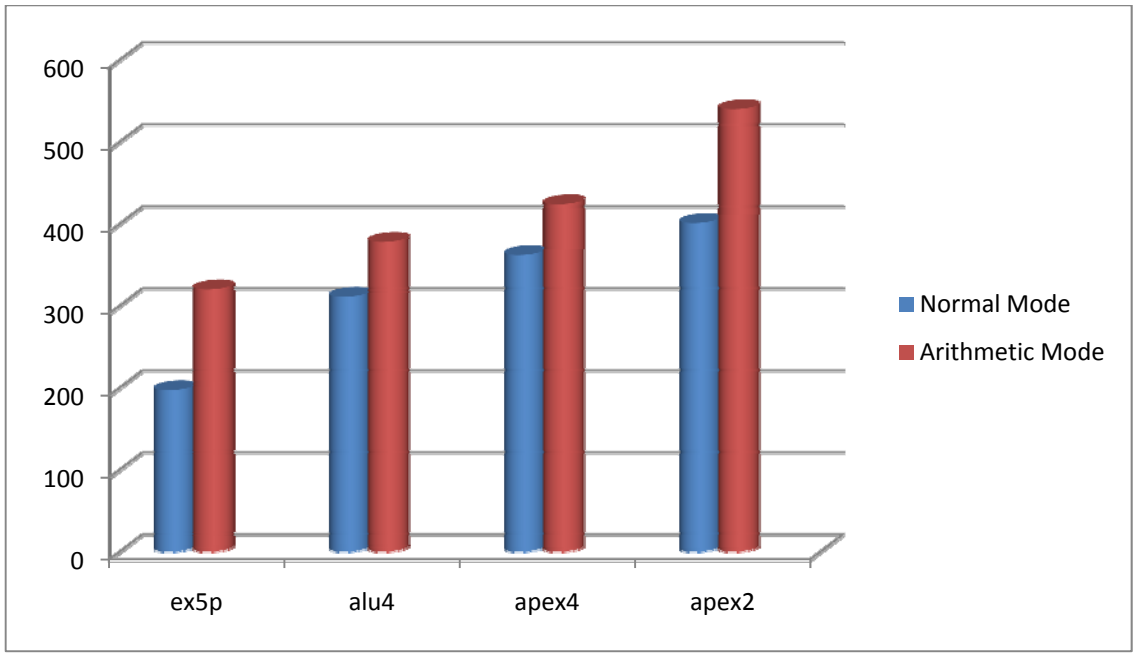

Figure 27: Representation of the overhead in used ALMs for EMTII

The number of used ALMs is almost doubled in this experiment, which is mainly due to the decomposition of 5-or-less-input functions. Instead of mapping such a function on one logic cell, decomposition is mapping it on two which, along with some Quartus' added overhead, results in such an increase in used ALMs.

Even with a high overhead, such technique would reveal useful when area is not a critical issue as opposed to power consumption, for example, which might be reduced through a minimization of the used routing resources.

6.2.2 - Statistics of mapping on specific configurations

The software platform proposed in Chapter 5 is used in this section, to test the mappability of logic functions on particular configurations and to generate related statistics. Two experiments were conducted under this scope, where each experiment goes through the benchmarks, parsing the logic functions and trying to map those functions onto a specific structure. The structures of interest are the 3-3-2 structure and the 4-4-4 structure; however any other structure can be chosen. The reported statistics highlight the percentage of directly mappable functions on the structure (i.e. without the use of decomposition) and then it provides the overall percentage of mappable functions after performing decomposition. It also reports the efficiency of the decomposition technique when applied on the set of unmappable functions.

For both experiments, statistics are divided in terms of the number of inputs of the functions to be mapped, which ensures clearer comparisons and a better understanding of the results.

Table 10 generates percentages over all benchmarks on the 3-3-2 structure, while the remaining Tables 11-18 report detailed statistics for every benchmark used.

Statistics for all benchmarks				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100%	77.00%	37.5%	63.19%
2-input functions	10.89%	100%	100%	0.0%
3-input functions	13.56%	100%	66.29%	100%
4-input functions	16.77%	100%	66.66%	100%
5-input functions	38.62%	80.12%	4.19%	79.25%
6-input functions	19.58%	24.57%	24.57%	0.0%
7-input functions	0.55%	-	-	-

Table 10: Mapping statistics over all benchmarks on a 3-3-2 structure

Statistics for alu4				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	78.96%	40.4%	64.7%
2-input functions	12.36%	100.0%	100.0%	0.0%
3-input functions	16.78%	100.0%	76.92%	100.0%
4-input functions	14.02%	100.0%	73.68%	100.0%
5-input functions	42.25%	79.91%	6.55%	78.5%
6-input functions	13.65%	14.86%	14.86%	0.0%
7-input functions	0.92%	-	-	-

Table 11: Mapping statistics over the alu4 benchmark on a 3-3-2 structure

Statistics for apex2				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100%	80.68%	46.55%	63.86%
2-input functions	10.92%	100.0%	100.0%	0.0%
3-input functions	15.41%	100.0%	96.11%	100.0%
4-input functions	16.61%	100.0%	87.38%	100.0%
5-input functions	37.57%	85.25%	1.59%	85.02%
6-input functions	19.16%	29.68%	29.68%	0.0%
7-input functions	0.29%	-	-	-

Table 12: Mapping statistics over the apex2 benchmark on a 3-3-2 structure

Statistics for apex4				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	75.08%	30.03%	64.38%
2-input functions	7.59%	100.0%	100.0%	0.0%
3-input functions	16.66%	100.0%	53.46%	100.0%
4-input functions	16.66%	100.0%	41.58%	100.0%
5-input functions	39.1%	70.88%	0.42%	70.76%
6-input functions	19.8%	32.5%	32.5%	0.0%
7-input functions	0.16%	-	-	-

Table 13: Mapping statistics over the apex4 benchmark on a 3-3-2 structure

Statistics for ex5p				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	83.72%	38.95%	73.33%
2-input functions	12.79%	100.0%	100.0%	0.0%
3-input functions	16.15%	100.0%	61.01%	100.0%
4-input functions	19.76%	100.0%	64.7%	100.0%
5-input functions	37.2%	84.37%	0.78%	84.25%
6-input functions	12.79%	20.45%	20.45%	0.0%
7-input functions	0.29%	-	-	-

Table 14: Mapping statistics over all benchmarks on a 3-3-2 structure

Statistics for ex1010				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	65.48%	31.65%	49.5%
2-input functions	5.86%	100.0%	100.0%	0.0%
3-input functions	9.14%	100.0%	64.17%	100.0%
4-input functions	16.64%	100.0%	55.73%	100.0%
5-input functions	39.29%	65.27%	6.25%	62.96%
6-input functions	28.37%	28.84%	28.84%	0.0%
7-input functions	0.68%	-	-	-

Table 15: Mapping statistics over the ex1010 benchmark on a 3-3-2 structure

Statistics for pdc				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100%	77.39%	36.09%	64.62%
2-input functions	10.68%	100.0%	100.0%	0.0%
3-input functions	11.72%	100.0%	55.55%	100.0%
4-input functions	17.13%	100.0%	69.96%	100.0%
5-input functions	38.63%	86.84%	6.74%	85.89%
6-input functions	20.78%	20.68%	20.68%	0.0%
7-input functions	1.04%	-	-	-

Table 16 Mapping statistics over the pdc benchmark on a 3-3-2 structure

Statistics for seq				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	78.69%	37.18%	66.07%
2-input functions	9.33%	100.0%	100.0%	0.0%
3-input functions	15.04%	100.0%	73.14%	100.0%
4-input functions	17.4%	100.0%	64.8%	100.0%
5-input functions	40.25%	80.96%	3.11%	80.35%
6-input functions	17.82%	24.21%	24.21%	0.0%
7-input functions	0.13%	-	-	-

Table 17: Mapping statistics over the seq benchmark on a 3-3-2 structure

Statistics for spla				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	78.28%	39.56%	64.06%
2-input functions	14.63%	100.0%	100.0%	0.0%
3-input functions	12.86%	100.0%	60.71%	100.0%
4-input functions	16.6%	100.0%	68.77%	100.0%
5-input functions	36.81%	80.92%	3.56%	80.22%
6-input functions	18.7%	23.5%	23.5%	0.0%
7-input functions	0.39%	-	-	-

Table 18: Mapping statistics over the spla benchmark on a 3-3-2 structure

The main observations derived from this set of results can be summarized as follows:

- All two input functions can be mapped on the structure without the need for any decomposition.
- With decomposition, all three and four-input functions were successfully mapped onto the structure.
- Without decomposition, an extremely low percentage of five-input functions were mappable on the structure.
- Decomposition maps a high percentage of five-input functions
- The percentage of overall mappable functions is highly boosted using the decomposition technique.

This experiment highlights the importance of the decomposition technique and the improvement it provides in terms of mappability. That is not only reflected from the increase in mapping percentages before and after decomposition, but also in the fact that the probability of a function's mappability increases as the number of inputs of the function decreases. Following this reasoning, decomposing into less-input functions increases the chances of mapping the function in question.

The second experiment was conducted using the same procedures as in the first experiment however now mapping is performed on a 4-4-4 structure. Table 19 provides statistics for all input functions and all benchmarks while Tables 20-27 report statistics for every benchmark.

Statistics for all benchmarks				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	72.89%	72.89%	0.0%
2-input functions	10.89%	100.0%	100.0%	0.0%
3-input functions	13.56%	100.0%	100.0%	0.0%
4-input functions	16.77%	100.0%	100.0%	0.0%
5-input functions	38.62%	80.27%	80.27%	0.0%
6-input functions	19.58%	3.29%	3.29%	0.0%
7-input functions	0.55%	-	-	-

Table 19: Mapping statistics over all benchmarks on a 4-4-4 structure

Statistics for alu4				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	77.85%	77.85%	0.0%
2-input functions	12.36%	100.0%	100.0%	0.0%
3-input functions	16.78%	100.0%	100.0%	0.0%
4-input functions	14.02%	100.0%	100.0%	0.0%
5-input functions	42.25%	79.91%	79.91%	0.0%
6-input functions	13.65%	6.75%	6.75%	0.0%
7-input functions	0.92%	-	-	-

Table 20: Mapping statistics over the alu4 benchmark on a 4-4-4 structure

Statistics for apex2				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	76.04%	76.04%	0.0%
2-input functions	10.92%	100.0%	100.0%	0.0%
3-input functions	15.41%	100.0%	100.0%	0.0%
4-input functions	16.61%	100.0%	100.0%	0.0%
5-input functions	37.57%	85.65%	85.65%	0.0%
6-input functions	19.16%	4.68%	4.68%	0.0%
7-input functions	0.29%	-	-	-

Table 21: Mapping statistics over the apex2 benchmark on a 4-4-4 structure

Statistics for apex4				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	69.14%	69.14%	0.0%
2-input functions	7.59%	100.0%	100.0%	0.0%
3-input functions	16.66%	100.0%	100.0%	0.0%
4-input functions	16.66%	100.0%	100.0%	0.0%
5-input functions	39.1%	70.88%	70.88%	0.0%
6-input functions	19.9%	2.5%	2.5%	0.0%
7-input functions	0.16%	-	-	-

Table 22: Mapping statistics over the apex4 benchmark on a 4-4-4 structure

Statistics for ex5p				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	81.1%	81.1%	0.0%
2-input functions	12.79%	100.0%	100.0%	0.0%
3-input functions	17.15%	100.0%	100.0%	0.0%
4-input functions	19.76%	100.0%	100.0%	0.0%
5-input functions	37.2%	84.37%	84.37%	0.0%
6-input functions	12.79%	0.0%	0.0%	0.0%
7-input functions	0.29%	-	-	-

Table 23: Mapping statistics over the ex5p benchmark on a 4-4-4 structure

Statistics for ex1010				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	57.7%	57.7%	0.0%
2-input functions	5.86%	100.0%	100.0%	0.0%
3-input functions	9.14%	100.0%	100.0%	0.0%
4-input functions	16.64%	100.0%	100.0%	0.0%
5-input functions	39.29%	65.27%	65.37%	0.0%
6-input functions	28.37%	1.44%	1.44%	0.0%
7-input functions	0.68%	-	-	-

Table 24: Mapping statistics over the ex1010 benchmark on a 4-4-4 structure

Statistics for pdc				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	73.94%	73.94%	0.0%
2-input functions	10.68%	100.0%	100.0%	0.0%
3-input functions	11.72%	100.0%	100.0%	0.0%
4-input functions	17.13%	100.0%	100.0%	0.0%
5-input functions	38.63%	86.84%	86.84%	0.0%
6-input functions	20.78%	4.07%	4.07%	0.0%
7-input functions	1.04%	-	-	-

Table 25: Mapping statistics over the pdc benchmark on a 4-4-4 structure

Statistics for seq				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	74.79%	74.79%	0.0%
2-input functions	9.33%	100.0%	100.0%	0.0%
3-input functions	15.04%	100.0%	100.0%	0.0%
4-input functions	17.4%	100.0%	100.0%	0.0%
5-input functions	40.25%	81.31%	81.31%	0.0%
6-input functions	17.82%	1.56%	1.56%	0.0%
7-input functions	0.13%	-	-	-

Table 26: Mapping statistics over the seq benchmark on a 4-4-4 structure

Statistics for spla				
	Contribution to the total number of inputs	% mapped functions	% mapped without <6-input decomposition	% mapped through <6-decomposition (out of the total initially unmappable functions
All functions	100.0%	74.73%	74.73%	0.0%
2-input functions	14.63%	100.0%	100.0%	0.0%
3-input functions	12.86%	100.0%	100.0%	0.0%
4-input functions	16.6%	100.0%	100.0%	0.0%
5-input functions	36.81%	81.28%	81.28%	0.0%
6-input functions	18.7%	3.85%	3.85%	0.0%
7-input functions	0.39%	-	-	-

Table 27: Mapping statistics over the spla benchmark on a 4-4-4 structure

The main observations on the statistics obtained from this experiment can be:

- Decomposition is useless if this structure is chosen. No functions require the use of the decomposition technique in order to me mapped.

- All two-input, three-input and four-input functions are fully mappable on the structure (without the need for decomposition).
- The structure itself has rather high mappability percentages (again without the use of the decomposition technique).

These experiments performed using the proposed generic software platform allow the designers to compare structures such as the 3-3-2 and the 4-4-4 and choose the one that better fits their preferences. Even though the 3-3-2 structure has higher overall mappability percentages, designers might prefer the 4-4-4 structure since it maps functions without the use of decomposition and as such without any theoretical increase in area and used basic-unit structures.

Chapter Seven

Conclusions

This work proposes some generic approaches that would enhance mapping on current FPGAs' cells as well as user-specified basic-unit structures. These approaches vary between variable cell configuration, Boolean Matching and decomposition techniques.

Experimental results show that the enhanced mapping technique manages to reduce the congestion of the routing resources since it can decrease the used interconnects by about 13.5%, on average. Moreover, this achievement can be even improved if a decomposition technique is embedded inside the enhanced mapping approach in order to increase the percentage of mappable functions and as such decrease the used interconnects. On large benchmarks, this newly-enhanced approach manages to decrease the used routing resource by up to half the amount that would usually be needed. Such a decrease in the congestion of the routing connections might stimulate a decrease in area and power consumption.

Furthermore, the proposed generic unit-based software platform provides the designer with high flexibility in the search for a structure that optimizes specific user-defined parameters while implementing particular applications. This tool, through its numerous modes and options, allows the designers to search the currently available architectures as well as a wide search-space of hypothetical structures in order to choose the structure

that fully implements the required circuit while abiding by the designer's predefined conditions and preferences. Not to forget that this tool also enables current FPGA companies' designers to perform minor modifications on their current cell structures, simulate and compare the performance of these modified cells to the performance of the current state-of-the-art FPGAs. Doing so might allow these companies to locate a minor modification to their logic cells that might substantially improve the performance of their FPGAs.

Future work can tackle various aspects of the proposed approaches. On one side, some intelligence needs to be added to the enhanced mapping technique whenever functions are mapped on ALMs in arithmetic mode. This intelligence might be in form of an algorithm that heuristically chooses the functions to be chained as opposed to the random approach that is currently used. Such a chain-selection algorithm would further reduce the congestion of the routing resources while minimizing the overhead in area and used ALMs. On a different angle, the proposed mapping technique is performed after going through synthesis using tools such as Quartus. These tools are not aware of the mode of operation of the proposed mapping technique and as such synthesis is performed as a general step and is not optimized for this specific mapping technique. So a possible future work would be in embedding both synthesis and mapping into one tool that performs both operations with some knowledge and intelligence so that synthesis generates functions in a way that ensures their mappability depending on the configuration of the used structure.

Bibliography

[1] P. Jamieson, W. Luk and S. Wilton, “An energy and power consumption analysis of FPGA routing architectures,” in *Int. Conf. Field-Programmable Tech.*, Sydney, 2009, pp.324-327.

[2] F. Li, D. Chen, L. He, and J. Cong, “Architecture evaluation for power-efficient FPGAs,” in *ACM/SIGDA Int. Symp. Field Programmable Gate Arrays,* New York, 2003, pp. 175-184.

[3] *Altera Inc. Stratix II device handbook*. [Online]. Available: http://www.altera.com.

[4] *Altera Inc. Stratix III device handbook.* [Online]. Available: http://www.altera.com.

[5] *Altera Inc. Stratix IV device handbook*. [Online]. Available: http://www.altera.com.

[6] S. Hauck, T. Fry, and M. Hosler, “High-performance carry chains for FPGA's,” in *Proc. 1998 ACM/SIGDA 6th Int. Symp. Field Programmable Gate Arrays,* New York, 1998, pp. 223-233.

[7] *Altera Inc. Altera QuartusII handbook*. [Online]. Available: http://www.altera.com.

[8] A. Ling, D.Singh, and S. Brown, “FPGA technology mapping: A study of optimality,” in *Proc. 42nd Annu. Design Automation Conf.*, New York, 2005, pp. 427-432.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton, “Improvements to technology mapping for LUT-based FPGAs,” in *Proc. 2006 ACM/SIGDA 14th Int. Symp. Field Programmable Gate Arrays*, New York, 2006, pp. 41-49.

[10] A. H. Farrahi and M. Sarrafzadeh, “FPGA technology mapping for power minimization,” in *4th Int. Workshop Field-Programmable Logic and Applications*, London, 1994, pp. 66-67.

[11] D. Chen, J. Cong, F. Li, and L. He, “Low-power technology mapping for FPGA architectures with dual supply voltages,” in *Proc. 2004 ACM/SIGDA 12th Int. Symp. Field Programmable Gate Arrays*, New York, pp. 109-117.

[12] J. Cong and Y. Ding, “FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs,” *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 13, pp. 1-12, Jan. 1994.

[13] A. H. Farrahi and M. Sarrafzadeh, “Complexity of the lookup-tale minimization problem for FPGA technology mapping,” *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 13, pp. 1319-1332, Nov. 1994.

[14] J. Lin, A. Jagannathan, and J. Cong, “Placement-driven technology mapping for LUT-based FPGAs,” in *Proc. 2003 ACM/SIGDA 11th Int. Symp. Field Programmable Gate Arrays*, New York, pp. 121-126.

[15] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, “Efficient SAT-based Boolean matching for FPGA technology mapping,” in *Proc. 43rd Annu. Design Automation Conf.*, New York, 2006, pp. 466-471.

[16] Y. Hu, V. Shih, R. Majumdar, and L. He, “Exploiting symmetry in SAT-based Boolean matching for heterogeneous FPGA technology mapping,” in *Proc. 2007 IEEE/ACM Int. Conf. Comput-Aided Design*, New Jersey, pp. 350-353.

[17] Z. Wei, D. Chai, A. Newton, and A. Kuehlmann, “Fast boolean matching with don't cares,” in *Proc. 7th Int. Symp. Quality Electronic Design*, Washington DC, 2006, pp. 346-351.

[18] G. Chen and J. Cong, “Simultaneous logic decomposition with technology mapping in FPGA designs,” in *Int. Symp. Field-Programmable Gate Arrays (FPGA)*, New York, 2001, pp. 48-55.

[19] T. S. Czajkowski and S. D. Brown, “Functionally linear decomposition and synthesis of logic circuits for FPGAs,” in *Proc. 45th Annu. Design Automation Conf.*, New York, 2008, pp. 18-23.

[20] M. T. Frederick and A. K. Somani, “Beyond the arithmetic constraint: Depth-optimal mapping of logic chains in LUT-based FPGAs,” in *Int. Symp. Field-Programmable Gate Arrays (FPGA)*, New York, 2008, pp. 37-46.

[21] H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Exploiting fast carry-chains of FPGAs for designing compressor trees,” in *Proc. 19th Int. Conf. Field-Programmable Logic and Applications*, Prague, 2009, pp. 242-249.

[22] H. Parandeh-Afshar, G. Zgheib, P. Brisk and P. Ienne, “Routing wire optimization through generic synthesis on FPGA carry chains,” presented at the Design Automation Conf., San Diego, CA, June 2011.

[23] H. Parandeh-Afshar, G. Zgheib, P. Brisk and P. Ienne, “Reducing the pressure on routing resources of FPGAs with generic logic chains,” in *Proc. 19th ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, New York, 2011, pp. 237-246.