ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη εργαλείου εκτίμησης κατανάλωσης ισχύος/ενέργειας για επαναδιαμορφούμενες αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΡΑΛΑΜΠΟΥ Ν. ΣΙΔΗΡΟΠΟΥΛΟΥ

Επιβλέπων : Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2010

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΈΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη εργαλείου εκτίμησης κατανάλωσης ισχύος/ενέργειας για επαναδιαμορφούμενες αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΡΑΛΑΜΠΟΥ Ν. ΣΙΔΗΡΟΠΟΥΛΟΥ

Επιβλέπων : Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2η Ιουλίου 2010.

.....	.....	.....
Δημήτριος Σούντρης	Κιαμάλ Πεκμετζή	Γιώργος Οικονομάκος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2010

.....

ΧΑΡΑΛΑΜΠΟΣ Ν. ΣΙΔΗΡΟΠΟΥΛΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Development of a Design Framework for Power/ Energy consumtion estimation in heterogeneous FPGA architectures

by

Charalampos N. Sidiropoulos

National Technical University of Athens

School of Electrical and Computer Engineering

Division of Computer Science

July 2010

Thesis Supervisor: Dimitrios Soudris

Title: Professor in the School of Electrical and Computer Engineering, National Technical University of Athens, Greece.

Abstract

The FPGA is an integrated circuit that contains many (64 to over 10,000) logic cells and hard blocks that can be viewed as standard components. The key to Fpgas' popularity is their ability to implement any circuit simply by being appropriately programmed (or reprogrammed). Most current reasonably sized FPGA user designs make use of hard specific-purpose heterogeneous blocks in addition to basic logic and routing fabric. These hard circuits are specific circuits included on an FPGA to perform specific logic functions, such as a multipliers or a memories, which could also be implemented using the base logic units and the routing fabric.

In order to investigate the quality of different FPGA architectures, one needs CAD tools capable of automatically implementing circuits in each FPGA architecture of interest. Once a circuit has been implemented in an FPGA architecture, one next needs accurate area, delay and power models to evaluate the quality (speed achieved, area required, power consumed) of the circuit implementation in the FPGA architecture under test.

Power is especially a concern in Field-Programmable Gate Arrays (FPGAs). The post-fabrication flexibility in these devices is provided using a large number of prefabricated routing tracks and programmable switches. These tracks can be long, and can consume a significant amount of energy every time they switch. In addition, the programmable switches add capacitance to each track; this further increases the power dissipation of FPGAs. Also the generic logic structures that are at the heart of every FPGA consume more power than the dedicated circuitry that would be found on an ASIC.

In this thesis frameworks and tools of FPGA design are analyzed and a complete framework its proposed for power estimation in heterogeneous FPGAs, named **NAROUTO**, along with a **Heterogeneity Support Toolset** (HST). With this framework one can explore different heterogeneous FPGA architectures in terms of delay , area and power, and the HST toolset can be used in collaboration with other academic tools for further research.

Keywords < Heterogeneous FPGA, Framework, Design Flow, Power Estimation, NAROUTO, Heterogeneity Support Toolset, HBVPR, HBT-Vpack>

Contents

1 Introduction **17**
- 1.1 What is an Fpga . . . 17
- 1.2 Advantages - disadvantages of FPGAs over ASIC . . . 19
- 1.3 Fpga Fabric . . . 21
- 1.4 CAD tools, design of an FPGA . . . 28
- 1.5 Organization of the Chapters . . . 31

2 State of the Art **33**
- 2.1 Standalone tools . . . 33
  - 2.1.1 Logic Synthesis and Technology mapping tools . . . 33
    - 2.1.1.1 MVSIS . . . 33
    - 2.1.1.2 ABC . . . 36
  - 2.1.2 Clustering, Place and Route tools . . . 40
    - 2.1.2.1 VPR 4.30 . . . 40
    - 2.1.2.2 Heterogeneous framework VPR 5.0.2 . . . 43
  - 2.1.3 Powermodel Framework . . . 47
  - 2.1.4 Nettovqm . . . 51
  - 2.1.5 Convert_arch_to_xml . . . 51
  - 2.1.6 VPR 5.0 with power estimation . . . 52
- 2.2 Existing Frameworks . . . 53
  - 2.2.1 Meander . . . 53
    - 2.2.1.1 What is Meander . . . 53
    - 2.2.1.2 Meander Design Flow . . . 56
  - 2.2.2 Quartus . . . 61
    - 2.2.2.1 What is Quartus . . . 61
    - 2.2.2.2 Quartus design flow . . . 61
  - 2.2.3 ALLIANCE . . . 70
    - 2.2.3.1 What is ALLIANCE . . . 70

2.2.3.2 ALLIANCE Design Flow 70

3 Proposed Framework **75**
3.1 Quartus, edif 78
3.2 Quartus, hierarchical blif 79
3.2.1 BLIF format 79
3.2.2 BLIF output from Quartus 81
3.3 Powermodel Ace 86
3.4 HBT-Vpack 89
3.5 Heterogeneity Support Toolset 96
3.5.1 Practical problems in Heterogeneity support 96
3.5.2 HST Tools 97
3.5.2.1 Blackbox-aware technology mapping 97
3.5.2.2 Blackbox Packing 98
3.5.2.3 Multiplexer 103
3.5.2.4 Net2xml 107
3.5.2.5 Activity_Updater 111
3.6 HBVPR 112
3.6.1 Power estimation 112
3.6.1.1 Dynamic Power 112
3.6.1.2 Short-Circuit Power 116
3.6.1.3 Leakage power 117
3.6.1.4 Blackboxe's power estimation 117
3.6.2 Placement and routing 118
3.7 DAGGER 121

4 Benchmarking **125**
4.1 Benchmarks information 125
4.2 Heterogeneity Support Toolset results 127
4.3 HBVPR results 132
4.3.1 Non-packed 134
4.3.2 Area and delay optimized architecture after level 1, 2 packing at 180nm 137
4.3.3 Area and delay optimized architecture after level 1, 2 packing at 130nm 145
4.3.4 Area and delay optimized architecture after level 1, 2 packing at 90nm 148

4.3.5 Area and delay optimized architecture after level 1, 2 packing at 65nm 152
4.3.6 Area and delay optimized architecture after level 1, 2 packing at 45nm 156
4.3.7 Comparison results 160

5 Conclusions **165**

6 Future work **167**

List of Figures

1.1 Abstract design of a generic FPGA 18
1.2 Applications of FPGAs [2] 19
1.3 A 3-LUT schematic (a) and the corresponding 3-LUT symbol and truth table (b) for a logical XOR 21
1.4 A simple lookup table logic block. 22
1.5 An island-style architecture with connect blocks and switch boxes. 23
1.6 A connection block. 24
1.7 A switch block. 25
1.8 Local (direct) connections and L2 connections augmenting a switched interconnect. 26
1.9 A typical FPGA mapping flow. 29

2.1 A cluster-based logic block. 41
2.2 Initial Random Placement 43
2.3 Final Placement 43
2.4 Completely (Detailed) Routed Circuit 44
2.5 Close-up View of the FPGA Routing Architecture 44
2.6 Heterogenous blocks in an FPGA. 45
2.7 Island style FPGA. 47
2.8 H-tree clock network. 48
2.9 Powermodel framework. 48
2.10 Meander design flow. 54
2.11 DAGGER required input files. 61
2.12 Quartus II design flow. 62
2.13 Quartus II design entry flow. 64
2.14 Quartus II synthesis flow. 64
2.15 Quartus II place and route flow. 65
2.16 Quartus II Power Analysis flow. 66

2.17 Quartus II Simulation flow. 68
2.18 Quartus II Programming flow. 69
2.19 How the tools are linked on the data structures in ALLIANCE design Flow. 71

3.1 NAROUTO framework design Flow. 76
3.2 The detailed design flow for power estimation of NAROUTO framework. 77
3.3 Design entry flow in NAROUTO taken from Quartus design flow (red highlighted area). 78
3.4 Example of an FPGA Routing Segment. 113
3.5 Schematic of a logic block. 114
3.6 RC Ladder network corresponding to a clock tree with two clock buffers. 117
3.7 Top view of a heterogeneous design with 8 blackboxes. The different types of blackboxes are shown with different colors (VPR only supports 6 different colors) and have height three times the basic soft logic cluster. 119
3.8 Horizontal routing tracks pass through at every grid location, even from blackboxes 120
3.9 Blackbox connections to the routing channel (red squares) . . 121
3.10 Blackbox connections to clbs (green blackbox is connected to blue and red clbs) 122

4.1 Oc_hdlc benchmark placement in HBVPR without Blackbox-aware technology mapping. The blackbox positions are marked with the red squares. 128
4.2 Oc_hdlc benchmark placement in HBVPR after level 1 Black-box packing. 129
4.3 Oc_hdlc benchmark placement in HBVPR after level 2 Black-box packing. 130
4.4 Comparison between non-packed and packed designs in terms of area requirement. The blue area of each bar represents the percent area savings of packed designs. The total area required by each design is shown in CLBs number, the number in the red area is for packed designs and the blue for non-packed . . 142
4.5 Comparison between non-packed and packed designs in terms of critical path delay. 143

4.6 Comparison between non-packed and packed designs in terms of total power consumption. 144
4.7 Comparison of different technologies in terms of critical path delay. 162
4.8 Comparison of different technologies in terms of power consumption. 163
4.9 Comparison in terms of power consumption for the same operating frequency. 164

6.1 Meander and NAROUTO design frameworks. 168

List of Tables

2.1 Comparison betwen Meander, Toronto university framework and ALLIANCE framework 55
2.2 Comparison beetwen EX-VPR and VPR 59

3.1 Ac2 file example 92
3.2 Fun file example 93
3.3 Net file blackboxe's example 94
3.4 Blackbox_Profiler log file 98
3.5 Blackbox example before (top) and after (bottom) blackbox-aware technology mapping 100
3.6 Blackbox example after level 1 packing (top) and after level 2 packing (bottom) 102
3.7 Blackbox example before (top) and after (bottom) I/O multiplexing 105
3.8 Extra clbs from blackbox I/O multiplexing 106
3.9 Log file of Multiplexer tool 107

4.1 Benchmark information 126
4.2 Number of blackboxes before and after packing level 1 and 2. In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so in level 2 there is a "-" 127
4.3 Estimated RAM bits each blackbox in each design has after Level 1 packing, the form is (No. of blackboxes× Ram bits) . 131
4.4 Pin multiplexing 131
4.5 Area results benchmarks without blackbox-aware technology mapping. 134
4.6 Delay results (in seconds) of benchmarks without blackbox-aware technology mapping. 134

4.7 Total power dissipation results (in Watts) of benchmarks without blackbox-aware technology mapping. 135
4.8 Leakage power dissipation results (in Watts) of benchmarks without blackbox-aware technology mapping. 135
4.9 Energy consumption results (in Joules) of benchmarks without blackbox-aware technology mapping. 136
4.10 Summarized results of benchmarks without blackbox-aware technology mapping. 136
4.11 Area results from benchmarks in 180nm optimized for area and delay FPGA architecture. 138
4.12 Delay results from benchmarks in 180nm optimized for area and delay FPGA architecture. 138
4.13 Total power dissipation results from benchmarks in 180nm optimized for area and delay FPGA architecture. 139
4.14 Leakage power dissipation results from benchmarks in 180nm optimized for area and delay FPGA architecture. 140
4.15 Energy consumption from benchmarks in 180nm optimized for area and delay FPGA architecture. 140
4.16 Summarized results from benchmarks in 180nm optimized for power FPGA architecture. 141
4.17 Area results from benchmarks in 130nm FPGA architecture. . 145
4.18 Delay results from benchmarks in 130nm FPGA architecture. 146
4.19 Total power dissipation results from benchmarks in 130nm FPGA architecture. 146
4.20 Leakage power dissipation results from benchmarks in 130nm FPGA architecture. 147
4.21 Energy consumption (in Joules) from benchmarks in 130nm FPGA architecture. 147
4.22 Summarized results from benchmarks in 130nm FPGA architecture. 148
4.23 Area results from benchmarks in 90nm FPGA architecture. . . 149
4.24 Delay results from benchmarks in 90nm FPGA architecture. . 149
4.25 Total power dissipation results from benchmarks in 90nm FPGA architecture. 150
4.26 Leakage power dissipation results from benchmarks in 90nm FPGA architecture. 150
4.27 Energy consumption (in Joules) from benchmarks in 90nm FPGA architecture. 151

4.28 Summarized results from benchmarks in 90nm FPGA architecture. 152
4.29 Area results from benchmarks in 65nm FPGA architecture. . . 153
4.30 Delay results from benchmarks in 65nm FPGA architecture. . 153
4.31 Total power dissipation results from benchmarks in 65nm FPGA architecture. 154
4.32 Leakage power dissipation results from benchmarks in 65nm FPGA architecture. 154
4.33 Energy consumption (in Joules) from benchmarks in 65nm FPGA architecture. 155
4.34 Summarized results from benchmarks in 65nm FPGA architecture. 156
4.35 Area results from benchmarks in 45nm FPGA architecture. . . 157
4.36 Delay results from benchmarks in 45nm FPGA architecture. . 157
4.37 Total power dissipation results (in Watts) from benchmarks in 45nm FPGA architecture. 158
4.38 Leakge power dissipation results (in Watts) from benchmarks in 45nm FPGA architecture. 158
4.39 Energy consumption (in Joules) from benchmarks in 45nm FPGA architecture. 159
4.40 Summarized results from benchmarks in 45nm FPGA architecture. 160

List of Algorithms

1 hb_for_ace 87
2 Blackbox_Profiler 97
3 Blackbox Packing level 1 99
4 Blackbox_Packing Level 2 101
5 Multiplexer 104
6 Activity_Updater 111

Chapter 1

Introduction

1.1 What is an Fpga

The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnect form a fabric of basic building blocks for logic circuits (figure 1.1). Complex designs are created by combining these basic blocks to create the desired circuit.

Field Programmable means that the FPGA's function is defined by a user's program rather than by the manufacturer of the device. A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the FPGA's function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either imprinted permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits.

As process geometries have shrunk into the deep-submicron region, the logic capacity of FPGAs has greatly increased, making FPGAs a viable implementation alternative for larger and larger designs (figure 1.2). Due to

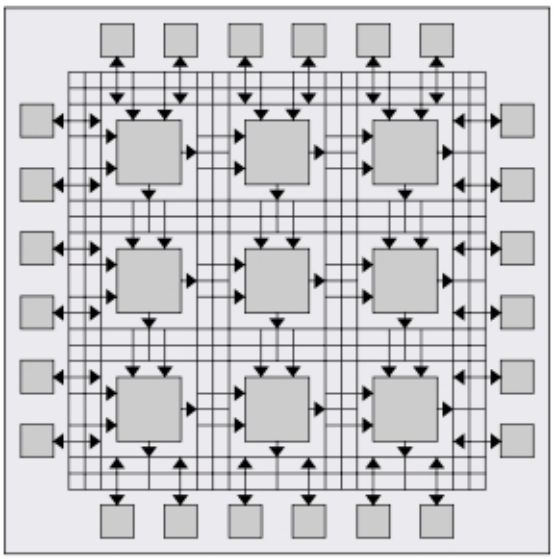

Figure 1.1: Abstract design of a generic FPGA

these technological advantages since their introduction in 1984 FPGAs have become one of the most popular implementation media for digital circuits and have grown into an industry of 2.75 billion dollar per year (2010 estimates).

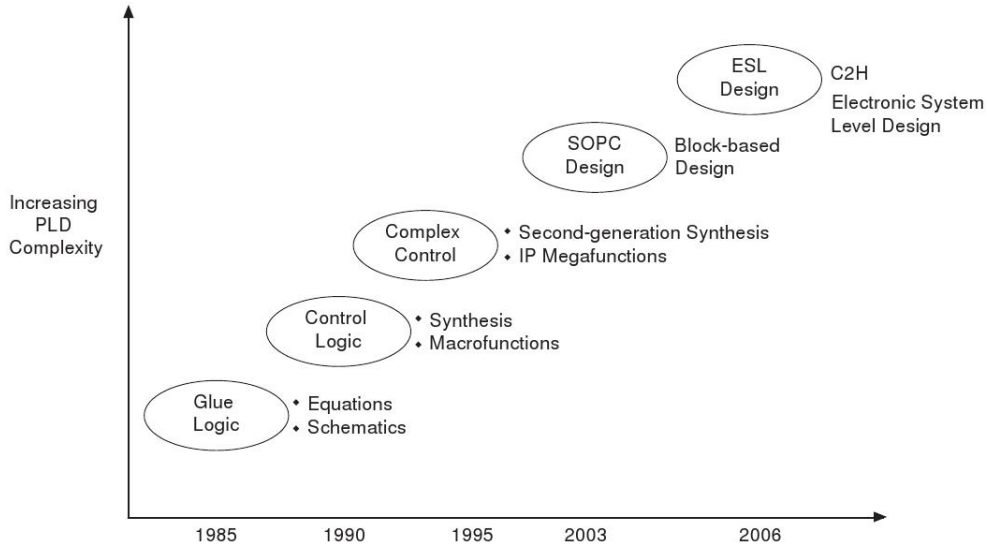

Figure 1.2: Applications of FPGAs [2]

1.2 Advantages - disadvantages of FPGAs over ASIC

The key to Fpgas' popularity is their ability to implement any circuit simply by being appropriately programmed (or reprogrammed). Other circuit implementation options, such as Standard Cells or Mask-programmed Gate Arrays (MPGAs), require that a different VLSI chip be newly fabricated for each design. The use of a standard FPGA over these custom technologies, has two key benefits: lower non-recurring engineering (NRE) costs, and faster time-to-market.

To implement a circuit in a MPGA or with Standard Cells, one sends the completed design to a silicon foundry which manufactures a chip to implement exactly (and only) that design. The non-recurring engineering (NRE) fees to have the first chip manufactured are typically between $100000 and $250000; these fees cover the costs of making lithography masks for the circuit and of running a new design through the fabrication plant. On the other hand, a design is implemented in an FPGA simply by programming the FPGA to have the desired functionality , so there are no NRE costs. This makes Fpgas the lowest cost implementation medium for small and medium

volume designs.

Time-to-market is the other key advantage of FPGAs. Full fabrication typically takes 6 - 8 weeks. If problems are found in the finished chip it must be thrown away, and designers have to wait another 6 - 8 weeks to fabricate a corrected design. FPGAs on the other hand can be programmed in seconds and once the chip is tested it can be reprogrammed in mere minutes. Sometimes reprogramming is merely a bug fix to correct faulty behavior, or it is used to add a new feature. Other times, it may be carried out to reconfigure a generic computation engine for a new task, or even to reconfigure a device during operation to allow a single piece of silicon to simultaneously do the work of numerous special-purpose chips. With today's demanding market for short product cycles, the time-to-market advantage this provides is often compelling.

FPGA programmability carries a price, however. In ASIC (MPGAs and Standar Cells) circuitry is interconnected with metal wires. FPGAs, in contrast, must connect circuitry via programmable switches. These switches have higher resistance than metal wires and add significant capacitance to connections, thus reducing circuit speed. The switches also take up more area than metal wires would, so an FPGA must be considerably larger than an MPGA to implement the same design. Due to the added capacitance to connections, FPGAs consume more power than the MPGA implementing the same circuit. Compared to ASICs, they may be 3 to 25 times worse in terms of area, delay, power and performance. The larger size of FPGA circuitry makes FPGA implementations more expensive than MPGAs for high volume designs, the limited speed and the power consumption precludes their use in very high speed and power sensitive designs. These differences mandate research into new FPGA architectures in order to reduce these speed, density and power penalties. In addition, because the FPGA marketplace is highly competitive, each FPGA manufacturer is constantly searching for better FPGA architectures in order to gain a speed, density and power advantage [1].

1.3 Fpga Fabric

All FPGAs are composed of three fundamental components : logic blocks, I/O blocks and programmable routing. The logic block used in an FPGA strongly influences the FPGA speed and area efficiency. Most current commercial FPGAs use logic blocks based on Look up tables (LUTs). From a circuit implementation perspective, a LUT can be formed simply from an N:1 (N-to-one) multiplexer and an N-bit memory. From the perspective of digital logic, a LUT simply enumerates a truth table. Therefore, using LUTs gives an FPGA the generality to implement arbitrary digital logic. Figure 1.3 shows a typical N-input lookup table that we might find in today's FPGAs.

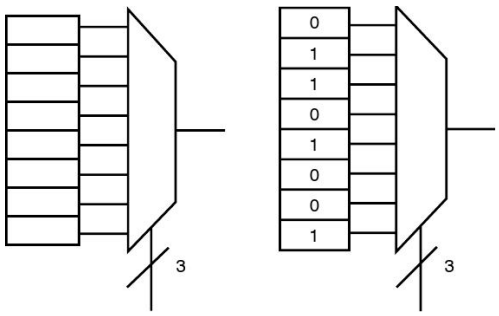

Figure 1.3: A 3-LUT schematic (a) and the corresponding 3-LUT symbol and truth table (b) for a logical XOR

LUT

The LUT can compute any function of N inputs by simply programming the lookup table with the truth table of the function we want to implement. As shown in the figure, if we wanted to implement a 3-input exclusive-or (XOR) function with our 3-input LUT (often referred to as a 3-LUT), we would assign values to the lookup table memory such that the pattern of select bits chooses the correct row's "answer". Thus, every "row" would yield a result of 0 except in the four cases where the XOR of the three select lines yields 1. Of course, more complicated functions, and functions of a larger number of inputs, can be implemented by aggregating several lookup tables together. For example, one can organize a single 3-LUT into an 8 x 1 ROM, and if the values of the lookup table are reprogrammable, an 8 x 1 RAM. But the basic building block, the lookup table, remains the same.

Although the LUT has more or less been chosen as the smallest computational unit in commercially available FPGAs, the size of the lookup table in each logic block has been widely investigated. On the one hand, larger lookup tables would allow for more complex logic to be performed per logic block, thus reducing the wiring delay between blocks as fewer blocks would be needed. However, the penalty paid would be slower LUTs, because of the requirement of larger multiplexers, and an increased chance of waste if not all of the functionality of the larger LUTs were to be used. On the other hand, smaller lookup tables may require a design to consume a larger number of logic blocks, thus increasing wiring delay between blocks while reducing per-logic block delay.Current empirical studies have shown that the 4-LUT structure makes the best trade-off between area and delay for a wide range of benchmark circuits.

Additionally there is a simple single-bit storage element in our base logic block in the form of a D flip-flop. The output multiplexer selects a result either from the function generated by the lookup table or from the stored bit in the D flip-flop. This is shown in figure 1.4.

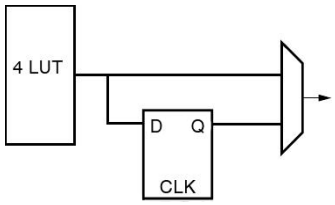

Figure 1.4: A simple lookup table logic block.

Interconnection

Current popular FPGAs implement what is often called island-style architecture. This design has logic blocks tiled in a two dimensional array and interconnected in some fashion. The logic blocks form the islands and "float" in a sea of interconnect. With this array architecture, computations are performed spatially in the fabric of the FPGA. Large computations are broken into 4-LUT-sized pieces and mapped into physical logic blocks in the array. The interconnect is configured to route signals between logic blocks appropriately. With enough logic blocks, we can make our FPGAs perform any kind of computation we desire.

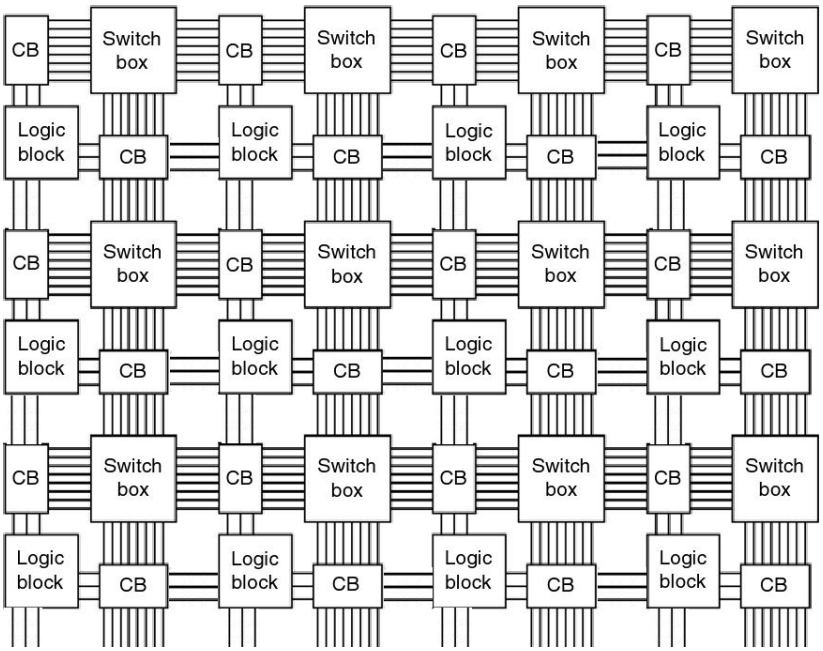

Figure 1.5: An island-style architecture with connect blocks and switch boxes.

In Figure 1.5 an island-style architecture FPGA is shown. Here the routing structure is generic and mesh-like. The logic block accesses nearby communication resources through the connection block, which connects logic block input and output terminals to routing resources through programmable switches, or multiplexers. The connection block (figure 1.6) allows logic block inputs and outputs to be assigned to arbitrary horizontal and vertical tracks, increasing routing flexibility.

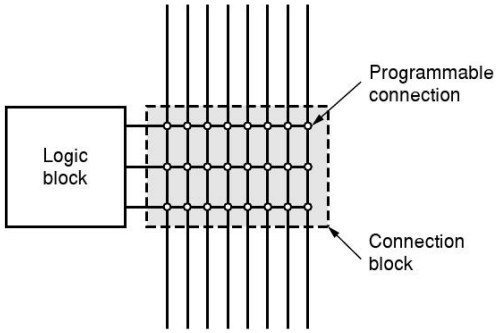

Figure 1.6: A connection block.

The switch block appears where horizontal and vertical routing tracks converge as shown in Figure 1.7. In the most general sense, it is simply a matrix of programmable switches that allow a signal on a track to connect to another track. Depending on the design of the switch block, this connection could be, for example, to turn the corner in either direction or to continue straight. The design of switch blocks is an entire area of research by itself and has produced many varied designs that exhibit varying degrees of connectivity and efficiency.

With this architecture, the concept of a segmented interconnect becomes more clear. For signals that need to travel longer distances, individual segments can be switched together in a switch block to connect distant logic blocks together. Think of it as a way to emulate long signal paths that can span arbitrary distances. The result is a long wire that actually comprises shorter segments. In this interconnect architecture introduction of connection blocks and switch boxes separates the interconnect from the logic, allowing long-distance routing to be accomplished without consuming logic block resources. To improve on our structure, we introduce longer-length wires. For instance, consider a wire that spans one logic block as being of length-1

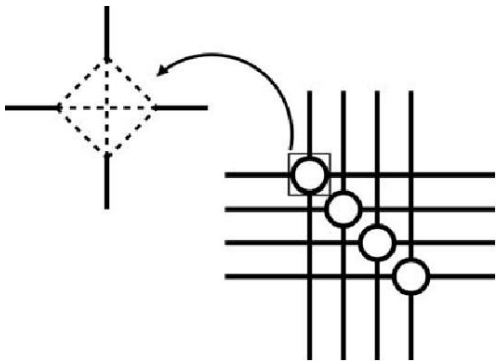

Figure 1.7: A switch block.

(L1). In some segmented routing architectures, longer wires may be present to allow signals to travel greater distances more efficiently. These segments may be length-4 (L4), length-8 (L8), and so on. The switch blocks (and perhaps more embedded switches) become points where signals can switch from shorter to longer segments. This feature allows signal delay to be less than O(N) when covering a distance of N logic blocks by reducing the number of intermediate switches in the signal path. Figure 1.8 illustrates augmenting the single-segment interconnect with two additional lengths: direct-connect between logic blocks and length-2 (L2) lines. The direct-connect lines leave general routing resources free for other uses, and L2 lines allow signals to travel longer distances for roughly the same amount of switch delay. This interconnect architecture closely matches that of the Xilinx XC4000 series of commercial FPGAs.

Programming Technologies

Each configurable element in an FPGA requires 1 bit of storage to maintain a user-defined configuration. For a simple LUT-based FPGA, these programmable locations generally include the contents of the logic block and the connectivity of the routing fabric. Configuration of the FPGA is accomplished through programming the storage bits connected to these programmable locations according to user definitions. For the lookup tables, this translates into filling it with 1s and 0s. For the routing fabric, programming enables and disables switches along wiring paths. There are many known methods for storing a single bit of binary information, the most popular

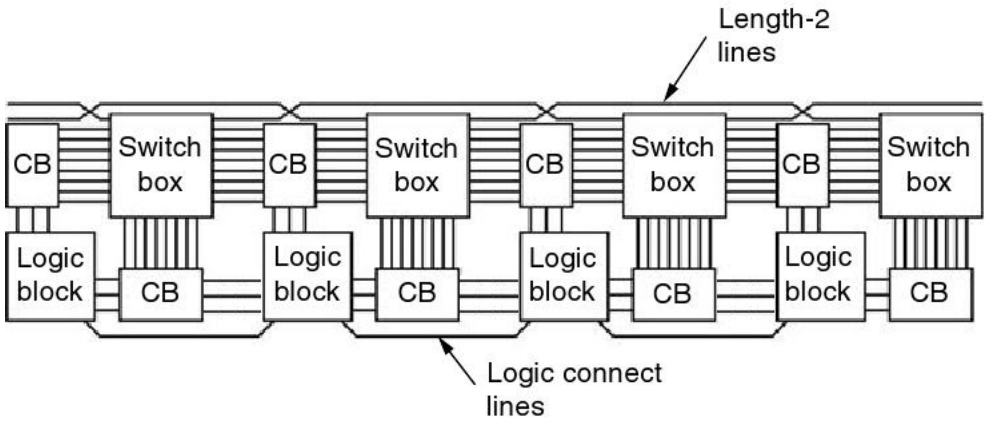

Figure 1.8: Local (direct) connections and L2 connections augmenting a switched interconnect.

methods used for FPGAs are SRAM, antifuse and Flash memory.

SRAM

The most widely used method for storing configuration information in commercially available FPGAs is volatile static RAM, or SRAM. This method has been made popular because it provides fast and infinite reconfiguration in a well-known technology. Drawbacks to SRAM come in the form of power consumption and data volatility. Compared to the other popular technologies, the SRAM cell is large (6-12 transistors) and dissipates significant static power because of leakage current. Another significant drawback is that SRAM does not maintain its contents without power, which means that at power-up the FPGA is not configured and must be programmed using off-chip logic and storage. This can be accomplished with a non-volatile memory store to hold the configuration and a micro-controller to perform the programming procedure. While this may seem to be a trivial task, it adds to the component count and complexity of a design and prevents the SRAM-based FPGA from being a truly single-chip solution.

Flash memory

Although less popular than SRAM, several families of devices use Flash memory to hold configuration information. Flash memory is different from SRAM in that it is nonvolatile and can only be written a finite number of times. The non-volatility of Flash memory means that the data written to it

remains when power is removed. In contrast with SRAM-based FPGAs, the FPGA remains configured with user-defined logic even through power cycles and does not require extra storage or hardware to program at boot-up. In essence, a Flash-based FPGA can be ready immediately. A Flash memory cell can also be made with fewer transistors compared to an SRAM cell. This design can yield lower static power consumption as there are fewer transistors to contribute to leakage current.

Drawbacks to using Flash memory to store FPGA configuration information stem from the techniques necessary to write to it. As mentioned, Flash memory has a limited write cycle lifetime and often has slower write speeds than SRAM. The number of write cycles varies by technology, but is typically hundreds of thousands to millions. Additionally, most Flash write techniques require higher voltages compared to normal circuits; they require additional off-chip circuitry or structures such as charge pumps on-chip to be able to perform a Flash write.

Antifuse

A third approach to achieving programmability is antifuse technology. Antifuse, as its name suggests, is a metal-based link that behaves the opposite of a fuse. The antifuse link is normally open (i.e. unconnected). A programming procedure that involves either a high-current programmer or a laser melts the link to form an electrical connection across it, in essence, creating a wire or a shortcircuit between the antifuse endpoints.

Antifuse has several advantages and one clear disadvantage, which is that it is not reprogrammable. Once a link is fused, it has undergone a physical transformation that cannot be reversed. FPGAs based on this technology are generally considered one-time programmable (OTP). This severely limits their flexibility in terms of reconfigurable computing and nearly eliminates this technology for use in prototyping environments. However, there are some distinct advantages to using antifuse in an FPGA platform. First, the antifuse link can be made very small, compared to the large multi-transistor SRAM cell, and does not require any transistors. This results in very low propagation delays across links and zero static power consumption, as there is no longer any transistor leakage current. Antifuse links are also not susceptible to high-energy radiation particles that induce errors known as single-event upsets, making them more likely candidates for space and military applications.

1.4 CAD tools, design of an FPGA

Three factors combine to determine the performance of an FPGA : the quality of the CAD tools used to map circuits into the FPGA, the quality of the FPGA architecture, and the electrical (i.e. transistor-level) design of the FPGA. In order to investigate the quality of different FPGA architectures, one needs CAD tools capable of automatically implementing circuits in each FPGA architecture of interest. Once a circuit has been implemented in an FPGA architecture, one next needs accurate area, delay and power models to evaluate the quality (speed achieved, area required, power consumed) of the circuit implementation in the FPGA architecture under test [1].

Implementing a circuit in a modern FPGA requires that hundreds of thousands or even millions of programmable switches and configuration bits be set to proper state, on or off. Clearly if a circuit designer has to specify the state of each programmable switch in an FPGA very few designs will ever be completed! Instead, users of FPGAs describe a circuit at a higher level of abstraction, typically using a hardware description language (VHDL) or schematic entry. Computer-Aided Design (CAD) programs then convert these high level description in a programming file specifying the state of every programmable switch in an FPGA. To keep the complexity of these procedure tractable, the problem of determining how to map a circuit into an FPGA is normally broken into a series of sequential sub-problems, as shown in Figure 1.9 [3].

Logic Synthesis

The first stage of synthesis converts the circuit description, which is usually in a hardware description language or schematic form, into a netlist of basic gates. Then the logic synthesis process converts this netlist of basic gates into a netlist of FPGA logic blocks such that the number of logic blocks needed is minimized and/or circuit speed is maximized. Technology-independent logic optimization removes redundant logic and simplifies logic wherever possible.

Technology mapping

In the next stage of synthesis several LUTs and registers are packed into one logic block, respecting limitations such as the number of LUTs a logic block may contains, and the number of distinct input signals and clocks a logic block may contain. These limitations are determined by the technology

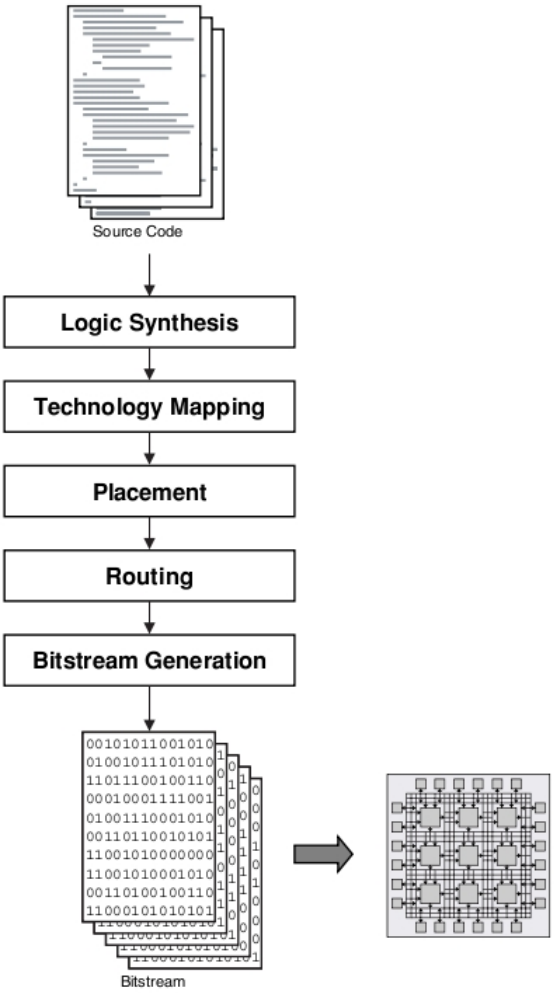

Figure 1.9: A typical FPGA mapping flow.

and architecture of the FPGA platform that will implement the circuit. The optimization goals in this phase are to pack connected LUTs together to minimize the number of signals to be routed between logic blocks, and attempt to fill each logic block to its capacity to minimize the number of logic blocks used.

Placement

Placement algorithms determine which logic block within the FPGA should implement each of the logic blocks required by the circuit. The optimization goals are to place connected logic blocks close together to minimize the required wiring (wirelength-driven placement), and sometimes to place blocks to balance the wiring density across the FPGA (routability-driven placement) or to maximize circuit speed (timing-driven placement).

Routing

Once locations for all the logic blocks in a circuit have been chosen, a router determines which programmable switches should be turned on to connect all the logic block input and output pins required by the circuit. In FPGA routing, one usually represents the routing architecture of the FPGA as a directed graph. Each wire and logic and each logic block pin becomes a node in this routing-resource graph and potential connections become edges. Routing a connection corresponds to finding a path in this routing-resource graph between the nodes representing the logic block pins to be connected. To avoid using up too many of the limited number of wires in an FPGA, one wants this path to be as short as possible. As well, it is important that the routing for one net not use up routing resources another net needs, so most FPGA routers have some king of congestion avoidance scheme to resolve contention for routing resources. An additional optimization goal is to make nets on or near the critical path fast by routing them using short paths and fast routing connections. Routers that attempt to optimize timing in this way are called timing-driven, whereas delay oblivious routers are purely routability-driven. Since most of the delay in FPGAs is due to the programmable routing, timing-driven routing is crucial to obtain good circuit speeds.

1.5 Organization of the Chapters

Subject of this thesis is to propose a complete framework for power, delay and area analysis of heterogeneous FPGAs. The rest of this thesis is organized in four parts:

- Chapter two is about State of Art in FPGA CAD tools. Complete frameworks and standalone tools of FPGA design will be analysed in order to study their capabilities, and their advantages and disadvantages.
- Chapter three presents the proposed framework NAROUTO. The practical problems occured in developement and every step in the design flow from VHDL circuit description to FPGA programming is thoroughly described. New tools, a Heterogeneity Support Toolset and modified existing tools are presented.
- The framework is tested in chapter four , in which the results are presented from eight benchmarks, concerning area, power ans delay of those benchmarks mapped in 6 different FPGA architectures.
- In Chapter five there are the conclutions of this thesis.
- Finally in chapter six there are thoughts for future reseacrh, one can do to expand the features and versatilty of NAROUTO framework.

Chapter 2

State of the Art

In the current chapter complete frameworks and standalone tools of FPGA design both academic and commercial will be analyzed. These tools are used mainly in research.

2.1 Standalone tools

2.1.1 Logic Synthesis and Technology mapping tools

2.1.1.1 MVSIS

The MVSIS group at Berkeley [4] studies logic synthesis and verification for VLSI design. MVSIS 2.0 is the newer version of SIS, which is an interactive tool for synthesis and optimization of sequential circuits. Given a state transition table, a signal transition graph, or a logic-level description of a sequential circuit, it produces an optimized net-list in the target technology while preserving the sequential input-output behavior. Many different programs and algorithms have been integrated into SIS, allowing the user to choose among a variety of techniques at each stage of the process. SIS serves as a tool for automatic synthesis and optimization of sequential circuits.

The main focus in MVSIS 2.0 is on new optimization algorithms that improve the quality of circuits generated by automatic synthesis tools and, at the same time, are scalable for practical use. Although the initial focus of MVSIS was on logic minimization for multivalued networks, over time it has developed into a full featured tool for synthesis and verification in general.

In MVSIS, techniques for combinational optimization of MV networks

have developed and included. MVSIS is an interactive tool, and has been made to have the look and feel of SIS. When applied to purely binary networks, it behaves almost exactly like the technology independent part of SIS but is faster. At this point in time, on binary files, MVSIS 2.0 is about 3-5 times faster than SIS, uses much less memory, and can be applied more robustly to much larger designs.

Currently in MVSIS 2.0 there is focus on three main areas of research:

Multi-Valued Logic Synthesis. Multi-level multi-valued (MV) logic synthesis can have many applications including:

1. Logic synthesis for multi-valued hardware devices such as current-mode circuits.
2. Initial manipulation of a hardware description before it is encoded into binary and processed by standard binary logic synthesis programs; MV is a natural way to describe procedures at a higher level.
3. A front end to a software compiler for embedded control applications. Software lends itself naturally to the evaluation of multi-valued variables in a single cycle. For embedded applications specified from synchronous languages with finite state machine semantics, strong logic synthesis transformations can be applied for control flow optimization.
4. Asynchronous synthesis of delay insensitive (DI) circuits.
5. Data mining, where the objective is a simple description that summarizes the content of some data.

The MVSIS logic network is an extension of the traditional boolean network to include multi-valued nodes, each with its own range. MVSIS includes all the technology-independent transformations of SIS for combinational logic synthesis generalized from binary to multi-valued case, e.g. algebraic decomposition and observability don't care based node minimization. The domain of these optimizations is expanded by opening up multi-valued possibilities. It also includes transformations specific to multi-valued nodes such as combining binary nodes and producing multi-valued ones and vice versa, node minimization by

exploring non-determinism in the network, and bidecomposition into multi-valued MIN/MAX gates, etc.

Boolean Technology Mapping. Technology mapping is the process of expressing a given boolean network in terms of library gates (for standard cells) or look-up tables (for FPGAs). MVSIS has technology mappers for both standard cells and FPGAs. The core mapping algorithm in both cases uses boolean matching which is superior to structural matching for large and complex libraries, leading to better quality results.

Since the final mapping is derived from the technology independent netlist by local re-writes, the quality of the initial netlist determines to a large extent the quality of the final mapping. To mitigate this structural bias the new mapper introduces supergates and choice nodes.

A **supergate** is a virtual gate built from a set of real library gates. By having these larger gates, the mapper can look deeper into the circuit, thus producing better results. This is because the mapper is constrained by the fact that there is a correspondence in the mapped netlist and the initial netlist at the gate boundaries. By mapping with larger gates, these constraints are relaxed, leading to better quality.

Choice nodes in the initial netlist specify different structural implementations and serve as an efficient way of encoding multiple netlists in one. The choices can be added in a number of ways such as by

1. algebraic re-writing like the Lehman Watanabe mapper
2. combining netlists from different scripts
3. combining netlists at different stages of the same script (this is useful since synthesis scripts are often heuristic, and there is no guarantee that the final netlist is overall the best netlist for mapping).

Combinational Verification. The combinational verification capabilities of MVSIS include a BDD-based and a SAT-based equivalence checking command.

The first command represents a traditional approach, which constructs and compares the BDDs for the primary outputs of the two circuits

to be verified. This command is applicable to non-deterministic multi-valued networks. In the case of non-deterministic networks, the containment of SS-behaviours of the networks is checked. The BDD-based verifier can also return an error trace, which pin-points the mismatch in the functionality of the two circuits. The error trace is a minterm in terms of (multi-valued) primary input variables, for which one pair of the primary outputs of the two circuits produces different values. This verification option is only applicable to the circuits, for which BDDs can be efficiently constructed.

The second verification option is based on the representation of Boolean functions in terms of And-Inv Graphs (AIGs). A semi-canonical form AIGs, called Functionally Reduced AIGs, is used to prove equivalence of primary outputs in a way similar to the use of BDDs. The FRAIGs for the primary outputs of the two circuits are constructed and compared. The functional reduction of FRAIGs guarantees that the primary outputs are equal if and only if they are represented by the same FRAIG node. This command is currently applicable only to the binary circuits, but it is substantially more robust than the BDD-based one. It has been successfully used to verify industrial circuits with hundreds of thousands of gates. The FRAIG-based verification command can also be applied to the problem of bounded sequential verification, when a sequential circuit is unrolled for k time-frames and verified combinationally for all pairs of primary outputs of each time frame. This option may help quickly find bugs with short error-traces for large circuits after sequential optimization.

2.1.1.2 ABC

ABC [6] is a growing software system for synthesis and verification of binary sequential logic circuits appearing in synchronous hardware designs. ABC combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification.

ABC provides an experimental implementation of these algorithms and a programming environment for building similar applications.

ABC is currently in version 1.01 , the features of this version are :

- Basic data structures to represent and manipulate combinational and

sequential technology-independent networks in a variety of ways: as a netlist, as a logic network with nodes represented by SOPs (a SIS-style network) or by BDDs (a BDS-style network), as a technology-mapped network, as an AIG, as a sequential AIG, etc.

- Input file parsers for binary BLIF, binary PLA, BENCH format, a subset of EDIF (for reading ISCAS benchmarks), a subset of Synopsys equation format, and a subset of structural Verilog (for reading IWLS 2005 benchmarks).

- Output file writers for binary BLIF (both technology-independent and mapped), binary PLA (collapsed networks only), BENCH format, Synopsys equation format, CNF (combinational miters only), and two representations for circuit graphs: DOT format (used in the graph visualization package GraphViz) and GML format (used by some graph editors, such as yEd, a free product of yWorks).

- A specialized format BAF (Binary Aig Format) for reading/writing large AIGs into binary files. With BAF, reading and writing of AIGs with millions of AND-nodes can be done in a few seconds. The memory requirements are also reduced by almost an order of magnitude, compared to BLIF and Verilog.

- The data structures to represent logic networks that are conceptually similar to those used in SIS and procedures to perform multi-purpose operations on logic networks, such as technology-independent sweeping, logic sharing, disjoint-support decomposition, structural hashing and balancing of AIGs, creating combinational and sequential miters (product machines), unrolling sequential circuits for a number of time frames, etc.

- Efficient combinational synthesis flow based on AIG balancing, rewriting, and refactoring using DAG-aware transformations inspired by Per Bjesse and Arne Boralv, "DAG-aware circuit compression for formal verification", (ICCAD 2004, pp. 42-49). These commands work on combinational networks only. They can be used to accumulate structural choices that lead to additional freedom in technology mapping and sequential synthesis.

- Procedures to detect and accumulate structurally different representations of Boolean functions (FRAIG package).
- Procedures working with several snapshots of the same network. These procedures implement the idea of lossless synthesis, which consists in accumulating all or some of the functionally-equivalent structurally-different representation of logic functions that appear in the course of logic transformations. Mapping over the multiple structures helps overcome structural bias and tends to improve delay and area.
- Technology mappers for variable-LUT-size FPGAs and standard cells, applicable to traditional logic networks and logic network with structural choices. Recently added priority-cut-based mapper has improved memory and runtime.
- Several area-oriented resynthesis commands for LUT mapping, such as imfs and lutpack.
- Combinational equivalence checking based on SAT sweeping, a resource-aware combination of simulation and SAT.
- Basic data structures for sequential synthesis (sequential AIG) and an experimental implementation of integrated sequential optimization, which combines logic synthesis, technology mapping, and retiming for standard cells and FPGAs. Initial state of circuits after retiming is computed by formulating and solving a SAT problem.
- Programmable interface to MiniSat, an extensible SAT solver by Niklas Eén and Niklas Sörensson. The C version of the solver is included in the current release of ABC.
- Procedures to construct the global BDDs of the primary output functions of the network using CUDD package by Fabio Somenzi. The code of CUDD Version 2.3.1 is included in the current release of ABC.
- Simple bounded sequential equivalence checking, which unrolls sequential circuits for a given number of timeframes and performs combinational equivalence of the resulting combinational circuits.
- Sequential synthesis commands lcorr and ssw for detecting and merging sequentially-equivalent registers and internal nodes. These command

can be seen as extension into the sequential domain of the notion of SAT sweeping (command fraig).

- Unbounded sequential equivalence checking command dsec applicable to two networks to be verified for equivalence (and its counterpart, command dprove that works on a sequential miter).

2.1.2 Clustering, Place and Route tools

2.1.2.1 VPR 4.30

T-Vpack and VPR tools are developed in university of Toronto. VPR is a placement and routing tool for array-based FPGAs, and T-VPack is a logic block packing (clustering) program. VPR was written to allow circuits to be placed and routed on a wide variety of FPGAs to facilitate comparisons of different architectures. The version since 1999 (until 2009) was 4.30[7] [1] but it is still popular in academic research.

T-Vpack T-VPack takes as input a technology-mapped netlist of lookup tables (LUTs) and flip flops in .blif format, and outputs a .net format netlist composed of more complex logic blocks. The logic block to be targeted is selected via command-line options. The simplest logic block T-VPack can target consists of a LUT and a FF. By default, T-VPack marks all clock nets in the input netlist as global nets which VPR should not route. Since clocks are typically routed via a dedicated network in FPGAs, this is usually the most realistic thing to do.

T-VPack is capable of targeting a more complex form of logic block, which its called a cluster-based logic block. Figure 2.1 depicts an example. A cluster-based logic block consists of N basic logic elements (i.e. N LUTs and N FFs), along with local interconnect that allows the N cluster outputs to be routed back to LUT inputs. Since the number of logic block inputs, I, can be less than the total number of LUT inputs (K×N, where K is the number of inputs per LUT), the local interconnect also allows each of the I inputs to be routed to any of the K×N LUT inputs. Cluster-based logic blocks are very similar to the logic blocks used in the Altera 8K and 10K FPGAs, and are reasonably similar to those used in the Xilinx 5200 and Virtex FPGAs.

VPR

VPR takes two input files, a netlist describing the circuit to be placed and routed, and a description of the FPGA architecture. Optionally, one can also input a placement file to VPR if one desires that an existing placement be routed only.

Some of the architectural parameters that can be specified in the architecture description file are:

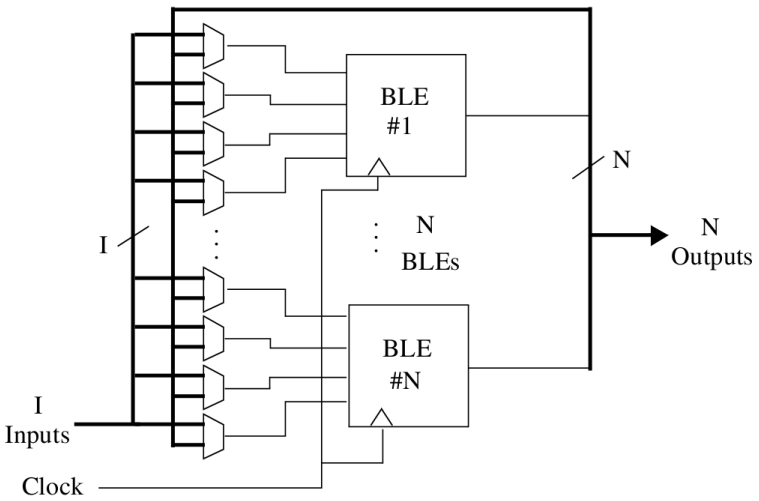

Figure 3: A cluster-based logic block.

Figure 2.1: A cluster-based logic block.

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

In addition, if global routing is to be performed, one can also specify:

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

Finally, if combined global and detailed routing is to be performed, one also specifies:

- the switch block [1] architecture (i.e. how the routing tracks are interconnected),
- the number of tracks to which each logic block input pin connects (Fc [1]),
- the Fc value for logic block outputs, and
- the Fc value for I/O pads.

In terms of routing area, VPR outperforms all other academic placement and routing packages so its a very popular tool for academic research.

VPR can be run in one of two basic modes. In its default mode, VPR places a circuit on an FPGA and then repeatedly attempts to route it in order to find the minimum number of tracks required by the specified FPGA architecture to route this circuit. If a routing is unsuccessful, VPR increases the number of tracks in each routing channel and tries again; if a routing is successful, VPR decreases the number of tracks before trying to route it again. Once the minimum number of tracks required to route the circuit is found, VPR exits. The other mode of VPR is invoked when a user specifies a specific channel width for routing. In this case, VPR places a circuit and attempts to route it only once, with the specified channel width. If the circuit will not route at the specified channel width, VPR simply report that it is unroutable. VPR can perform either global routing or combined global and detailed routing.

VPR also has graphics for a more "human readable" placing and routing, as shown in figures 2.2, 2.3, 2.4, 2.5.

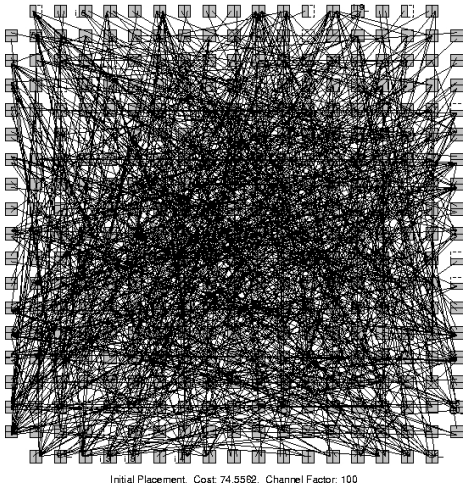

Figure 2.2: Initial Random Placement

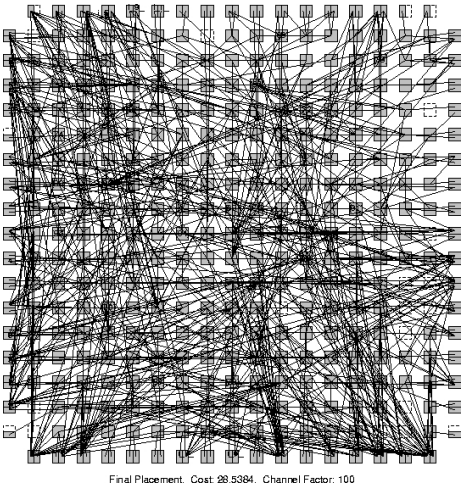

Figure 2.3: Final Placement

2.1.2.2 Heterogeneous framework VPR 5.0.2

The current version of the tools is (since 2009) 5.0.2 [8]. There are four important added features in this version[9]:

- **Single Driver Routing** - Single-Driver routing architectures have routing segments that are driven by only one driver that is fed by a multiplexer. This style has been shown to dominate multi-driver routing architectures, and is in wide use in the major commercial FPGAs. The new version of VPR supports this routing in addition to the multi-driver (bi-directional) routing supported by the original version of VPR.

- **Heterogeneity** - VPR is now designed to handle a heterogeneous mixture of block types. Most current reasonably sized user designs make use of heterogeneous blocks such as multipliers and memories.

- **Architecture Files** - Architecture files define the FPGA architecture to the VPR flow. With this release we are providing a broad swath of different architectures, which are optimized to meet different area and delay criteria, in a wide range of IC process generations. Architecture files in version 5.0.2 are in .xml format.

- **Regression Tests** - Included with this release of VPR are regression

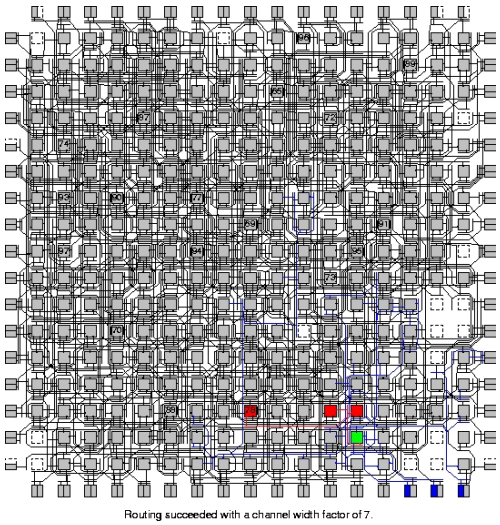

Figure 2.4: Completely (Detailed) Routed Circuit

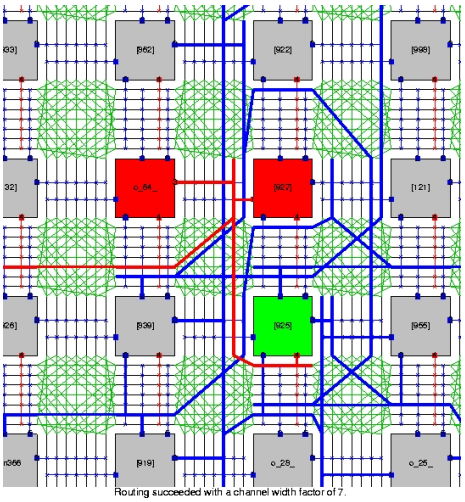

Figure 2.5: Close-up View of the FPGA Routing Architecture

tests that are used to verify the correct operation of VPR and the full CAD flow; so if one changes the source code, he can check if he have broken anything he didn't intend to.

Heterogeneity

One very important added feature to the new VPR is support of heterogeneity. A heterogeneous FPGA contains hard specific-purpose circuits in addition to basic logic and routing fabric (figure 2.6). A hard circuit is a specific circuit included on an FPGA to perform specific logic functions, which could also be implemented using the base logic units and the routing fabric. Given these definitions, there are two common ways hard circuits are added to FPGAs.

The first is to add specific circuits into all the tiles in an FPGA. In this case, all the tiles are the same, and the FPGA remains a homogeneous array of tiles. We call this soft fabric heterogeneity. The hard circuits within each tile allow certain functions to be implemented with better area-efficiency and faster speed. For example, flip flops are commonly paired with LUTs to save both speed and area in, arguably, all cases compared to implementing the flip flop in the base soft logic fabric. We call a unit of soft fabric heterogeneity that consists of the base logic unit and additional hard circuits a Soft Fabric Logic Unit (SFLU). The SFLU is called a Logic Element (LE) by Altera and

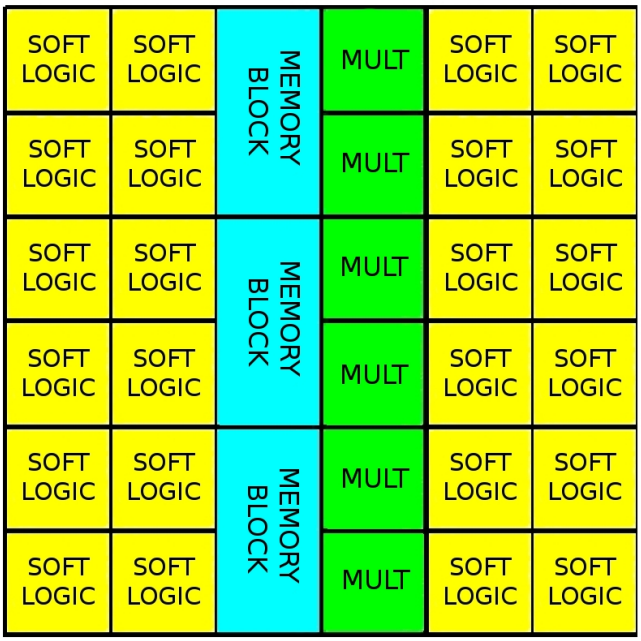

Figure 2.6: Heterogenous blocks in an FPGA.

a slice by Xilinx[10].

The second way to employ hard circuits in an FPGA is to include a specific circuit as a differentiated tile, which separately abuts the SFLU tiles. We call this tile-based heterogeneity. Examples of heterogeneous tiles are multipliers which are large circuits added to an FPGA as unique tiles.

Heterogeneity support in VPR

Supporting heterogeneous FPGA requires an entire CAD flow above the packing placement and routing supplied by VPACK and VPR. Accompanying this release of VPR is a full set of CAD tools offering a full Verilog-to-routing flow. This flow starts with ODIN, that provides elaboration of Verilog. Logic synthesis is performed using a specific version of the ABC tool from UC Berkeley. Packing is performed with an updated version of TVPack that can pass heterogeneous blocks through untouched. Finally, placement and routing is performed with the latest version of VPR. Though there are limitations in this flow. In particular, the front end synthesis tool, ODIN, only identifies multipliers as hard blocks. Other blocks may be added by modifying the program to find other hard structures. Any hard blocks that are found are treated as block boxes for all the tools except VPR. This means inputs to the blocks are treated as Primary Outputs and outputs are treated as Primary Inputs and, as a result, timing-driven optimization of paths containing blocks is not possible. The architectural Specification of a Hard Block in VPR's architecture file is :

- Column based
- Each block has parameterized (multi-row) height
- Transparent routing
- All input-output timing paths of block can be specified
- Allow combinational or registered outputs
- All other parameters same as soft cluster

2.1.3 Powermodel Framework

Powermodel is aimed at island-style FPGA architectures, which have logic blocks, switch blocks, connection blocks, and routing, as shown in Figure 2.7, with an H-tree clock network, as shown in Figure 2.8. The model has two modules: an activity generation module, and a power estimation module. The first module employs the transition density model to determine the switching activities inside the circuit. The second module estimates the power consumption at the transistor level. The second module is based on T-Vpack and VPR 4.30 tools. The model was calibrated using HSPICE with the technology parameters from TSMC for a 1.8 volt, 0.18mm CMOS technology [13].

The model contains includes terms for dynamic power, short-circuit power, and leakage power. Although the techniques have been used before, the integration of these techniques into a flexible power model for FPGAs is a novel approach. The model is flexible enough to target FPGAs with different look-up table (LUT) sizes, different interconnect strategies (segment length, switch block type, connection flexibility), different cluster sizes (for a hierarchical FPGA), and different process technologies.

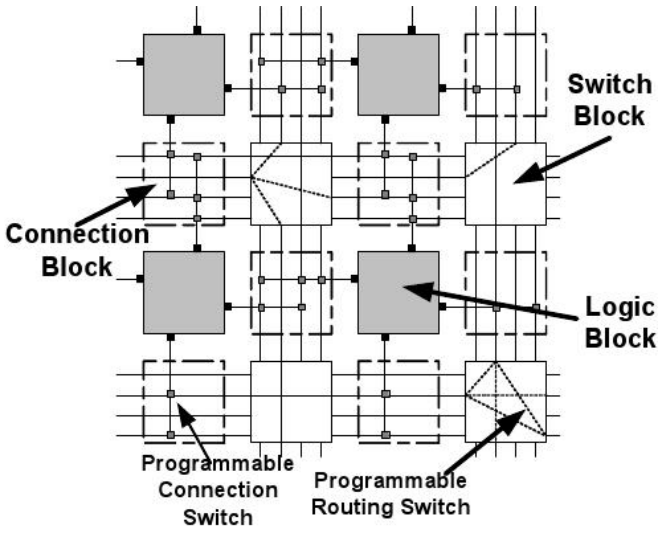

Figure 2.7: Island style FPGA.

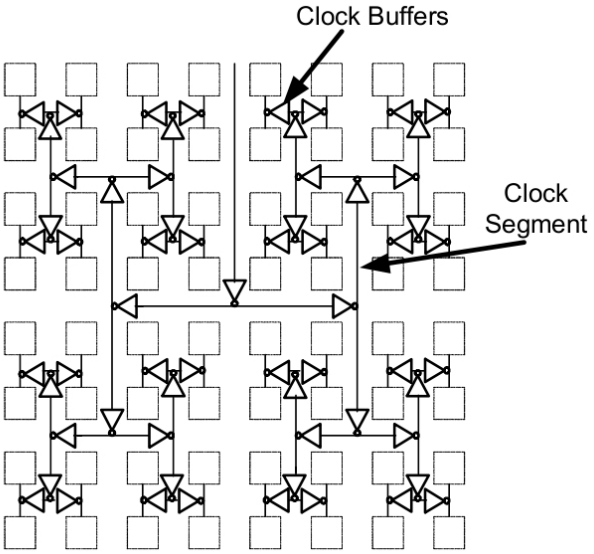

Figure 2.8: H-tree clock network.

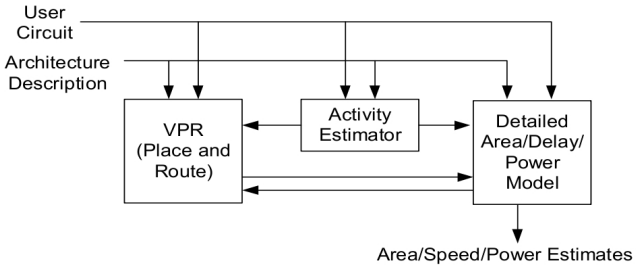

Figure 2.9: Powermodel framework.

ACE 1.1

ACE 1.1 implementing probabilistic techniques is responsible for the activity generation step. The probabilistic technique used is the Transition Density Model. The Transition Density model is based on two parameters for each signal: the transition density and the static probability. The transition density of a signal represents the average number of transitions of that signal per unit time, and the static probability is the probability of the signal being high at any given time. The transition density and static probability values of all the signals are calculated iteratively from the primary inputs to the primary outputs.

ACE 1.1 takes as input the netlist of the circuit in BLIF format and as output it produces an .act file which has the transition density and the static probability of each pin in the design.

ACE 2.0

Currently there is a new version of ACE (2.0) by J. Lamoureux[12]. This new version of Ace has addressed and solve some accuracy and speed problems of previous version.

Probability-based estimates are less accurate than simulation-based estimates for two reasons. First, they typically ignore wire and gate delays, which are the cause of glitching activity. Second, they typically ignore both spatial and temporal correlation between signals. Spatial correlation occurs when the logic value of a wire depends on the value of another wire. Spatial correlation can occur between primary inputs when the input data has correlation or between internal nodes when gates fan-out to multiple gates and later reconverge. Temporal correlation occurs when the value of a wire depends on previous values of that same wire. This can also occur at the primary inputs or within sequential circuits which have feedback. Ignoring temporal correlation introduces between 15% and 50% error, and ignoring spatial correlation introduces between 8% and 120% error.

Ace 2.0 algorithm have three phases :

1. The first phase determines the static and switching probability for logic and flip-flops within sequential feedback loops in a circuit. ACE in this phase simulate switching probabilities instead of switching activities. Each cycle of the simulation begins by updating the values of the primary inputs with the next input vector. If vectors are not supplied, ACE-2.0 pseudo-randomly generates vectors which match the specified

input activities. Once the input values are specified, the routine determines the output value of each gate in the feedback logic in topological order from the primary inputs to the primary outputs. Finally, at the end of each cycle, the routine updates the value at the output of each flip-flop based on the input value.

2. Although ACE-2.0 uses simulation to obtain static and switching probabilities for logic and flip-flops within sequential feedback loops, switching probabilities are also required for logic and flip-flops not within sequential feedback loops. These remaining probabilities are calculated using the Lag-one model, which produces exact switching probabilities (assuming that inputs are not correlated). For speeding up this phase combined Partial Collapsing and BDD Pruning are used

3. The final phase of the ACE-2.0 algorithm addresses the issue of accurately and efficiently modeling the glitch component of switching activities.

T-Vpack

In powermodel the modified T-Vpack has an added feature. If the .act file from the ACE is given as an input, the T-Vpack has two extra outputs required for the final power estimation:

1. An .act2 file which will be used by VPR later on in the flow. Each global net in the design has its corresponding probability and the transition density value listed. Nets that connect clusters (ie. use the general-purpose routing) are listed also in the same way. Each subblock (LUT and/or FF) has its corresponding activities of all inputs followed by the clock static probability and clock transition density. Then, the probability and transition density of the node between the LUT and flip-flop (if there is both a LUT and flip-flop used within this subblock) is listed. (this number is 0 if a flip-flop is not used). Finally, the activity of the output is listed.

2. A .fun file which will also be used by VPR in the power estimation flow. Each subblock in file .fun its described by its name and the corresponding logic function implemented in the subblock.

VPR

In powermodel the VPR has been modified to estimate power consumption. In the current implementation, the activity estimator and the power model are not used to guide the placement and routing. It estimates the power consumption only after placement and routing has occurred. In order to estimate dynamic power, the powermodel separate the resources in an FPGA into three categories: routing resources, logic blocks, and the clock network. The power dissipated by resources in each category is estimated separately.

The modified VPR takes as inputs the .ac2 and .fun outputs from T-Vpack. Also the architecture file needs additional parameters for power analysis. The outputs of the powermodel's VPR is the placement and routing file (same as simple VPR), and the power analysis files :

LBpower.echo that lists the power dissipation of each logic block

Routing_power.echo that lists the power consumed in routing in each net

CLKpower.echo that lists the power dissipation by clock network

power.echo that lists the total power dissipation

2.1.4 Nettovqm

Nettovqm is a tool provided in by VPR 4.30 research team. This tool can convert a VPR format (.net) netlist into the .vqm format netlist used by Altera's Quartus CAD suite to represent a technology-mapped netlist. This is allowing the designer to run a VPR-format netlist targeting 4-LUTs and size 1 logic clusters through Altera's commercial tools, and compare result quality. It doesn't work with other LUT sizes since they don't match the LUT size in Altera's Stratix and Cyclone parts, and it needs a cluster of size 1 since Quartus needs an unclustered, technology-mapped netlist.

2.1.5 Convert_arch_to_xml

Since the architectures files in new VPR are in .xml format and these files in old VPR are in .arch format, in VPR 5.0 toolset is the Convert_arch_to_xml

tool. This is a program that converts old vpr architecture files into the Heterogeneous VPR's XML format, in order to support backward compatibility.

2.1.6 VPR 5.0 with power estimation

In 2009 a version of VPR 5.0 with power estimation was developed by Dr. Peter A. Jamieson in Miami university [14]. The power estimation is based in Kara Poon's powermodel in VPR 4.30 (for further details see 2.1.3). This software is open source and its currently unsupported.

The updated power models support the new data structures in VPR 5.0, the new architecture file format, and can estimate power on both uni and bi directional routing structures. It was tested in non-heterogeneous fpga architectures since there wasn't a complete power estimation design framework for heterogeneous FPGAs. In the process of testing this VPR version there were some errors in the power estimation of homogeneous FPGA designs (very large, or even negative results).

2.2 Existing Frameworks

2.2.1 Meander

2.2.1.1 What is Meander

Meander[15] is a result from AMDREL (Architectures and Methodologies for Dynamic REconfigurable Logic) project[16]. Its an FPGA design framework that has the following features :

- It supports a complete –from VHDL to FPGA programming– design in which all tools are exclusively open source software
- Its used under Linux environment
- Its written in C/C++ language
- The input files can be RT VHDL, Structural VHDL, EDIF, or BLIF files
- The output is an FPGA programming file
- The design flow is independent from the underlying architecture
- It works in a variety of processors (Pentium / SPARC)
- It runs either locally or in a network
- Every tools in the flow can be used individually
- It has Graphical User interface (GUI)

The design flow and some comparisons between Meander, T-Vpack/Vpr and ALLIANCE is shown in figure 2.10 and to table 2.1.

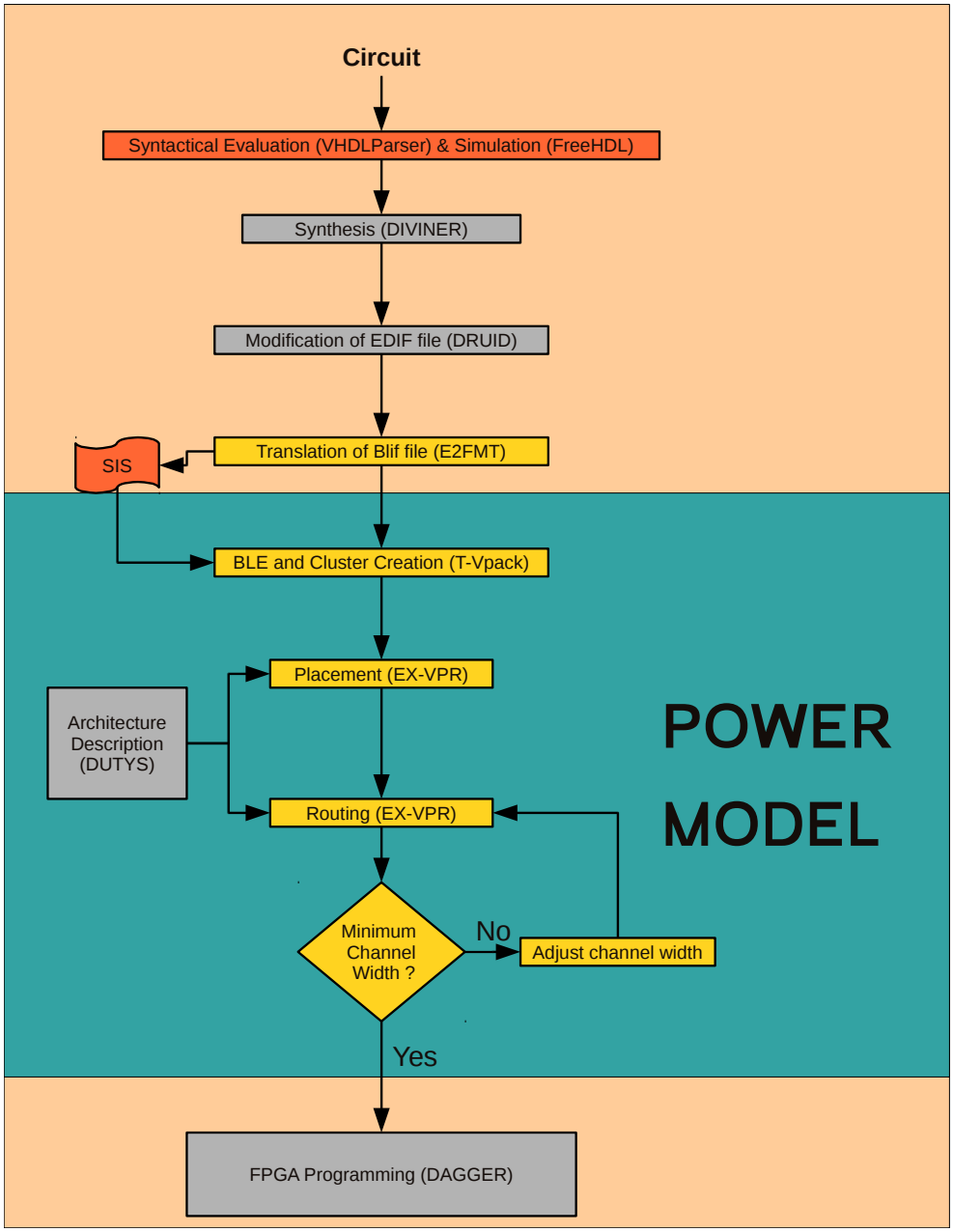

Figure 2.10: Meander design flow.

FEATURE	**Meander**	**TORONTO (T-Vpack, VPR)**	**ALLIANCE**
Data Input Format	VHDL /Verilog	BLIF	VHDL
Synthesizer	✓	✗	✓
Format Translation	✓	✗	✗
Architecture Description	✓	✗	✗
Architecture Exploration /Modification	✓	✗	✗
Place & Route	✓	✓	✓
Bitstream Generation	✓	✗	✗
Back annotation	✗	✗	✗
Power Estimation	✓	✗	✗
Area Estimation	✓	✗	✗
GUI	✓	✗	✗
User Manual	✓	✓	✓
OS	Linux	Solaris	Linux

Table 2.1: Comparison betwen Meander, Toronto university framework and ALLIANCE framework

2.2.1.2 Meander Design Flow

VHDL Parser

The flow of design tools included in Meander starts with the description of circuit in VHDL hardware description language which is very popular specially in the field of circuitry design for FPGA. Initially the syntax of the description of circuit is evaluated according to VHDL-93 standard. The evaluation is done with the VHDL-Parser which is written in Java programming language.

FreeHDL

After the syntax evaluation of the .vhdl file a simulation is needed to confirm that the circuit has the expected behaviour. This evaluation checks the outputs of the circuit when in the inputs are applied user-defined signal waveforms and delays. The tool that carries out this evaluation is FreeHDL which is compatible with VHDL-93 standard and has a graphical user interface in which the user can see the waveforms of the signals.

DIVINER

Democritus University of Thrace RTL Synthesizer (DIVINER) is a software tool that performs the basic function of the RTL synthesis procedure. It converts a VHDL description to an EDIF format netlist. DIVINER supports a subset of VHDL as all synthesis tools do. It does not support Verilog or other hardware description languages. It outputs a generic EDIF format netlist, which can then be used with technology mapping tools in order to implement the digital system described in the DIVINER input file in any ASIC or FPGA technology and not only the AMDREL fine-grain reconfigurable hardware platform. It only performs a partial syntax check of input VHDL files, therefore the input files should be compiled first using any VHDL simulation tool, commercial (Modelsim) or open-source (FreeHDL). Additionally, at this stage, DIVINER does not perform Boolean optimization, therefore the designer should be careful in the VHDL coding of Boolean expressions.

DRUID

DemocRitus University of Thrace EDIF to EDIF translator (DRUID) is a tool that converts the EDIF format netlist produced by Leonardo or DIVINER to an equivalent EDIF format netlist compatible with the next tool in the tool chain. Even though the output of the synthesis tools and the

input of the LUT-mapping tools are in the same format, there are differences in their generic libraries. Some cells are identical in functionality but with different names and port names, while other cells present in one tools library are absent in the others, and then they may have to be decomposed to gates. These conversions among other necessary functions compose the functionality of DRUID. DRUID is a tool that integrate the output of a synthesis tool (DIVINER or Leonardo Spectrum) with the rest of the tool flow. The input file that is processed by DRUID is an EDIF format netlist file and the output file that is produced is also an EDIF file, compatible with the rest of the tool flow. It is based on the generic output EDIF file produced by Leonardo Spectrum, because of the extensive use of this commercial synthesis tool. However, DRUID can appropriately modify any EDIF file of version 2 0 0 with only minor changes.

DRUID serves a threefold purpose:

1. It modifies the names of the libraries, cells (in EDIF format all structures are called cells) etc, found in the input EDIF file

2. It simplifies the structure of the EDIF file in order to make it compatible to Meander tool flow

3. It constructs, in the simplest way possible, the cells and generated modules that are included in the input EDIF file and are not found in the libraries of the following in Meander design flow tools.

This modification is necessary because the tools that follow DRUID in the proposed design flow can handle structures of limited complexity. Without DRUID, the hardware architectures that could be processed by the proposed flow would be the ones specified in structural level by using only the following basic components: inverter, AND, OR and XOR gates of 8 inputs maximum, a 2-input multiplexer, a latch, and a D flip-flop without set or reset signals. Moreover, signal vectors would not be supported. DRUID is necessary in order to implement real-life applications on the proposed fine-grain platform.

E2FMT

E2fmt takes as input a set of EDIF netlist files and converts them to another format. Supported output formats are EDIF, SLIF, and BLIF. There are options for flattening the hierarchy to a single level netlist and for loading a file, which contains definitions of the target technology gates. E2fmt

is provided to maintain compatibility with the following in Meander design flow tools which require netlists in BLIF format.

SIS

SIS is an interactive tool for synthesis and optimization of sequential circuits. For more details see section 2.1.1.1.

ACE ACE (activity estimator) has been incorporated in the framework to estimate the switching frequencies of all nodes in the circuit using the Transitional Density Model. Its included in the powermodel framework which will be analyzed in section 2.1.3.

T-Vpack

T-VPack takes as input a technology-mapped netlist of lookup tables (LUTs) and flip-flops in BLIF format (the output of SIS), and outputs a .net format netlist composed of logic blocks as presented in section 2.1.2.1.

EX-VPR

The Extended Versatile Placement and Routing tool (EX-VPR) is an FPGA placement and routing tool based on VPR (for more details see section 2.1.2.1).

In order to implement a more user-friendly interface than the command line, there is a GUI that helps the user input the files and options. The required fields are the input netlist file in .net format and the target FPGA architecture file. The produced outputs are the placement and routing files, Also the output placement and routing can be seen graphically. Additional available outputs are the power estimation and the critical path details.

EX-VPR has some important extra features compared to VPR 4.30. These features are shown in the table 2.2

DUTYS

DUTYS (Democritus University of Thrace Architecture file generator-synthesizer) creates the architecture file of the FPGA that is required by T-VPack and VPR. The architecture file contains a description of various parameters of the FPGA architecture, including size (array of CLBs), number of pins and their positions, number of BLEs per CLB, plus interconnection layout details such as relative channel widths, switch box type, etc. It has a GUI that helps the designer select the FPGA architecture features and then

Feature	**VPR**	**EX-VPR**
Placement	✓	✓
Routing	✓	✓
Supported Switch Boxes (SBs)	Subset, Wilton, Universal	Subset, Wilton, Universal, User specified Switch-Box
Multiple switch boxes	✗	✓
Multiple Segments	✓	✓
Thermat/Temperature Analysis	✗	✓
Insertion of IP core	✗	✓
Power Estimation	✓	✓(with powermodel, see 2.1.3)
Timing info (sec)	✓	✓
Silicon Area estimation (um^2)	✗	✓
Application specific FPGA design	✗	✓

Table 2.2: Comparison beetwen EX-VPR and VPR

automatically creates the architecture file in the required format.

DAGGER

DAGGER (DEMOCRITUS UNIVERSITY OF THRACE E-FPGA BITSTREAM GENERATOR) is the tool that programs the AMDREL fine-grain reconfigurable hardware. DAGGER currently only supports the type of FPGAs that are supported by the rest of the tools in the tool chain. The input files that required by the DAGGER tool in order to generate the bitstream for the FPGA program are:

- The netlist file that describes the circuit in ".NET" format which produced by the T-VPACK tool
- The output function file (.FUN) which is produced by the ACE and the T-VPACK tools
- The FPGA architecture description file (.ARCH) which is produced by the DUTYS tool
- The placement file (.P) of the circuit into the FPGA which is produced by the VPR tool
- The routing file (.R) of the circuit into the FPGA which is produced by the VPR tool

The DAGGER tool input, it is shown schematically in Figure 2.11. In this schematic, the rectangular boxes represent the four different tools that produce the required from DAGGER tool files. These files are shown in the ellipses bellow the tools, and the file format is written inside the eclipse. The ellipses that are filled only with only one color represent files produced only by one tool, while ellipses with two colors represent files produced with the cooperation of two different tools.

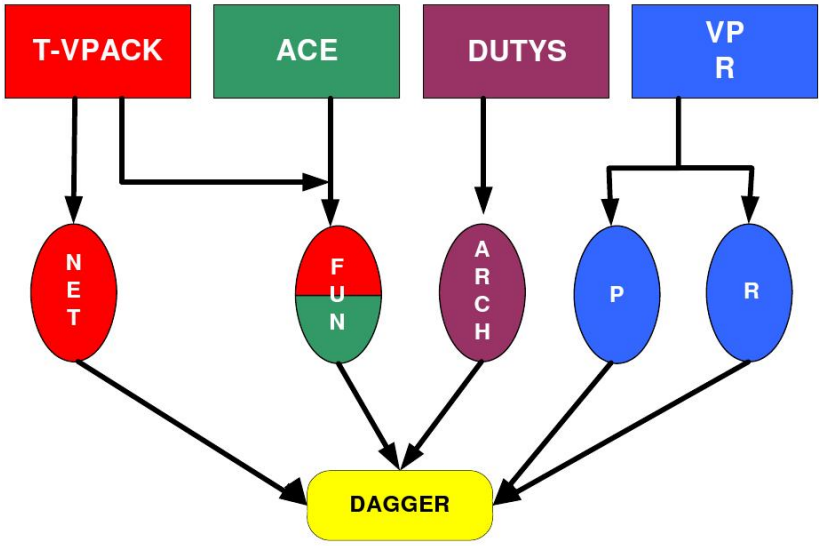

Figure 2.11: DAGGER required input files.

2.2.2 Quartus

2.2.2.1 What is Quartus

The Altera Quartus II design software provides a multiplatform design environment . It is a comprehensive environment for system-on-a-programmable-chip (SOPC) design and includes solutions for all phases of FPGA and CPLD design.

2.2.2.2 Quartus design flow

Design entry

The Quartus II software supports the following design entry methods[18]:

- Altera HDL (AHDL) Text Design File (.tdf)

- Block Diagram File (.bdf)

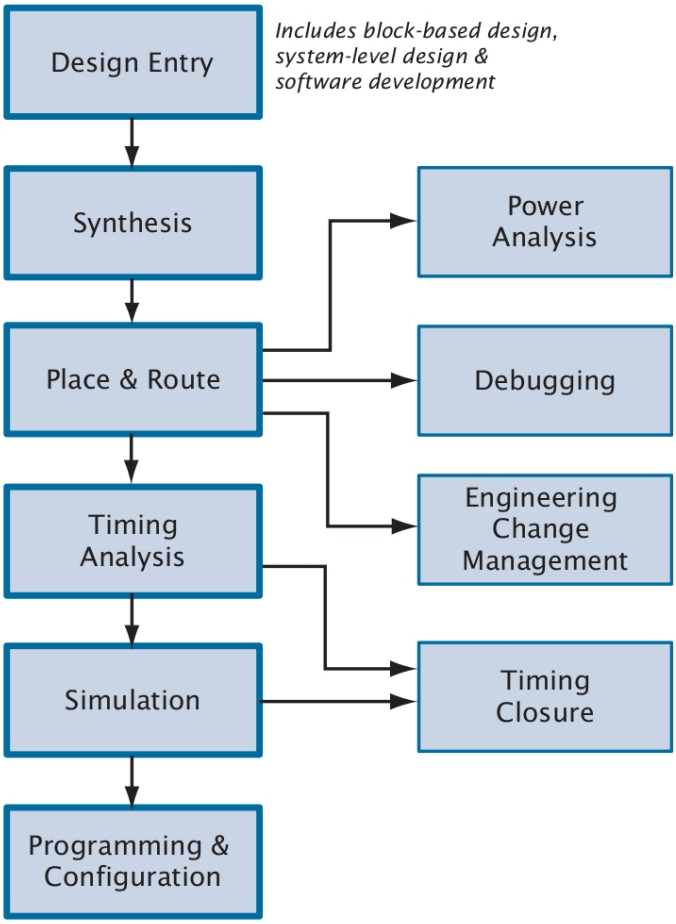

Figure 2.12: Quartus II design flow.

- EDIF Netlist File (.edf)
- VHDL (.vhd)
- Verilog HDL (.v) and System Verilog (.sv)

The Quartus II software has an advanced integrated synthesis engine that fully supports the Verilog HDL and VHDL languages and provides options to control the synthesis process.

The designer can use the Quartus II Block Editor, Text Editor, MegaWizard Plug-In Manager, and EDA (Electronic design automation) design entry tools to create design files that include Altera megafunctions, library of parameterized modules (LPM) functions, and intellectual property (IP) functions. All the availiable options are shown in figure 2.13.

Synthesis

The Quartus II advanced integrated synthesis software fully supports the industry standard hardware description languages and provides a complete, easy-to-use, stand-alone solution for today's designs. The designer select one of these optimization techniques: Speed, Area, or Balanced. For higher design performance, one can turn on synthesis netlist optimizations that are available when targeting certain devices. A designer can unmap a netlist created by an EDA tool and remap the components in the netlist back to Altera primitives.

Analysis & Synthesis module in Quartus II supports the Verilog-1995 (IEEE Std. 1364-1995) and Verilog-2001 (IEEE Std. 1364-2001) standards, a subset of features of the SystemVerilog-2005 (IEEE Std. 1800-2005) standard, and also supports the VHDL-1987 (IEEE Std. 1076-1987) and 1993 (IEEE Std. 1076-1993) standards. Analysis & Synthesis uses Verilog-2001 and VHDL-1993 by default[19]. If the designer is using another EDA synthesis tool, he can also specify a Library Mapping File (.lmf) that the Quartus II software should use to map non-Quartus II functions to Quartus II functions. The synthesis flow is shown in figure 2.14.

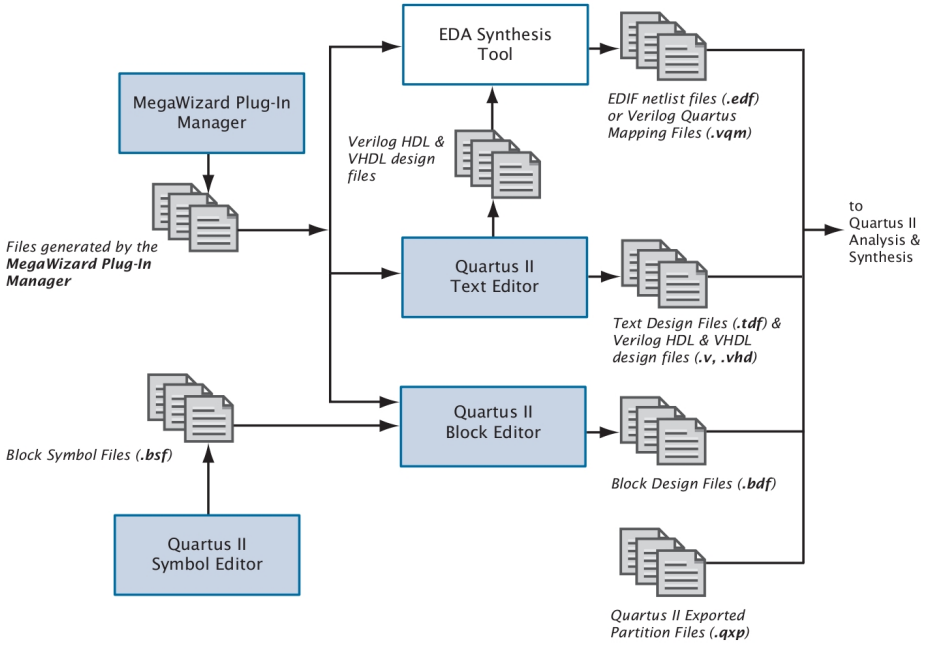

Figure 2.13: Quartus II design entry flow.

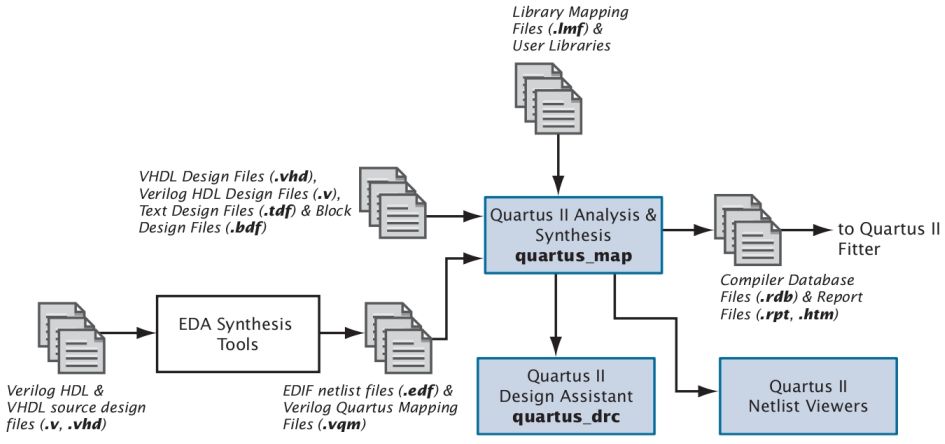

Figure 2.14: Quartus II synthesis flow.

Place and Route

The Quartus II Fitter places and routes a design, which is also referred to as "fitting" in the Quartus II software. Using the database that has been created by Analysis & Synthesis, the Fitter matches the logic and timing requirements of the project with the available resources of the target device. It assigns each logic function to the best logic cell location for routing and timing, and selects appropriate interconnection paths and pin assignments. Figure 2.15 shows the place and route design flow.

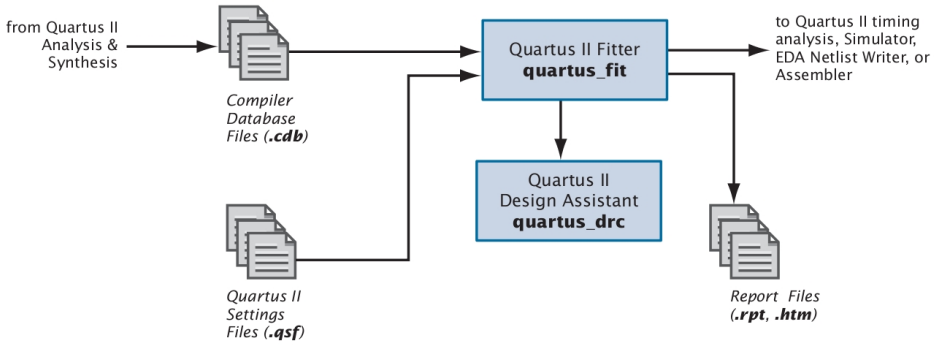

Figure 2.15: Quartus II place and route flow.

High-density device families supported in the Quartus II software, such as the Stratix series, sometimes require significant fitter effort to achieve an optimal fit. The Quartus II software offers several options to reduce the time required to fit a design. The designer can control the effort the Quartus II Fitter expends to achieve his timing requirements with options. If minimizing compilation time is more important than achieving specific timing results, one can turn off the optimization options.

Power-analysis

To develop an appropriate power budget and to design the power supplies, voltage regulators, heat sink, and cooling system, one needs an accurate estimate of the power that his design consumes. Power in Quartus II design flow can be estimated by using the PowerPlay Early Power Estimation spreadsheet available by the Altera, or with the PowerPlay Power Analyzer in the

Quartus II software. The designer can perform early power estimation with the PowerPlay Early Power Estimation spreadsheet by entering device resource and performance information. The Quartus II PowerPlay Analyzer tool performs vector-based power analysis by reading either a Signal Activity File (.saf) generated from a Quartus II simulation, or a Verilog Value Change Dump File (.vcd) generated from a third-party simulation. The power analysis flow is shown in Figure 2.16

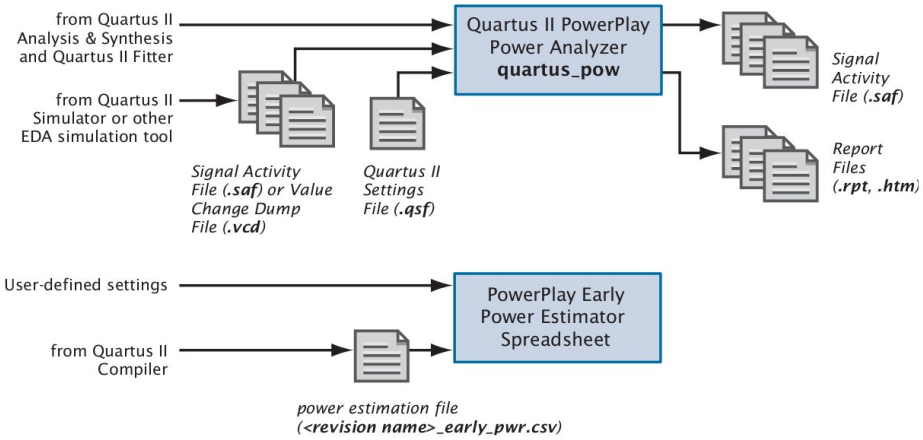

Figure 2.16: Quartus II Power Analysis flow.

The Quartus II PowerPlay Power Analysis Tools provide an interface that allows the estimation of static and dynamic power consumption throughout the design cycle. The PowerPlay Power Analyzer performs post-fitting power analysis and produces a power report that highlights, by block type and entity, the power consumed. The Altera PowerPlay Early Power Estimator estimates power consumption at other stages of the design process and produces a Microsoft Excel-based spreadsheet with estimate information.

Timing Analysis

The Quartus II TimeQuest Timing Analyzer allows an analysis of the performance of all logic in a design and help to guide the Fitter to meet timing requirements. The TimeQuest analyzer uses industry-standard Synopsys Design Constraint (SDC) methodology for constraining designs and reporting results. The designer can use the information generated by the

timing analyzer to analyze, debug, and validate the timing performance of the design.

Timing analysis measures the delay along the various timing paths and verifies the performance and operation of the design. The designer can specify constraints and assignments that help the design meet timing requirements. If constraints or assignments are specified, the Fitter optimizes the placement of logic in the device to meet those constraints. The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in the design with industry standard constraint, analysis, and reporting methodologies. The TimeQuest Timing Analyzer can be used by its graphical user interface (GUI) or command-line interface to constrain, run, and view results for all timing paths in the design. Before running the TimeQuest Timing Analyzer, one must specify initial timing constraints that describe the clock characteristics, timing exceptions, and external signal arrival and required times. The Quartus II Fitter optimizes the placement of logic in the device to meet your specified constraints.

Early in the design process, before final device fitting is completed, preliminary timing data is availiable by running an early timing estimate with the Start Early Timing Estimate command. When the design is complete, the designer can run a full timing analysis following compilation. During timing analysis, the TimeQuest Timing Analyzer analyzes the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as slack in the Report pane and in the console. If the TimeQuest Timing Analyzer reports any timing violations, the designer can determine whether the design requires additional timing constraints or exceptions, or if the design requires logic changes or place-and-route constraints.

Simulation

Functional and timing simulation of a design can be performed by using EDA simulation tools or the Quartus II Simulator. The Quartus II software provides the following features for performing simulation of designs in EDA simulation tools:

- NativeLink integration with EDA simulation tools
- Generation of output netlist files
- Functional and timing simulation libraries

- Generation of test bench template and Memory Initialization Files (.mif)
- Generation of Signal Activity Files (.saf) for power analysis

Figure 2.17 shows the simulation flow with EDA simulation tools and the Quartus II Simulator.

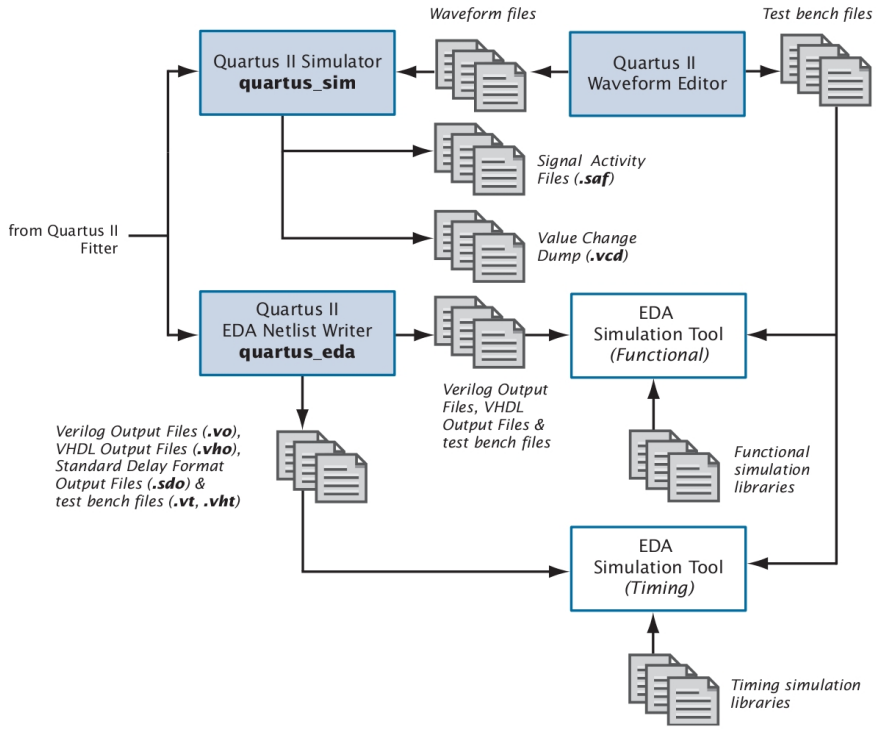

Figure 2.17: Quartus II Simulation flow.

Programming

Once a project has successfully compiled with the Quartus II software, the designer can program or configure an Altera device. The Assembler module of the Quartus II Compiler generates programming files that the Quartus II Programmer can use to program or configure a device with Altera

programming hardware.A stand-alone version of the Quartus II Programmer can be also used to program and configure devices. Figure 2.18 shows the programming design flow.

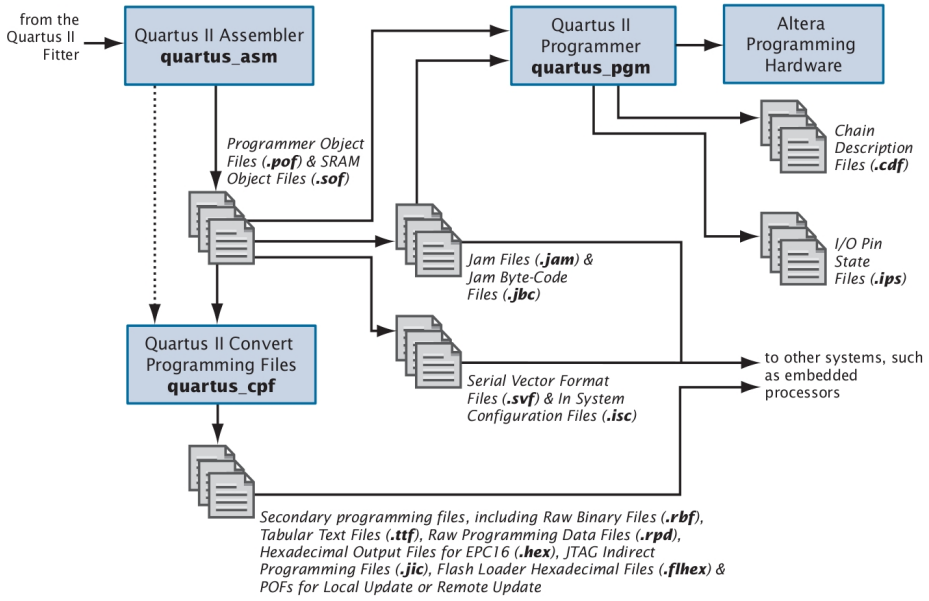

Figure 2.18: Quartus II Programming flow.

The Assembler automatically converts the Fitter's device, logic cell, and pin assignments into a programming image for the device, in the form of one or more Programmer Object Files (.pof) or SRAM Object Files (.sof) for the target device.

The Programmer uses the Programmer Object Files and SRAM Object Files generated by the Assembler to program or configure all Altera devices supported by the Quartus II software.

2.2.3 ALLIANCE

2.2.3.1 What is ALLIANCE

Alliance[20] is a complete set of free CAD tools and portable libraries for VLSI design. It includes a VHDL compiler and simulator, logic synthesis tools, and automatic place and route tools. A complete set of portable CMOS libraries is provided. Alliance is the result of the research at ASIM department of LIP6 laboratory of the Pierre et Marie Curie University (Paris VI, France). Alliance has been used for research projects such as the 875.000 transistors StaCS superscalar microprocessor and 400.000 transistors IEEE Gigabit HSL Router.

Each Alliance tool therefore supports several standard VLSI description formats : SPICE, EDIF, VHDL, CIF, GDS2. In that respect, the tools outputs are fully usable under the Compass and Cadence Opus environment, provided these tools have the necessary configuration files.

2.2.3.2 ALLIANCE Design Flow

One of the distinct characteristics of the Alliance system is that it provides a common internal data structure to represent the three basic views of a chip:

- the behavioral view,
- the structural view,
- the physical view.

Figure 2.19 details how all the Alliance tools are linked together around the basic behavioral, structural and physical data structures.

Below the 5 major steps of the basic design methodology in ALLIANCE are introduced.

Capture and Simulation of the behavioral view. The capture of the behavioral view is the very first step of the design flow. Within Alliance, any VLSI design begins with a timing independent description of the circuit with a subset of VHDL behavior primitives. This subset of VHDL, called vbe, is fairly restricted: it is the data-flow subset of this language. It is not very easy to modelize an architecture using this subset, but it has the great advantage of allowing simulation, logic synthesis and bit level formal proof on the same files.

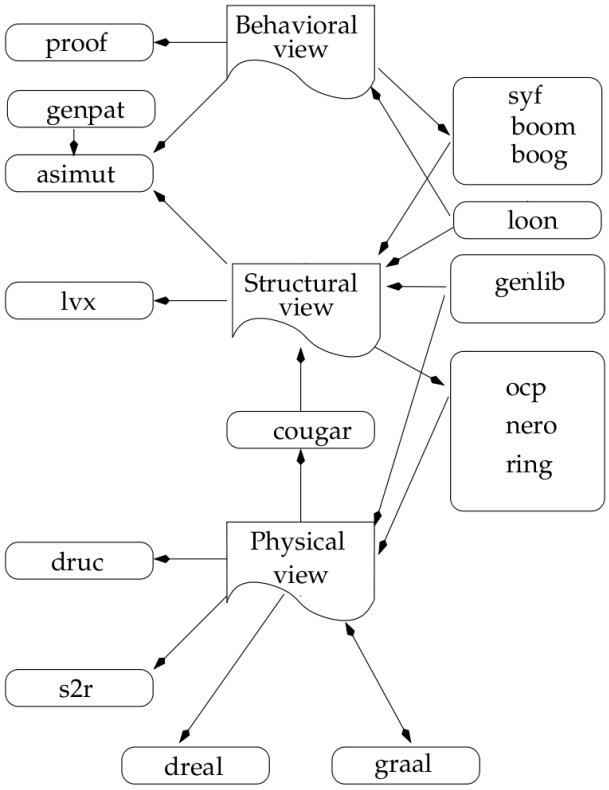

Figure 2.19: How the tools are linked on the data structures in ALLIANCE design Flow.

A VHDL analyzer called *vasy* can be used to automatically convert most common VHDL descriptions (using for example IEEE 1164 packages) to the restricted Alliance subsets (vbe and vst).
Patterns, VHDL simulation stimuli, are described in a specific formalism that can be captured using a dedicated language genpat. Once a VHDL behavioral description written and a set of test vectors have been determined, a functional simulation is ran. The behavioral VHDL simulator is called *asimut*. It validates the input behavior, according to the input/output vectors.

Capture and Validation of the structural view. The structural view can be captured once the data flow description is validated. The actual capture of the netlist relies either on specific description languages, genlib for standard cells or dpgen for data-path, or on direct synthesis from the data flow using the *boom* tool for optimization and the *boog* tool to map on a cell library. Genlib and dpgen are netlist-oriented libraries of C functions. The advantage of such an approach is that designers do not have to learn several language with specific syntax and semantics. Usually, the main behavior is partitionned in several sub-behaviors. Some are described recursively using the genlib language and the other ones can be directly synthesized from a VHDL description of the corresponding sub-behaviors. The *boog* tool takes an RTL description and generates a netlist of standard cell gates. An other subset of VHDL allows to capture finite state machines. This subset, called fsm, can be translated into a RTL description using the tool *syf*, and then the resulting description optimized using *boom* and finally synthesized as a netlist using once more *boog*.
Since *asimut* can operate on both RTL and structural views, the structural description is checked against the behavioral description by using the same set of patterns that has been used for behavioral validation.

Physical implementation of the design. Once the circuit netlist has been captured and validated, each leaf of the hierarchy has to be physically implemented. A netlist issued from *boog* is usually placed and routed

using the over cell router *nero*. If the netlist has been captured using genlib and if it has a high degree of regularity, it can be placed manually for optimization using other genlib functions.
The different parts can be placed and assembled together using ocp and routed using overcell router called *nero*, and this generates what ALLIANCE call a core. The circuit core is now ready to be connected to external pads. The core-to-pads router, ring, aims at doing this operation automatically, provided the user has given an appropriate netlist and some indications on pad placement.
The last stage of the physical implementation is the translation of the symbolic layout to a foundry compliant layout using the *s2r* tool. After that, the tape containing the circuit can be processed by the silicon supplier.

Layout verification. Powerful tools to perform behavior, netlist and layout verifications, have been introduced in the design flow .
The correctness of the design rules is checked using the design rule checker *druc*. An extracted netlist can be obtained from the resulting layout. *Cougar*, the layout extractor operates on both hierarchical and flattened layout and can output both flattened netlists (transistor netlist) and hierarchical netlists. The transistor netlist can be the input of a spice simulator. When extracted hierarchically, the resulting netlist can be compared with the original netlist by using the *lvx* tool. *lvx*, that stands for Logical Versus Extracted, is a netlist comparator that matches every design object found in both netlists.
The critical path of the circuit is evaluated using a commercial static timing analyzer, as Alliance doesn't provided one.

Test and Coverage evaluation. For now, the fault coverage provided by the functional patterns is evaluated using a commercial fault simulator, as Alliance doesn't provide one yet.

Chapter 3

Proposed Framework

Motivation

Power dissipation is becoming a major concern for semiconductor vendors and customers. Power is especially a concern in Field-Programmable Gate Arrays (FPGAs). The post-fabrication flexibility in these devices is provided using a large number of prefabricated routing tracks and programmable switches. These tracks can be long, and can consume a significant amount of energy every time they switch. In addition, the programmable switches add capacitance to each track; this further increases the power dissipation of FPGAs. Finally, the generic logic structures that are at the heart of every FPGA consume more power than the dedicated circuitry that would be found on an ASIC. For all these reasons, FPGA vendors have indicated that power is one of the primary concerns of their customers.

As concluded from Chapter two there isn't an academic complete framework for power dissipation for heterogeneous FPGAs. This thesis is proposing a complete framework for power estimation in heterogeneous FPGAs , named **NAROUTO**. With this framework one can explore different heterogeneous FPGA architectures in terms of delay , area and power.

NAROUTO

NAROUTO (**N**ational Technical University of **A**thens Hete**r**ogene**ou**s Fpga's power es**t**imation framew**o**rk), is composed by both commercial and open source CAD tools. It is based in Quartus, ACE, T-Vpack, VPR, and DAGGER, and all tools can also be used as standalone tools. The abstract design flow of NAROUTO is shown in figure 3.2. In figure 3.1 a more detailed flow for power estimation in NAROUTO framework is presented.

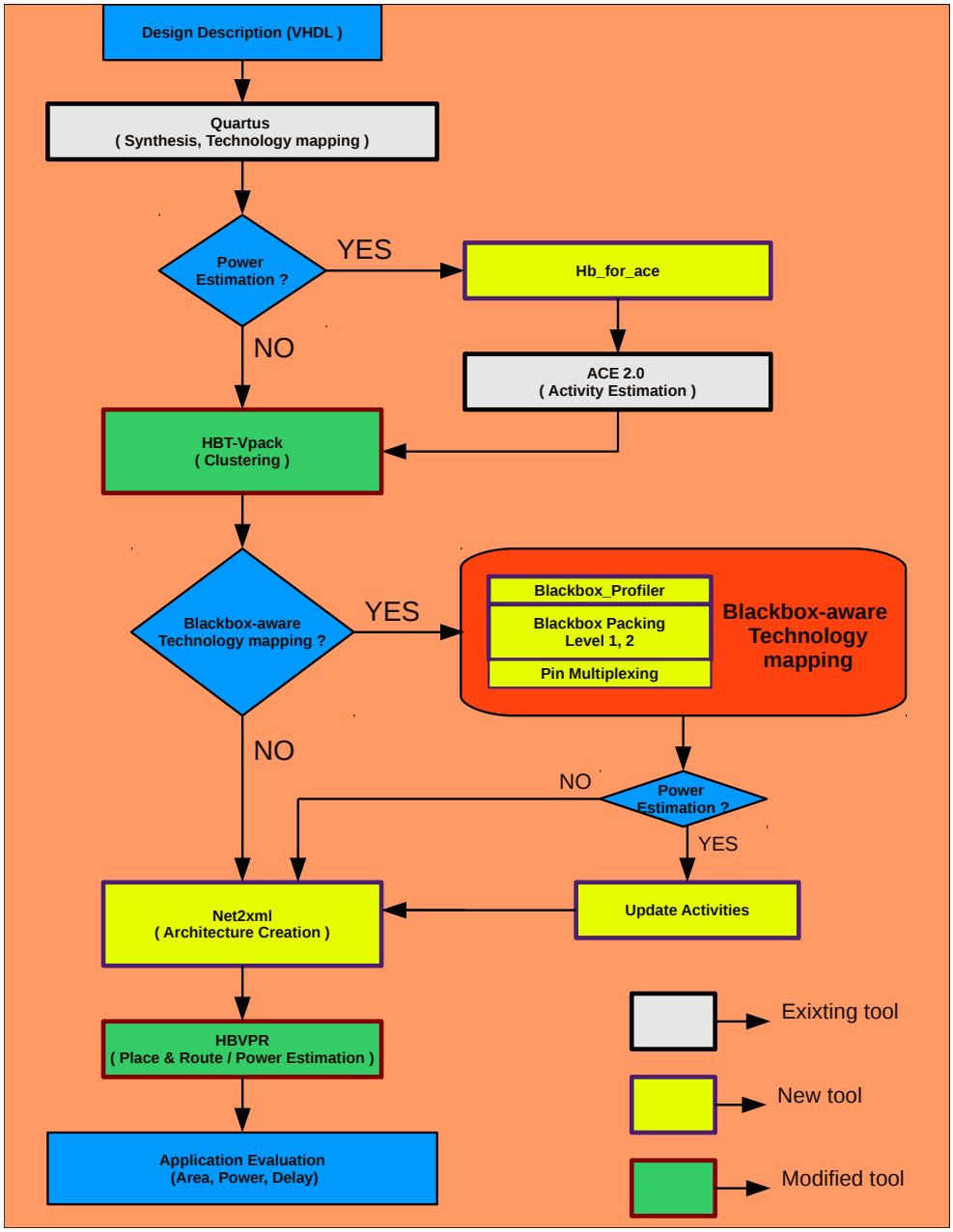

Figure 3.1: NAROUTO framework design Flow.

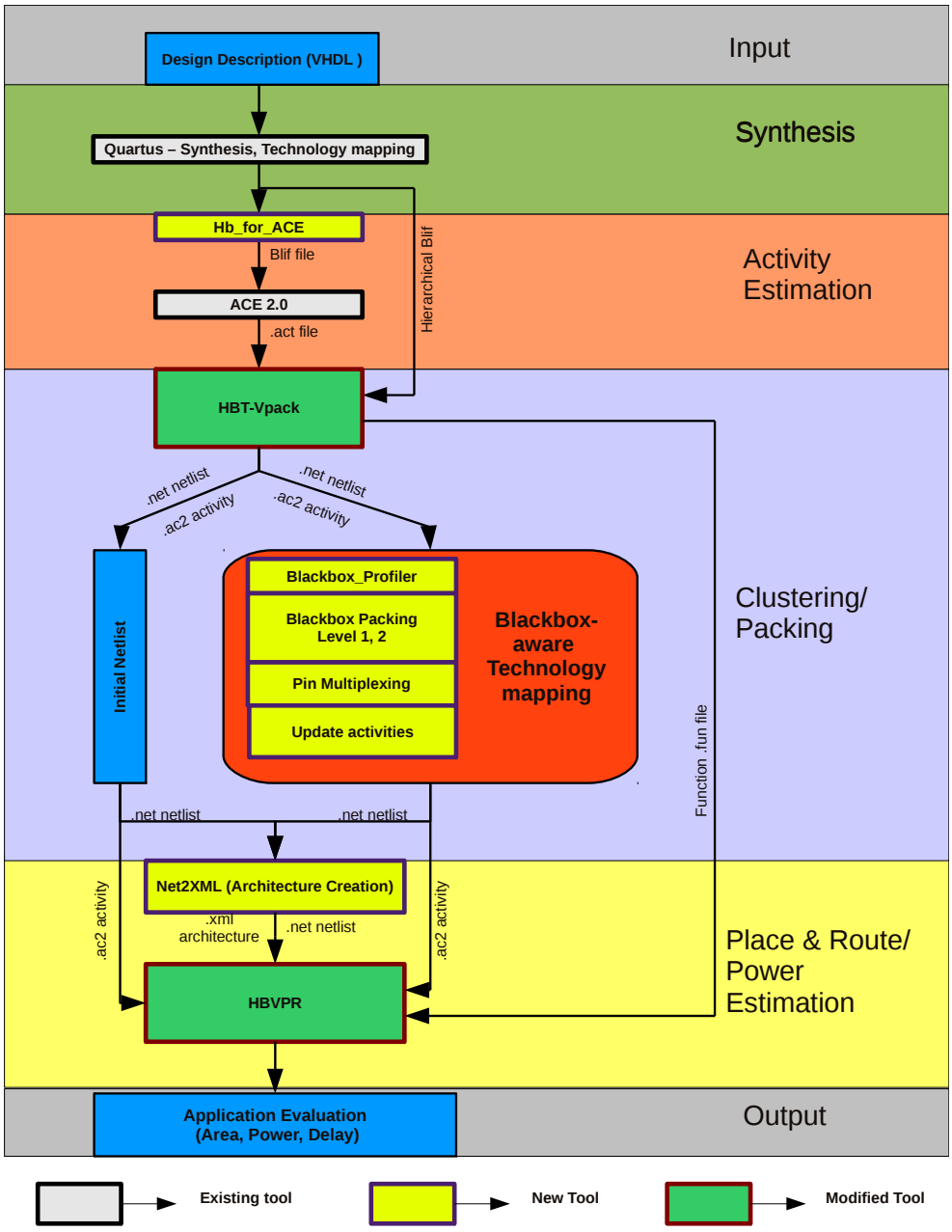

Figure 3.2: The detailed design flow for power estimation of NAROUTO framework.

3.1 Quartus, edif

In synthesis step as shown in figure .. the free Quartus II Web Edition Software v9.1 is used. The input files used in the design flow are in VHDL hardware description language, but all the Quartus II available options are possible. The output is an .edif netlist file as shown in figure 3.3 (the red highlighted area is the Quartus design entry flow that is used in NAROUTO).

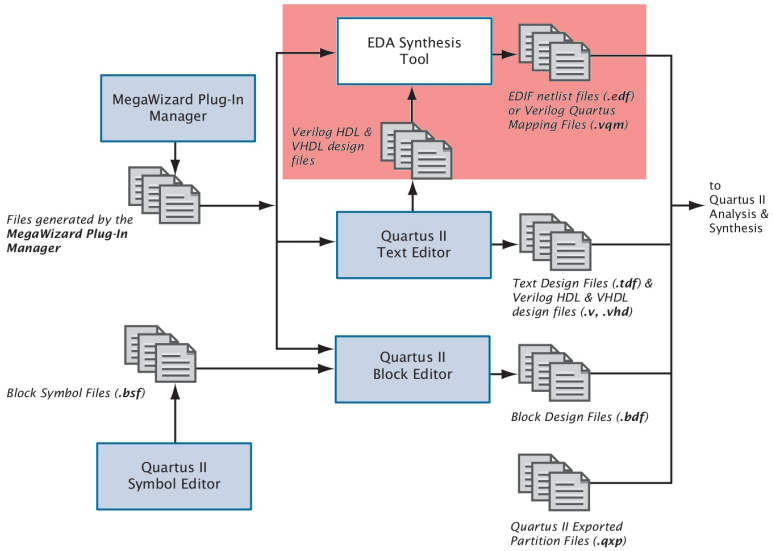

Figure 3.3: Design entry flow in NAROUTO taken from Quartus design flow (red highlighted area).

3.2 Quartus, hierarchical blif

ACE and T-Vpack require as inputs netlists in BLIF format. In the second step of NAROUTO design flow Quartus II it is used to make the appropriate netlist type transformation.

In Quartus II there are methods to output BLIF (Berkeley Logic Interchange Format). In the current Quartus II hierarchical BLIF is supported by handling all unknown netlist constructs such as DSP and memory blocks as black boxes. This increases not only the number of designs that can be written to BLIF, but also the quality of those designs since all modern designs of significant size contain either RAM or DSP blocks.

3.2.1 BLIF format

BLIF [21] is rather restrictive as a synthesis netlist format, as it is unable to express back-boxes that would be required to implement primitives such as DSP (multiplier) blocks, RAM that are not defined at the gate-level. The most glaring issue with BLIF is that there is no way to express arithmetic carry chains without converting them to gates. A quick view of some important structs in BLIF format is following.

A **model** is a flattened hierarchical circuit. A BLIF file can contain many models and references to models described in other BLIF files. A model is declared as follows:

```
.model <decl-model-name>
.inputs <decl-input-list>
.outputs <decl-output-list>
.clock <decl-clock-list>
     <command>
          .
          .
          .
     <command>
.end
```

A **logic-gate** associates a logic function with a signal in the model, which can be used as an input to other logic functions. A logic-gate is declared as follows:

```
.names <in-1> <in-2> ... <in-n> <output>
<single-output-cover>
```

A **generic-latch** is used to create a delay element in a model. It represents one bit of memory or state information. The generic-latch construct can be used to create any type of latch or flip-flop. A generic-latch is declared as follows:

```
.latch <input> <output> [<type> <control>] [<init-val>]
```

A **model-reference** is used to insert the logic functions of one model into the body of another. It is defined as follows:

```
.subckt <model-name> <formal-actual-list>
```

A .subckt construct can be viewed as creating a copy of the logic functions of the called model model-name, including all of model-name's generic-latches, in the calling model.

The following examples consists of an OR gate fed by 2 2-input AND gates, and a 4 bit adder made from 4 1 bit adders:

OR gate fed by 2 2-input AND gates

```
.model and\_or
.inputs a c d b
.outputs f
.names a c d b f
0110 1
1110 1
1001 1
1101 1
1011 1
0111 1
1111 1
.end
```

4 bit adder made from 4 1 bit adders

```
.model 4bitadder
.inputs A3 A2 A1 A0 B3 B2 B1 B0 CIN
.outputs COUT S3 S2 S1 S0
.subckt fulladder a=A0 b=B0 cin=CIN     s=S0 cout=CARRY1
.subckt fulladder a=A3 b=B3 cin=CARRY3 s=S3 cout=COUT
.subckt fulladder b=B1 a=A1 cin=CARRY1 s=XX cout=CARRY2
.subckt fulladder a=JJ b=B2 cin=CARRY2 s=S2 cout=CARRY3
                  # for the sake of example,
.names XX S1      # formal output 's' does not fanout to a primary output
1 1
.names A2 JJ      # formal input 'a' does not fanin from a primary input
1 1
.end
.model fulladder
.inputs a b cin
.outputs s cout
.names a b k
10 1
01 1
.names k cin s
10 1
01 1
.names a b cin cout
11- 1
1-1 1
-11 1
.end
```

3.2.2 BLIF output from Quartus

BLIF can be output from QIS in three different formats[22]:

1. **After RTL inference and LPM instantiation, but before technology independent multilevel logic optimization (MLS).** To dump BLIF before technology independent (MLS) optimization, one needs to turn on the following control variables:

- no_add_ops = on (adder-synthesis will not be performed because BLIF cannot support adder carry-chains)
- opt_dont_use_mac = on (DSP/multiply-accumulate blocks will not be used)
- dump_blif_before_optimize = on (the BLIF output will take place before gate-level optimizations)

2. **After technology independent optimization, but before technology mapping to LUTs.** To write BLIF after technology independent (MLS) optimization, one needs to turn on the following control variables:

   - no_add_ops = on (adder-synthesis will not be performed because BLIF cannot support adder carry-chains)
   - opt_dont_use_mac = on (DSP/multiply-accumulate blocks will not be used)
   - dump_blif_after_optimize = on (the BLIF output will take place after gate-level optimizations)

3. **After complete optimization and technology mapping.** To write BLIF after LUT mapping, one uses a different 3^{rd} setting:

   - no_add_ops = on (adder-synthesis will not be performed because BLIF cannot support adder carry-chains)
   - opt_dont_use_mac = on (DSP/multiply-accumulate blocks will not be used)
   - dump_blif_after_lut_map = on (the BLIF output will take place after LUT mapping)

There are several methods to set these variables. The method used was to add the appropriate TCL-like syntax to quartus settings file (QSF), which is named <projectname>.qsf once the project has created. This example sets all three variables.

set_global_assignment -name INI_VARS
"no_add_ops=on;opt_dont_use_mac=on;dump_blif_before_optimize=on"

It is important to note the differences between these three formats. In the first case, the assumption is that the research goal is to do gate-level synthesis on the design. In the second case, you have a good starting point for evaluating a new technology mapping algorithm. In the third case, your goal is either to extract a comparison point from QIS, or to use the mapped netlist for some other purpose (e.g. to convert to .net format for VPR).

Hierarchical BLIF: Black Box Primitive Support

If the BLIF writer encounters any FPGA hard block such as a RAM or MAC block or any other unsupported gate, it gives an internal error. The supported gate types are combinational gates (and,or,xor,not), DFF, input pin, output pin, bidir, buffers (carry-sum,expander,cut and other wire buffers), LUT, tri-bus and I/O buffer (tri-state, open-drain).

Quartus II black box primitives are supported to BLIF. Any block in the netlist that is not supported by traditional BLIF will be turned into a black box leading to a hierarchical BLIF netlist. Black boxes are instantiated in the current model as sub-circuits using the .subckt construct. Every black box needs to be defined outside of the current model. It requires a name, an interface consisting of inputs and outputs and the keyword ".blackbox". The support of black box primitives is activated by adding the following variable to the QSF file:

set_global_assignment -name INI_VARS "dump_blif_with_blackboxes=on"

An example of 6x6bits multiplier in Verilog and the corresponding hierarchical BLIF netlist from Quartus is shown below:
VHDL

```
module mult(a,b,f);
input [5:0] a;
input [5:0] b;
output [11:0] f;
assign f = a * b;
endmodule
```

BLIF output

```
.model mult
.inputs a[0] a[1] a[2] a[3] a[4] a[5] b[0] b[1] b[2] b[3] b[4] b[5]
.outputs f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9] f[10] f[11]
.subckt blackbox_g1 g14=g14 g15=g15 g16=g16 g17=g17 g18=g18 g19=g19
g20=g20 g21=g21 g22=g22 g23=g23 g24=g24 g25=g25 f[0]=f[0] f[1]=f[1]
f[2]=f[2] f[3]=f[3] f[4]=f[4] f[5]=f[5] f[6]=f[6] f[7]=f[7] f[8]=f[8]
f[9]=f[9] f[10]=f[10] f[11]=f[11]
.subckt blackbox_g14 a[0]=a[0] a[1]=a[1] a[2]=a[2] a[3]=a[3] a[4]=a[4]
a[5]=a[5] b[0]=b[0] b[1]=b[1] b[2]=b[2] b[3]=b[3] b[4]=b[4] b[5]=b[5]
g14=g14 g15=g15 g16=g16 g17=g17 g18=g18 g19=g19 g20=g20 g21=g21 g22=g22
g23=g23 g24=g24 g25=g25
.end
# blackbox_g1 = lpm_mult:Mult0|mult_fl01:auto_generated|result[0]
#g14 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2
#g15 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT1
#g16 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT2
#g17 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT3
#g18 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT4
#g19 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT5
#g20 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT6
#g21 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT7
#g22 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT8
#g23 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT9
#g24 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT10
#g25 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT11
#f[0] = lpm_mult:Mult0|mult_fl01:auto_generated|result[0]
#f[1] = lpm_mult:Mult0|mult_fl01:auto_generated|result[1]
#f[2] = lpm_mult:Mult0|mult_fl01:auto_generated|result[2]
#f[3] = lpm_mult:Mult0|mult_fl01:auto_generated|result[3]
#f[4] = lpm_mult:Mult0|mult_fl01:auto_generated|result[4]
#f[5] = lpm_mult:Mult0|mult_fl01:auto_generated|result[5]
#f[6] = lpm_mult:Mult0|mult_fl01:auto_generated|result[6]
#f[7] = lpm_mult:Mult0|mult_fl01:auto_generated|result[7]
#f[8] = lpm_mult:Mult0|mult_fl01:auto_generated|result[8]
#f[9] = lpm_mult:Mult0|mult_fl01:auto_generated|result[9]
#f[10] = lpm_mult:Mult0|mult_fl01:auto_generated|result[10]
#f[11] = lpm_mult:Mult0|mult_fl01:auto_generated|result[11]
.model blackbox_g1
```

```
.inputs g14 g15 g16 g17 g18 g19 g20 g21 g22 g23 g24 g25
.outputs f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9] f[10] f[11]
.blackbox
.end
# blackbox_g14 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2
#g14 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2
#g15 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT1
#g16 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT2
#g17 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT3
#g18 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT4
#g19 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT5
#g20 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT6
#g21 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT7
#g22 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT8
#g23 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT9
#g24 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT10
#g25 = lpm_mult:Mult0|mult_fl01:auto_generated|mac_mult2~DATAOUT11
.model blackbox_g14
.inputs a[0] a[1] a[2] a[3] a[4] a[5] b[0] b[1] b[2] b[3] b[4] b[5]
.outputs g14 g15 g16 g17 g18 g19 g20 g21 g22 g23 g24 g25
.blackbox
.end
```

The complete variable added to the QSF file in order to output an hierarchical blifs with blackbox primitives is :

set_global_assignment -name INI_VARS "no_add_ops=on;
dump_blif_with_blackboxes=on;dump_blif_after_lut_map = on;opt_dont_use_mac=on"

3.3 Powermodel Ace

In NAROUTO framework the activity estimator used is the current version of ACE, ACE 2.0 (see section 2.1.3). ACE calculates static probability and transision density of each pin beggining from primary inputs to primary outputs, but it doesn't support hierarchical blifs.

In order to compute activities in hierarchical blifs a tool was created to prepare the blif netlist for ACE. This tool is called *hb_for_ace* (transform Hierarchical Blifs for ACE), its open source and its written in perl.

It removes blackboxes from blifs (not functional removal, hardcoded), it adds all blackboxes inputs to the main model's primary outputs and all blackboxes outputs to the main model's primary inputs. The algorithm of this tool is shown in algorithm figure 1. It takes as input the hierarchical blif file and outputs a blif appropriate for ACE.

Ace as said before starts from primary inputs. Treating blackbox outputs as primary inputs it means that ACE will give them static probability 0.5 and transition density 0.2 (as in all primary inputs if its not specified differently). These are acceptable average values because we don't know the outputs of blackboxes, and ACE can continue normally to compute the activities of the rest of the pins.

Ace ends at primary outputs. So if blackbox inputs are included in primary outputs their static probability and transition density will be computed normally, all the signals will be calculated iteratively until the blackbox inputs and primary outputs.

In an abstract level view in an heterogeneous FPGA an heterogeneous unique block (of whatever type, DSP, RAM, e.t.c.) it behaves as a blackbox. The rest of Logic Elements of FPGA circuitry outputs some signals that are processed by the blackbox. This unique block outputs some signals that are fed back to the Logic Elements as inputs. This process its equivalent to an external blackbox that communicates with the FPGA through primary inputs and outputs.

After the use of *hb_for_ace*, the new BLIF is input to ACE which it outputs an .act file containing the activities of each signal in the design circuitry.

Algorithm 1 hb_for_ace

```
hb_for_ace(input_blif){
        blb_inputs();
        blb_outputs();
        primary_inputs();
        primary_outputs();
        primary_inputs()=grab_primary_inputs;
        primary_outputs()=grab_primary_outputs;
        for all blackboxes do
        {
                ins()=grab_blackbox_inputs(input_blif);
                outs()=grab_blackbox_ouputs(input_blif);
                add(blb_inputs,ins);
                add(blb_outputs,outs);
        }

        delete_subcircuits(input_blif);
        delete_blackbox_models(input_blif);

        add(primary_inputs,blb_outputs);
        add(primary_outputs,blb_inputs);

        printout_final_blif();
}
```

ACE 2.0 it is invoked by the following command :

```
./ace_linux
                                  --+
    -b [circuitname.blif]           | required
                                  --+

                                  --+
    -o [output activity filename] | optional
                                  --+

                                  --+
    -a [input activity filename]  | optional
          or                      |
    -v [input vector filename]    |
          or                      |
    -p [PI static probability]    |
    -d [PI switching activity]    |
                                  --+
```

3.4 HBT-Vpack

HBT-Vpack its a merging of Heterogeneity supporting T-Vpack of VPR 5.0 and powermodel's modified T-Vpack 4.30. The powermodel's code for T-Vpack was annotated to the new Heterogeneous T-Vpack.

It takes as input the hierarchical blif outputed from Quartus II and it outputs an .ac2, a .fun file and a .net file. The options are these of Heterogeneous T-Vpack of VPR 5.0 framework :

1. **Architecture Description Options That Are Always Valid**

   -lut_size <int>
   Number of inputs per LUT (i.e. K).
   Default: 4.

   -no_clustering
   Specifies that no clustering is to be performed – i.e. the logic block consists of one BLE (a LUT and a FF) with no local routing.
   Default: cluster.

   -global_clocks {on | off}
   Indicates whether clocks should be marked as being routed via a special, global resource. VPR does not route global signals.
   Default: on.

2. **Architecture Options Valid Only When -no_clustering Is Not Specified**

   -cluster_size <int>
   Number of BLEs in a cluster-based logic block (i.e. N).
   Default: 1.

   -inputs_per_cluster <int>
   Number of distinct inputs in a logic cluster (i.e. I).
   Default: lut_size × cluster_size.

   -clocks_per_cluster <int>
   Number of distinct clocks in a logic cluster.
   Default: 1.

   -muxes_to_cluster_output_pins {on | off}
   If "off", each BLE output is hooked directly to a cluster output pin. If "on", a set of N (one per cluster output) N:1 multiplexers

allows each output pin to be driven by any of the N BLEs within a cluster.
Default: off.

3. **CAD Optimization Options**

-timing_driven {on | off}
Controls whether the clustering algorithm attempts to optimize circuit timing by attempting to capture critical connections within a logic cluster.
Default: on.

-connection_driven {on | off}
Controls whether or not T-VPack attempts to absorb, within one cluster, connections from the output of one BLE to the input of another.
Default: off.

-hill_climbing {on | off}
Controls whether the algorithm used to pack BLEs into clusters allows hill climbing or is strictly greedy.
Default: on.

-cluster_seed {timing | max_inputs}
Specifies the way in which the cluster packing algorithm picks the first BLE to be placed in an empty cluster. Max_inputs picks the BLE with the most used inputs, while timing picks the BLE on the most critical path.
Default: timing if timing_driven is on, max_inputs otherwise.

-allow_unrelated_clustering {on | off}
Controls whether or not BLEs with no attraction to the current cluster can be packed into it.
Default: on.

-alpha <float>
A tradeoff parameter that controls the optimization of delay in packing vs. the optimization of signal sharing. A value of 0 focuses solely on signal sharing, while a value of 1 focuses solely on timing. This option is meaningful only when timing_driven is on.
Default: 0.75.

-recompute_timing_after <int>
T-VPack will recompute its estimate of how timing-critical each connection is after packing the specified number of BLEs into clusters. This option is meaningful only when timing_driven is on.
Default: 32 000.

-block_delay <float>
The relative delay of a BLE. This option is meaningful only when timing_driven is on.
Default: 0.1.

-intra_cluster_net_delay <float>
The relative delay of a signal that goes from one BLE to another using the local routing within a cluster. This option is meaningful only when timing_driven is on.
Default: 0.1.

-inter_cluster_net_delay <float>
The relative delay of a signal that goes from one BLE to another BLE that is in a different cluster, or an I/O pad. This option is meaningful only when timing_driven is on.
Default: 1.0.

-allow_early_exit {on | off}
If on, the clusterer will stop re-timing analyzing a circuit once it believes the current, partially complete packing, has fixed ("locked") the critical path.
Default off.

In .ac2 file each global net, net that connect clusters and subblock in the design, has its corresponding probability and the transition density values listed, as shown in the example (table 3.1).

In file .fun each subblock its described by its name and the corresponding logic function implemented in the subblock. This is shown in the example (table 3.2) .

The .net file that outputs the HBT-Vpack contains the blackboxes of the design as show in the example (table 3.3).

```
global_net_probability clk 0.500000
global_net_density clk 1.000000
        .
        .
        .
intercluster_net_probability g103 0.329200
intercluster_net_density g103 0.321000
intercluster_net_probability g409 0.509400
intercluster_net_density g409 0.197000
intercluster_net_probability g411 0.501600
intercluster_net_density g411 0.206000
intercluster_net_probability g413 0.498400
intercluster_net_density g413 0.204600
        .
        .
        .
subblock_probabiliy g281 0.048000 0.000000 0.000000 0.500000
0.000000 0.048000
subblock_density g281 0.026400 0.000000 0.000000 0.000000
1.000000 0.000000 0.026400
subblock_probabiliy g282 0.078000 0.025600 0.042200 0.000000
0.000000 0.001200
subblock_density g282 0.052600 0.051000 0.026800 0.024800
0.000000 0.000000 0.002600
subblock_probabiliy g283 0.048000 0.001200 0.517400 0.000000
0.000000 0.050400
subblock_density g283 0.026400 0.002600 0.206200 0.080200
0.000000 0.000000 0.050800
subblock_probabiliy g284 0.533800 0.017400 0.040000 0.000000
0.000000 0.048000
subblock_density g284 0.194000 0.035000 0.080200 0.050800
0.000000 0.000000 0.026400
        .
        .
        .
```

Table 3.1: Ac2 file example

```
        .
        .
        .
subblock_function g80 0100011100110011
subblock_function g84 0000110000111111
subblock_function g86 0000110000111111
subblock_function g87 0011000100111101
subblock_function g89 0000111111111111
subblock_function g90 0101101001100110
subblock_function g91 0100011100110011
subblock_function g92 0000111100110011
subblock_function g97 0000110000111111
subblock_function g99 0000110000111111
subblock_function g100 0100110001001111
subblock_function g101 0111001101000011
subblock_function g102 0001011100110101
subblock_function g103 0011000100111101
subblock_function g104 0000100011111111
subblock_function g106 1000101100110011
subblock_function g107 0011001100001111
        .
        .
        .
```

Table 3.2: Fun file example

```
        .
        .
        .
.blackbox_g130 blackbox_g130_4
pinlist: g406 clk g254 g409 g411 g413 g418 g420 g422 \
g415 g426 g428 g430 g434 g435 g436 g432 g131 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open

.blackbox_g154 blackbox_g154_5
pinlist: g406 clk g207 g409 g411 g413 g418 g420 g422 \
g415 g426 g428 g430 g434 g435 g436 g432 g155 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open

.blackbox_g169 blackbox_g169_6
pinlist: g406 clk g278 g409 g411 g413 g418 g420 g422 \
g415 g426 g428 g430 g434 g435 g436 g432 g170 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open

.blackbox_g198 blackbox_g198_7
pinlist: g406 clk g302 g409 g411 g413 g418 g420 g422 \
g415 g426 g428 g430 g434 g435 g436 g432 g199 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
        .
        .
        .
```

Table 3.3: Net file blackboxe's example

The HBT-Vpack is invoked by the command :

t-vpack input.blif output.net [-activity_ in activity.act] [-activity_ out activity.ac2]

3.5 Heterogeneity Support Toolset

HST (Heterogeneity Support Toolset) is a collection of open source tools written in perl to prepare the inputs that the VPR needs. It works for both VPR 5.0 and *HBVPR* (in NAROUTO framework).

3.5.1 Practical problems in Heterogeneity support

Even though the synthesis output from Quartus software can produce BLIF netlists with black-boxes, the resulted netlist is not logically equivalent with the original RTL description. This mainly occurs since BLIF format describes application's functionality at finer level. For instance, application's logic is translated at gate level, while memories are usually reported at word level. For example, an 128 $\times$ 8 bit RAM block will be encoded as 128 blackboxes, in the output blif.

If an hierarchical Blif such as that described above is given in HBT-Vpack as input it will result in a .net netlist with the same number of blackboxes representing individual cells. If this netlist is fed into VPR 5.0 or HBVPR it creates the following problems :

- The design is not realistic, because heterogeneous blocks are implemented as whole in commercial FPGAs.
- Because the number of blackboxes in .net file is greater than the actual number of heterogeneous blocks, the place and route step of the design flow requires more available slots for black-boxes, as well as significantly wider routing channels.
- The number of blackboxes in .net file can be so large (for example a 2048 $\times$ 8 bit RAM block will result in 2048 blackboxes!!!) than VPR, or any other Place and Route tool can't handle.

Another problem that creates the support of heterogeneity in VPR 5.0 (and in *HBVPR* which is based in VPR 5.0), is that for every .net file that includes blackboxes one needs a corresponding architectural file in .xml format, but until now the only way for this file to be created was manual.

3.5.2 HST Tools

3.5.2.1 Blackbox-aware technology mapping

In order to overcome the problem of breaking an heterogeneous block into individual cells of finer logic by the Quartus II, a blackbox-aware technology mapping must be applied to the design before this is placed and routed by VPR.

Blackbox_Profiler tool was created for this reason. Blackbox_Profiler's algorithm its based in the fact that sub-blackboxes that belong to a single hardware block have the same signals as the hardware block for controlling and communicating with the rest of the FPGA logic elements (for example a RAM hardware block may have read/write input to prevent simultaneous read and write in the same address).

Algorithm figure 2 shows the algorithm that performs this black-box-aware technology mapping.

Algorithm 2 Blackbox_Profiler

```
function blackbox-aware technology mapping

{

types()=find_diffrerent_types(blackboxes);

new_blackboxes()=types();

}
```

Blackbox_Profiler takes as input the design circuitry's netlist in .net format which is output in HBT-Vpack or T-Vpack, and the hierarchical blif that Quartus II outputs. It outputs a .net.res file that shows the results blackbox-aware technology mapping and its needed by various tools in HST, and a log file that contains informations about the mapping.

Table 3.4 shows the contents of a log file.

```
  Blackbox name     : .blackbox_g70 blackbox_g70_0
  Blackboxes merged : 8
  Total pins        : 32
  Input pins        : 24
  Output pins       : 8
  Initial Inputs    : 17
  Initial outputs   : 1


--- Total number of 8 blackboxes merged in 1 ---
--- Input file: oc_minirisc.net  Output file: Packed-oc_minirisc.net
```

Table 3.4: Blackbox_Profiler log file

3.5.2.2 Blackbox Packing

After the profiling of blackboxes the initial .net netlist from T-Vpack or HBT-Vpack can be updated to reflect the results of Blackbox_Profiler.

Blackbox packing can be done in two levels.

Blackbox Packing level 1

In level 1 packing, the packer groups all the subcomponents that belong to a single hardware block into a distinct black-box instance. Its algorithm can be shown in 3.

Blackbox_Packing level 1 takes as input the design circuitry's netlist in .net format which is output in HBT-Vpack or T-Vpack, the hierarchical blif that Quartus II outputs and information from Blackbox_Profiler. Then it outputs the new blackbox-aware technology mapped netlist in .net format ready for Heterogeneous VPR and HBVPR.

Example 3.5 shows an example of a blackbox instance in a netlist, before and after blackbox-aware technology mapping

Blackbox Packing level 2

In heterogeneous FPGAs some times unique hardware blocks can be part

Algorithm 3 Blackbox Packing level 1

```
function Level_1_packing
{
for all blackboxes do
  {
  x=types(0);
  while (not_matched)

    {

    if (blackbox_is_of_type(x))

      {
      multiplex(blackbox,new_blackboxes(x));
      not_matched=FALSE;
      }

    else x= type->next;

    }
}
}
```

.blackbox_g70 blackbox_g70_0
pinlist: g406 clk g103 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g71 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g82 blackbox_g82_1
pinlist: g406 clk g143 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g83 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g95 blackbox_g95_2
pinlist: g406 clk g231 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g96 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g117 blackbox_g117_3
pinlist: g406 clk g179 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g118 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g130 blackbox_g130_4
pinlist: g406 clk g254 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g131 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g154 blackbox_g154_5
pinlist: g406 clk g207 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g155 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g169 blackbox_g169_6
pinlist: g406 clk g278 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g170 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open
.blackbox_g198 blackbox_g198_7
pinlist: g406 clk g302 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g199 open
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 open

.blackbox_g70 blackbox_g70_0
pinlist: g406 g103 g302 g278 g207 g254 g179 g231 g143 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g71 g199 g170 g155 g131 g118 g96 g83 clk
subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Table 3.5: Blackbox example before (top) and after (bottom) blackbox-aware technology mapping

of a larger block. For example , 2 × 1Kbyte RAM blocks can be packed into one larger RAM hardware block. In odrer to expand the exploration space of research in placement and routing of NAROUTO framework, a second level of blackbox packing developed.

Blackbox_Packing level 2 , groups all blackboxes that a netlist contains, in one larger blackbox. The algorithm of the tool is shown in algorithm figure 4 .

Algorithm 4 Blackbox_Packing Level 2

```
function group_all_blackboxes
{
all_inputs();

all_outputs();

for all blackboxes do
        {
        ins() = grab_inputs(blackbox);
        outs()= grab_outputs(blackbox);

        add(all_inputs,ins);
        add(all_outputs,outs);
        }

new_blackbox = build_blackbox(all_inputs,all_outputs);

}
```

It takes as input the design circuitry's netlist in .net format which is output in HBT-Vpack, T-Vpack, or level 1 Packer, information from Blackbox_Profiler and the hierarchical blif that Quartus II outputs. Then Blackbox Packing level 2 outputs the new netlist, with grouped blackboxes, in .net format.

Example 3.6 shows an example of two blackboxes in a netlist after packing level 1, and after packing level 2.

.blackbox_g394 blackbox_g394_0 pinlist: g412 g564 g572 g571 g570 g569 g568 g567 g566 g298 g297 g296 g295 g294 g293 g292 g486 g485 g484 g482 g480 g479 g477 g395 g452 g446 g440 g434 g428 g422 g416 CLK_I subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 **.blackbox_g747 blackbox_g747_8** pinlist: g744 g795 g929 g897 g873 g872 g871 g860 g859 g135 g136 g140 g141 g145 g146 g150 g277 g279 g282 g284 g285 g287 g289 g748 g907 g864 g837 g831 g825 g812 g806 CLK_I subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
.blackbox_g394 blackbox_g394_0 pinlist: g929 g897 g873 g872 g871 g860 g859 g289 g287 g285 g284 g282 g279 g277 g150 g146 g145 g141 g140 g136 g135 g795 g744 g572 g571 g570 g569 g568 g567 g566 g412 g564 g298 g297 g296 g295 g294 g293 g292 g486 g485 g484 g482 g480 g479 g477 g395 g416 g422 g428 g434 g440 g446 g452 g748 g806 g812 g825 g831 g837 g864 g907 CLK_I subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62

Table 3.6: Blackbox example after level 1 packing (top) and after level 2 packing (bottom)

3.5.2.3 Multiplexer

Since BLIF format describes application's functionality at finer level that results in demultiplexion of the signals of the heterogeneous hardware block. The Blackbox Packing tool may result a blackbox instantiation with 700 pins!!!. The channel width of intercluster connections needed for this block to be routed by VPR or any similar tool is significantly larger than a real RAM block or DSP would need (if VPR can even handle this channel width).

Multiplexer is a tool that transforms the blackbox instantiation in a .net type netlist of the design in a realistic –concerning the number of pins– heterogeneous hardware block.

In the development of Multiplexer tool some practical problems concerning the integrity of the initial circuitry design occurred. If the method for reducing the input and output pins of blackboxes was to simply merge many signals into one that would undermine the structural and functional integrity of .net netlist. For example : if every reference in the netlist of signals g30, g31, g32, g33, g34 to become reference of signal g3_0, some nets will result in multiple fanin and VPR will result in unexpected exit with "error in netlist" error.

The method used to reduce the number of pins, was to "simulate" a multiplexer through clbs. Input signals of a blackbox, first pass through "multiplexing" clbs, and the new multiplexed signals are the new inputs of a blackbox. Output signals of a blackbox are multiplexed and pass through "demultiplexing" clbs before connecting with the rest of the clbs. That way the rest of the design in the netlist file remains intact. If it is needed the new inputs and outputs of the blackbox will be multiplexed again (as many times its needed).

The Multiplexer tool algorithm is shown in algorithm figure 5.

One disadvantage of this method is that adds to the existing netlist extra clbs. This means utilization of more FPGA fabric than the design actually needs. On the other hand this penalty its easily computed since the number of axtra clbs is known to the designer. Additionally the percentage of extra clbs compared to the total number of clb's a design's netlist contains is in most cases 2-4%, so the penalties this creates can be overlooked.

Multiplexer needs as input the netlist of a design in blif and net format (.net file is the result of Blackbox Packing tool), and information from Blackbox_Profiler. It outputs a .net.res file that its needed by net2xml tool, and a log file that contains informations about the multiplexing.

Algorithm 5 Multiplexer

```
function multiplexe-pins
(input_compression_level, output_compression_level)

{

inputs();
outputs();
extra_clbs();

for all blackboxes do
  {
        inputs()=grab_blackbox_inputs();
        outputs()=grab_blackbox_outputs();

        for i=0 to i=input_compression_level{

                new_signals()=create_new_input_names(inputs);
                new_clbs=create_in_clbs(inputs,new_signals);

                add(extra_clbs,new_clbs);

                inputs()=new_signals;
        }

        for i=0 to i=output_compression_level{

                new_signals()=create_new_output_names(outputs);
                new_clbs=create_out_clbs(outputs,new_signals);

                add(extra_clbs,new_clbs);

                outputs()=new_signals;
        }
  }
}
```

.blackbox_g70 blackbox_g70_0 pinlist: g406 g103 g302 g278 g207 g254 g179 g231 g143 g409 g411 g413 g418 g420 g422 g415 g426 g428 g430 g434 g435 g436 g432 g71 g199 g170 g155 g131 g118 g96 g83 clk subblock: sub0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
.blackbox_g70 blackbox_g70_0 pinlist: JE_4 JE_5 JE_0 JE_3 JE_2 JE_1 SI_3 SI_0 SI_1 SI_2 clk subblock: sub0 0 1 2 3 4 5 6 7 8 9 10

Table 3.7: Blackbox example before (top) and after (bottom) I/O multiplexing

Example 3.7 shows an example of a blackbox instance in a netlist, before and after I/O multiplexing. In 3.8 the extra clb's added are presented, and 3.9 shows the contents of the corresponding log file.

```
.clb Inplex_1_0
pinlist: g406 g103 g409 g411 g413 g418 g420 g422 g415 g426 \
g428 g430 g434 g435 g436 g432 g143 g231 g179 g254 open open \
open open open open open JE_0 JE_1 JE_2 JE_3 JE_4 clk
subblock: JE_4 0 1 2 3 31 32
subblock: JE_3 4 5 6 7 30 32
subblock: JE_2 8 9 10 11 29 32
subblock: JE_1 12 13 14 15 28 32
subblock: JE_0 16 17 18 19 27 32

.clb Inplex_1_1
pinlist: g207 g278 g302 open open open open open open open \
open open open open open open open open open open open open \
open open open open open open open open open JE_5 clk
subblock: JE_5 0 1 2 open 31 32

.clb Outplex_1_0
pinlist: SI_0 SI_1 SI_2 SI_3 open open open open open open \
open open open open open open open open open open open open \
open open g71 g83 g96 g118 g131 g155 g170 g199 clk
subblock: g199 0 open open open 31 32
subblock: g170 1 open open open 30 32
subblock: g155 2 open open open 29 32
subblock: g131 3 open open open 28 32
subblock: g118 0 open open open 27 32
subblock: g96 1 open open open 26 32
subblock: g83 2 open open open 25 32
subblock: g71 3 open open open 24 32
```

Table 3.8: Extra clbs from blackbox I/O multiplexing

```
--- Number of clbs: 73 ---

--- Total inputs: 23  Total outputs: 8 ---

Input Compression level: 1 -- Inputs: 23=>6 -- Extra clbs: 2

Output Compression level: 1 -- Outputs: 8=>4 -- Extra clbs: 1

Input Compression level: 1 -- Output Compression level: 1

--- New File : PL-Packed-oc_minirisc.net ---
```

Table 3.9: Log file of Multiplexer tool

3.5.2.4 Net2xml

As mentioned in section 3.5.1 for every .net file that includes blackboxes one needs a corresponding unique architectural file in .xml format to use as input in VPR 5.0 (and HBVPR). In order to create this architectural file automatically a tool has been added in Heterogeneity Support Toolset.

Net2xml (that tool) can be used in two ways.

1. One can use the initial netlist that T-Vpack outputted along with the corresponding hierarchical blif,

2. or he can use the .net.res output of Blackbox_Profiler along with the netlist from either level 1 or 2 blackbox packing

There are three files that one can change manually to configure the architecture of the FPGA.

- **template.xml** that contains information about the FPGA fabric.
- **blackbox.conf** that contains information about blackboxes such as the position of pins and input to output delays.
- **power.xml** that contains information that needed in power estimation.

Net2xml has two modes of architecture creation:

- If no parameter used the .xml output is ready for VPR 5.0.
- If the <*-p*> parameter used, .xml output has power information that HBVPR needs.

Net2xml then outputs the appropriate .xml architecture file that HBVPR or VPR 5.0 needs.

It follows an example which is shows a blackbox architecture description.

```
<type name=".blackbox_g70" height="1">
  <subblocks max_subblocks="1" max_subblock_inputs="6"
     max_subblock_outputs="4">
    <timing>
      <T_comb>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
        <trow>2e-09 2e-09 2e-09 2e-09 </trow>
      </T_comb>
      <T_seq_in>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
      </T_seq_in>
      <T_seq_out>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
        <trow>-1e-10</trow>
      </T_seq_out>
    </timing>
  </subblocks>
  <fc_in type="frac">0.25</fc_in>
  <fc_out type="full" />{sec:multiplexer}

  <pinclasses>
    <class type="in">0</class>
    <class type="in">1</class>
    <class type="in">2</class>
```

```
    <class type="in">3</class>
    <class type="in">4</class>
    <class type="in">5</class>
    <class type="out">6 7 8 9 </class>
    <class type="global">10</class>
  </pinclasses>

  <pinlocations>
    <loc side="left">0 4 8 </loc>
    <loc side="top">1 5 9 </loc>
    <loc side="right">2 6 10 </loc>
    <loc side="bottom">3 7 </loc>
  </pinlocations>
  <gridlocations>
    <loc type="col" start="2" repeat="0" priority="3"/>
  </gridlocations>
  <timing>
    <tedge type="T_sblk_opin_to_sblk_ipin">2e-9</tedge>
    <tedge type="T_fb_ipin_to_sblk_ipin">3e-9</tedge>
    <tedge type="T_sblk_opin_to_fb_opin">4e-9</tedge>
  </timing>
 </type>
```

Below it is shown the power information needed in .xml architecture file for HBVPR power estimation.

```
    <clocks>
      <clock buffer_R="124" buffer_Cin="80e-15" buffer_Cout="
          200e-15" Rwire="40" Cwire="20e-15"
          Cin_per_clb_clock_pin="7.512e-15"/>
    </clocks>

    <power>
      <temperature_records>
        <temp NMOS_NFS="8.28e18" PMOS_NFS="8.32e18">-40</temp
            >
        <temp NMOS_NFS="8.50e18" PMOS_NFS="8.46e18">-30</temp
            >
        <temp NMOS_NFS="8.72e18" PMOS_NFS="8.62e18">-20</temp
            >
        <temp NMOS_NFS="8.94e18" PMOS_NFS="8.79e18">-10</temp
            >
        <temp NMOS_NFS="9.16e18" PMOS_NFS="8.96e18">0</temp>
        <temp NMOS_NFS="9.38e18" PMOS_NFS="8.14e18">10</temp>
```

```
        <temp NMOS_NFS="9.59e18" PMOS_NFS="9.31e18">20</temp>
        <temp NMOS_NFS="9.70e18" PMOS_NFS="9.49e18">25</temp>
        <temp NMOS_NFS="9.81e18" PMOS_NFS="9.40e18">30</temp>
        <temp NMOS_NFS="1.00e19" PMOS_NFS="9.67e18">40</temp>
        <temp NMOS_NFS="1.02e19" PMOS_NFS="9.85e18">50</temp>
        <temp NMOS_NFS="1.05e19" PMOS_NFS="1.00e18">60</temp>
        <temp NMOS_NFS="1.07e19" PMOS_NFS="1.02e19">70</temp>
        <temp NMOS_NFS="1.09e19" PMOS_NFS="1.04e19">80</temp>
        <temp NMOS_NFS="1.11e19" PMOS_NFS="1.06e19">90</temp>
        <temp NMOS_NFS="1.13e19 " PMOS_NFS="1.07e19">100</
           temp>
      </temperature_records>
      <Nmos Vth="0.4" CJ="1E-3" CJSW="2E-10" CJSWG="5e-10"
         CGDO="3E-10" COX="8e-3" EC="4e6"/>
      <Pmos Vth="0.4" CJ="1E-3"  CJSW="2E-10" CJSWG="5e-10"
         CGDO="3E-10" COX="8e-3" EC="5e6"/>
      <poly Cpoly="1E-10" poly_extension="0.18e-6"/>
      <min_transistor_size length="0.18e-6" width="0.36e-6"/>
      <Vdd>1.8</Vdd>
      <Vswing>1.8</Vswing>
      <Vgs_for_leakage>0.2</Vgs_for_leakage>
      <SRAM_leakage>0</SRAM_leakage>
      <short_circuit_power_percentage>0.1</
         short_circuit_power_percentage>
    </power>
```

3.5.2.5 Activity_Updater

After multiplexing the pins of the blackboxes (section 3.5.2.3), the .ac2 activity file must be updated to include the signals of added "multiplexing" clbs.

The Activity_Updater tool of HST toolset generates the new activities of each new signal. The algorithm is calculating the average static probability and transition density of the signals multiplexed to output new signal's corresponding values. The algorithm is shown in algorithm figure 6.

Algorithm 6 Activity_Updater

```
function update_activities
{
signals();
activities();
for all new_signals do
{
        x=current_signal;
        signals()=find_all_signals_multiplexed_in(x);

        prob=calculate_average_prob(signals);
        dens=calculate_average_dens(signals);

        add(activities, prob, dens);

}
```

Activity_Updater takes as input, the .act and .ac2 activity files of the design, and the .net netlist the Multiplexer tool outputs. It generates a new .ac2 activity file that will be used from HBVPR.

3.6 HBVPR

HBVPR is a Placement & Routing tool in NAROUTO framework that supports power estimation in both homogeneous and heterogeneous FPGA designs. It is based in VPR 5.0 (see section 2.1.2.2), Powermodel framework of VPR 4.30 (see section 2.1.3) and VPR 5.0 with power estimation (see section 2.1.6).

3.6.1 Power estimation

If an .ac2 activity file, an .fun function file, and an .xml architecture file with power concerning information are given, HBVPR estimates power dissipation of the given design. This step is done after the final Placement and routing of the design, so placement and routing are not power-aware.

The average power consumption in digital circuits consists of three main components:

- dynamic,
- short-circuit
- and leakage power.

3.6.1.1 Dynamic Power

Dynamic power is the dominant component of the total power. It is dissipated every time a signal changes due to the charging and discharging of load and parasitic capacitances. Therefore, dynamic power is closely related to the transition density of all nodes inside the circuit. The total dynamic power dissipation can be written as:

$$DynamicPower = \sum_{allnodes} 0.5 \times C_y \times V_{supply} \times V_{swing} \times D(y) \times f_{clk} \quad (3.1)$$

The expression $0.5 \times C_y \times V_{supply} \times V_{swing} \times D(y)$ determines the energy per clock cycle, where V_{swing} is the swing voltage of each node, V_{supply} is the supply voltage, D(y) is the transition density at node y, and C_y is the capacitance of node y that is charged and discharged during each transition. The dynamic power is then equal to the energy per clock cycle multiplied by

the clock frequency, f_{clk}, which is bounded by the critical path delay of the circuit.

To estimate dynamic power, the resources in an FPGA are separated into three categories: routing resources, logic blocks, and the clock network. The estimation of power dissipation by these resources is done in each category separately.

Routing Resource Dynamic Power

A large part of the dynamic power is due to switching tracks within the routing fabric of the FPGA. Since the power model needs to be flexible enough to model the power in any FPGA that can be described within VPR, and since the capacitances of the routing tracks vary greatly with the track length and the number of attached buffers, a single value for track capacitance is not enough. Instead, capacitance information is extracted from the routing resource graph within VPR for each metal track separately. Figure 3.4 shows an example metal track that spans four logic blocks and is attached to a number of programmable switches. In general, the capacitance of a track depends on the number of logic blocks spanned by the segment, the size of each logic block (since a larger logic block implies a longer metal track), the number of pins on each logic block, the switch block and connection block connectivities, and information about the target technology. Using this information, the overall capacitance of each track is estimated by adding the metal capacitance of the track itself and the parasitic capacitances of all switches attached to the track.

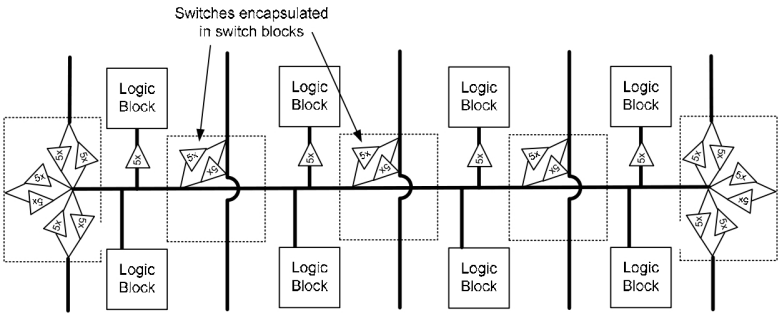

Figure 3.4: Example of an FPGA Routing Segment.

After calculating the capacitance information for each track, the overall dynamic power of the routing fabric is calculated. For each net in the design, the capacitance of all tracks that are used to route the net are summed, and the activity of the net is then used, along with this capacitance, to calculate the power dissipated by that net.

Logic Block Dynamic Power

Like the power model for the routing fabric, the power model for the logic block must be flexible. It must accommodate any lookup-table size, any number of lookup-tables in each cluster, and any number of inputs to each cluster. The model assumes the architecture in Figure 3.5, and consists of four components: the power dissipated in the lookup-tables, the power dissipated in the input multiplexers, the power dissipated in the flip-flops, and the power dissipated in the other nodes and wires within the logic block.

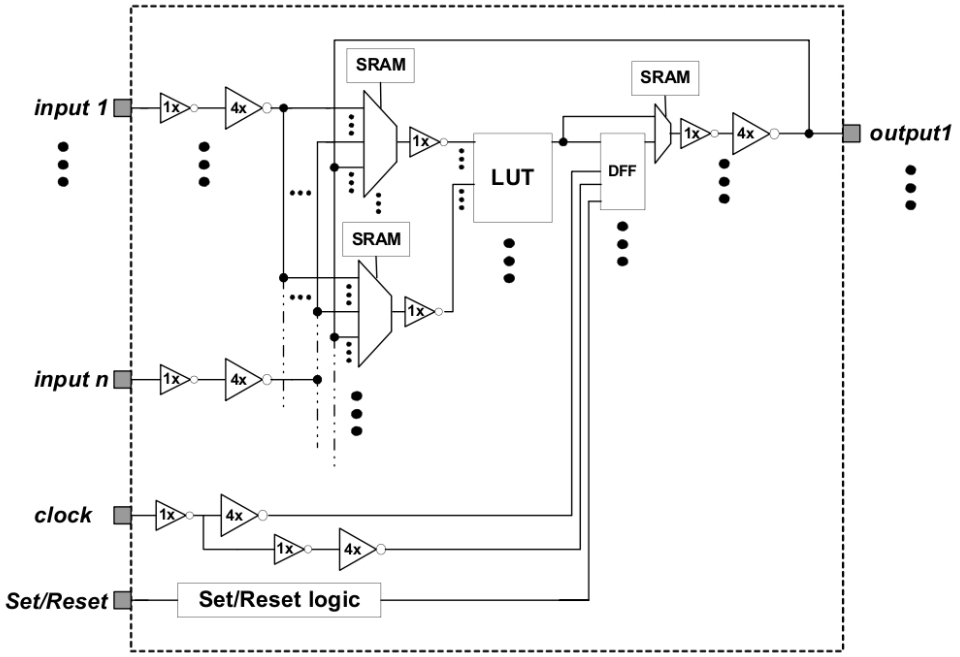

Figure 3.5: Schematic of a logic block.

Power Dissipated in the Lookup-Tables. Lookup-tables in FPGA's are commonly implemented as multiplexer trees. To estimate the power dissipated in a multiplexer tree, the tree is represented as a set of two-input multiplexers. Then from the transition density model (as before) the activity of each node within the lookup-table is calculated. The capacitance of each node within the lookup-table is estimated by noting that each node is associated with three source/drain capacitances and one gate capacitance (the gate capacitance is due to the Miller effect spread over two transistors).

Power Dissipated in the Input Multiplexers. The input multiplexers select the lookup-table input signals from among the routing tracks. Since these multiplexers are similar in structure to the lookup-tables, the modeling is similar. There are, however, two important differences. First, the gates of the pass transistors inside the LUTs are connected directly to the internal routing; therefore, the internal nodes inside the LUT can be affected by thev body effect of the pass transistors, and may swing at a degraded supply voltage. On the other hand, the gates of the pass transistors inside the input multiplexers are connected to SRAM cells. The SRAM cells are assumed to be powered by a higher voltage than the core voltage, meaning that the internal nodes inside the input multiplexers are not affected by the body effect and swing at the full core voltage.

A second reason that the input multiplexers have different power behavior than the multiplexers within the lookup tables is that the internal nodes within input multiplexers are often more correlated to each other than those within the LUTs. Such a phenomenon in LUTs may not happen as frequent as in the input multiplexers because the input signals to the LUTs can switch at different times.

Power Dissipated in Flip-Flops To determine the dynamic power dissipated inside each D-flip-flop in an FPGA logic block, the following model used.

$$DynamicPower(DFF) = 0.5 \times C_{DFF} \times (EffectiveDensity) \times V_{supply} \times V_{swing} \times f_{clk} \quad (3.2)$$

$$EffectiveDensity = (-0.074) \times D(input) + (5.2486) \times D(input) \quad (3.3)$$

where D(input) is the transition density of the input signal for the D-flip-flop, V_{supply} is the supply voltage, V_{swing} is the swing voltage, and f_{clk} is the clock frequency. The quantity C_{DFF} is the total capacitance of all nodes inside a flip-flop that toggle when a flip-flop changes state (this was estimated using reasonable transistor sizes and source/drain overlaps for our flip-flop circuit).

Power Dissipated in Clock Tree. Finally, the dynamic power of the clock network is determined by assuming an H-tree clock network. The clock network consists of a set of clock buffers connected using clock segments. The optimum number of clock buffers and clock segments, as well as the optimum buffer size, depends on the size of the FPGA. Since the power model needs to be flexible enough to estimate the power for any size FPGA, a method is used of predicting the number and size of the clock buffers and segments based on the size of the FPGA.

Given the number of logic blocks in the FPGA, the length of the longest path from the clock source to a flip-flop clock pin is calculated. Then a single path in the clock tree network its modeled as a distributed RC ladder network, as shown in figure 3.6. In general, there are M stages (corresponding to M clock buffers), and each clock buffer is of size N. After calculating these values of M and N, the power dissipated in the clock network can be calculated as before.

3.6.1.2 Short-Circuit Power

Short-circuit power is dissipated through a direct current path between the power supply and ground during each transition. Short-circuit power is a function of the rise and fall time and the load capacitance. Short-circuit power is modeled as 10% of the dynamic power calculated above.

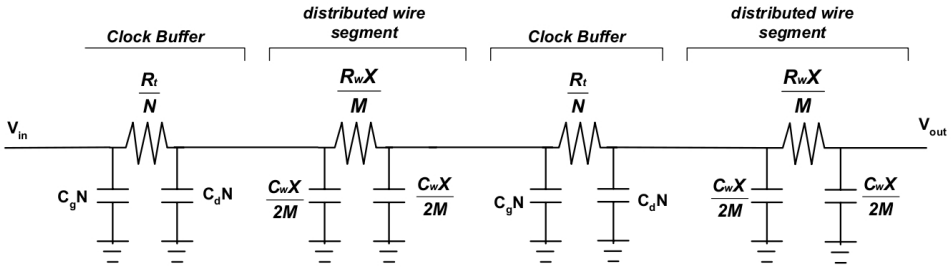

Figure 3.6: RC Ladder network corresponding to a clock tree with two clock buffers.

3.6.1.3 Leakage power

Leakage power dissipation comes from two sources: reverse-bias leakage power and sub-threshold leakage power. As the majority of leakage power is from sub-threshold current the reverse bias leakage current is assumed to be negligible. A first-order estimation model is applied to estimate the sub-threshold current.

All the logic blocks and routing switches, including the unused logic blocks and unused routing switches, are considered in the leakage power calculation. The leakage current of each SRAM cell can be defined by the users in the architecture input file in order to include the SRAM leakage in the power estimation.

3.6.1.4 Blackboxe's power estimation

As mentioned before in .net netlist all unique blocks are described as black-boxes. For this reason HBVPR tool can't calculate the power consumed by these hardware blocks. On the other hand, it does calculate the routing and clock power consumed by these blocks.

Complete final power estimation is easily done by simply adding to the power consumed by logic blocks, the power consumed by heterogeneous blocks. This can actually expand the research space in terms of power dissipation. The designer can effortlessly evaluate different types and hierarchies of appropriate unique blocks, like different types of DSP multipliers or different types and hierarchies of RAM memories. The power consumption of these blocks can be easily obtained from data sheets or even simulation.

3.6.2 Placement and routing

Placement and routing process in HBVPR is the same as in VPR 5.0. The basic X-Y coordinate system in HBVPR is dictated by the soft logic cluster tile: one grid unit in the X and Y directions is designated as the space taken up by the basic soft logic cluster. All other blocks must be multiples of this size. Hard blocks are restricted to be in one grid width column, and that column can be composed of only one type of block. Although this restriction prevents a more general cross-column approach, it appears sufficient for all but the extremely large hard blocks.

Each hard block may be broken in a different number of subblocks, not unlike the logic elements in a cluster. Each type of block may have different timing characteristics, routing connectivity, and height. The height of a block must be an integral number of grid units. This is shown in figure 3.7 which shows a top view of a placed and routed heterogeneous design with 8 blackboxes. In the event that a block's height is indivisible with the height of the core, some grid locations are left empty.

The blackckboxes are transparent in the routing step; this means that if a hard block spans multiple rows, the horizontal routing tracks pass through at every grid location, but there are no input or output pins from the block where the routing passes through, as shown in figure 3.8.

Figure 3.9 shows the connections from one blackbox to the global routing channel and figure 3.10 shows a top-view of the clbs a blackbox (the green one) is connected.

HBVPR (as VPR 5.0) models logic blocks, heterogeneous blocks, and I/Os using the same data structure. The timing parameters of all blocks are specified using a timing matrix. The timing in a subblock is modeled as a complete set of all possible delays from each input to each output of the subblock. Heterogeneous subblocks can have purely combinational or registered output.

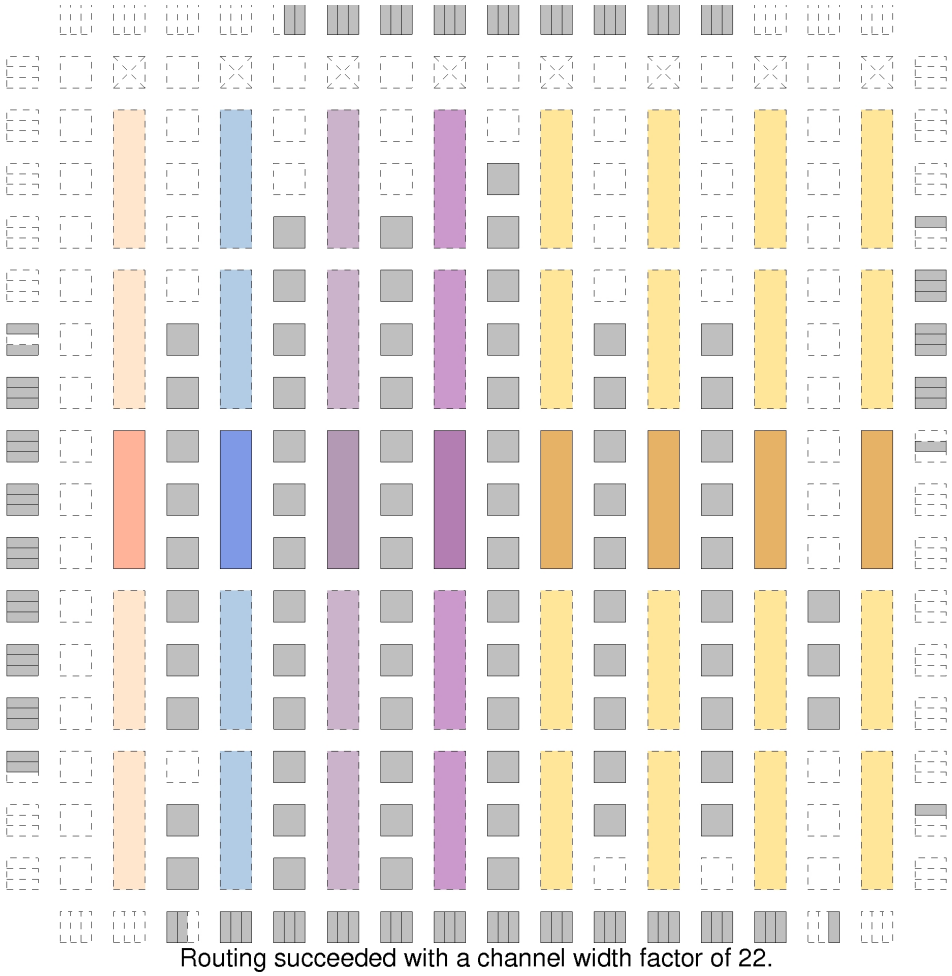

Figure 3.7: Top view of a heterogeneous design with 8 blackboxes. The different types of blackboxes are shown with different colors (VPR only supports 6 different colors) and have height three times the basic soft logic cluster.

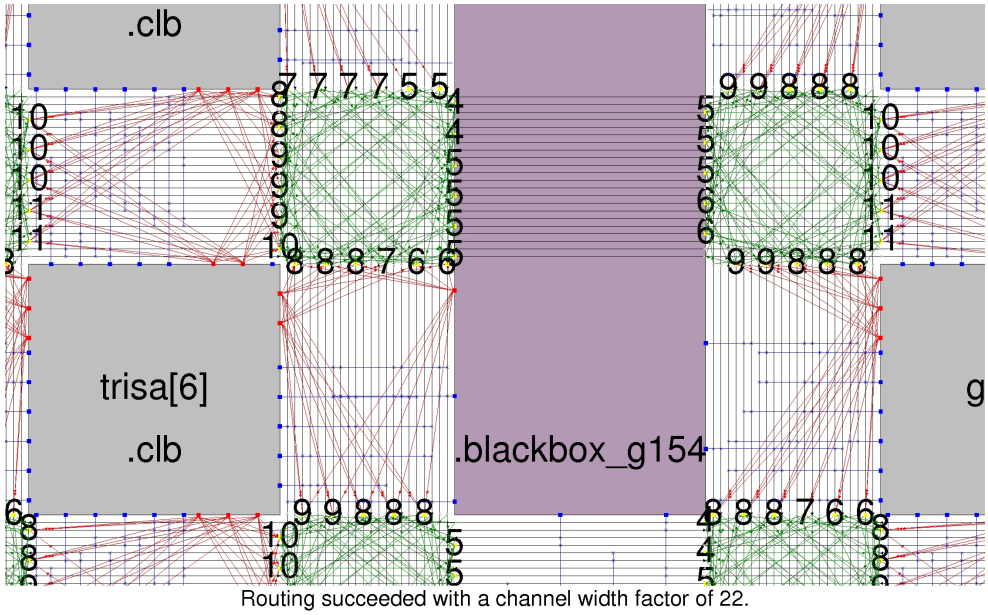

Figure 3.8: Horizontal routing tracks pass through at every grid location, even from blackboxes

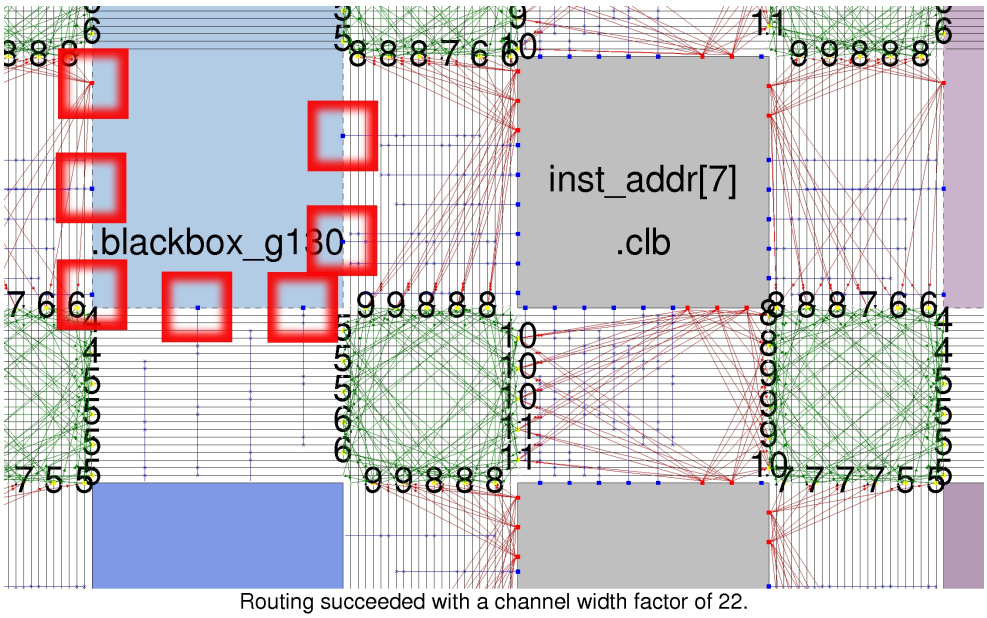

Figure 3.9: Blackbox connections to the routing channel (red squares)

3.7 DAGGER

NAROUTO framework if an FPGA design is homogeneous it can co-operate with MEANDER Framework. If one need to evaluate a heterogeneous design he can use in the last step of the design Flow the Dagger tool from MEANDER framework (see 2.2.1), that programs the AMDREL fine-grain reconfigurable hardware.

The input files that required by the DAGGER tool in order to generate the bitstream for the heterogeneous FPGA programming from NAROUTO framework:

- The netlist file that describes the circuit in ".NET" format which is ready after Multiplexer
- The output function file (.FUN) which is produced by the HBT-VPACK tool
- The FPGA architecture description file which is produced by the Net2xml tool (it needs to be converted again in .arch format)

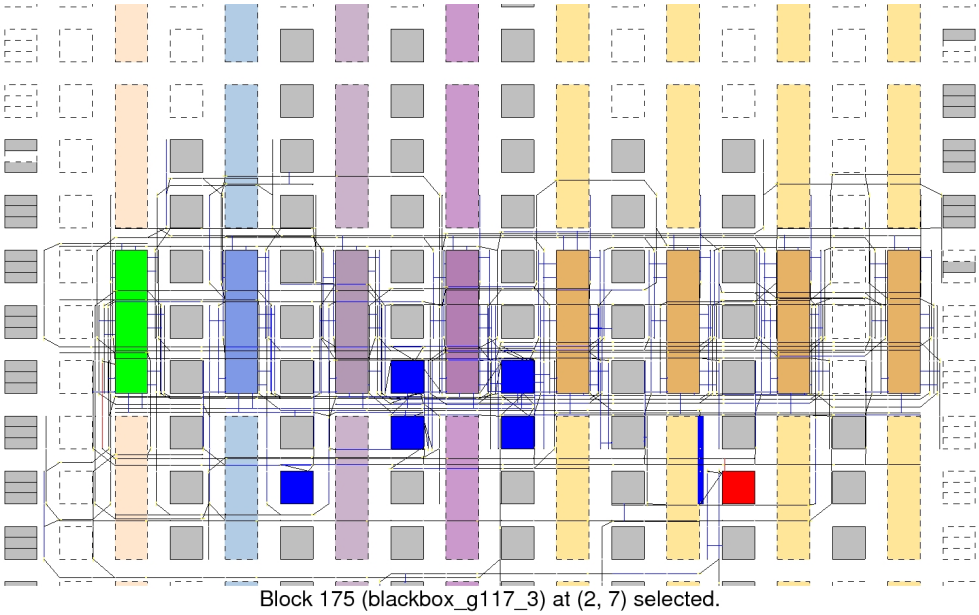

Figure 3.10: Blackbox connections to clbs (green blackbox is connected to blue and red clbs)

- The placement file (.P) of the circuit into the FPGA which is produced by the HBVPR tool
- The routing file (.R) of the circuit into the FPGA which is produced by the HBVPR tool

Chapter 4

Benchmarking

In this chapter the results of eight benchmarks are presented. The design flow used to evaluate these benchmarks is NAROUTO framework, from VHDL hardware description to Placement and Routing.

This chapter is divided in three distinct parts:

- The first section contains information about the benchmarks.
- In the second section the results of design circuitries from HST toolset are presented.
- In the third and last section of this chapter the benchmark designs are evaluated in terms of delay, area and power, that HBVPR resulted.

4.1 Benchmarks information

The six design used as benchmarks for NAROUTO framework are the following :

1. **oc_aes_core_inv.** Encryption-type design taken from Quip toolkit version 9.0
2. **oc_ata_ocidec3.** Processor, control-type design from Quip toolkit version 9.0
3. **oc_hdlc.** Processor, control-type design from Quip toolkit version 9.0

Benchmark	I/Os	4-LUTs	F/Fs	Ram bits
oc_aes_core_inv	389	5,144	536	34,176
oc_ata_ocidec3	130	1,589	594	224
oc_hdlc	82	859	926	2,048
oc_minirisc	389	908	300	1,024
oc_oc8051	189	4,306	754	4,608
os_blowfish	585	5,368	891	67,168

Table 4.1: Benchmark information

4. **oc_minirisc.** Processor, control-type design from Quip toolkit version 9.0

5. **oc_oc8051.** Processor, control-type design from Quip toolkit version 9.0

6. **os_blowfish.** Encryption-type design taken from Quip toolkit version 9.0

Table 4.1 contains information about the designs such as lut number, F/F number, memory bits, and I/O numbers

Benchmark	No. of Blackboxes		
	Initial	level 1 packing	level 2 packing
oc_aes_core_inv	128	1	-
oc_ata_ocidec3	32	1	-
oc_hdlc	16	2	1
oc_minirisc	8	1	-
oc_oc8051	67	21	1
os_blowfish	160	5	1

Table 4.2: Number of blackboxes before and after packing level 1 and 2. In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so in level 2 there is a "-"

4.2 Heterogeneity Support Toolset results

The benchmark designs are outputted from Quartus II in Hierarchical BLIF format that describes application's functionality at finer level. Then the resulted BLIFs were passed through hb_for_ace and ACE 2.0 in the activity estimation step of the flow.

HBT-Vpack packed the LUTs into clusters of the designs. For all the benchmarks the cluster size was 10, the lut size was 4 and inputs per cluster were set to 22.

After HBT-Vpack, tools from Heterogeneity Support Toolset were used to profile, pack, and multiplex the pins of blackboxes.

Table 4.2 shows the number of blackboxes before packing and after packing (level 1 and 2). In table 4.3 is shown an estimation of memory bits each blackbox after level 1 packing contains[1].

The blackboxes, before and after level 1 and level 2 packing are shown graphical in figures 4.1, 4.2 and 4.3 correspondingly. The images are taken from HBVPR after placement of oc_hdlc benchmark (the only one that has a "viewable" number of blackboxes and 2 levels of packing). The CLBs are the gray color squares, the blackboxes possible positions in the column are with a different light color and the actual blackboxes positions after placement are shown by a darker color (not grey).

In table 4.4 the results of pin Multiplexing are shown.

[1]In level 2 all blackboxes are merged into one, so the RAM bits of each blackbox is the total number as shown in table 4.1

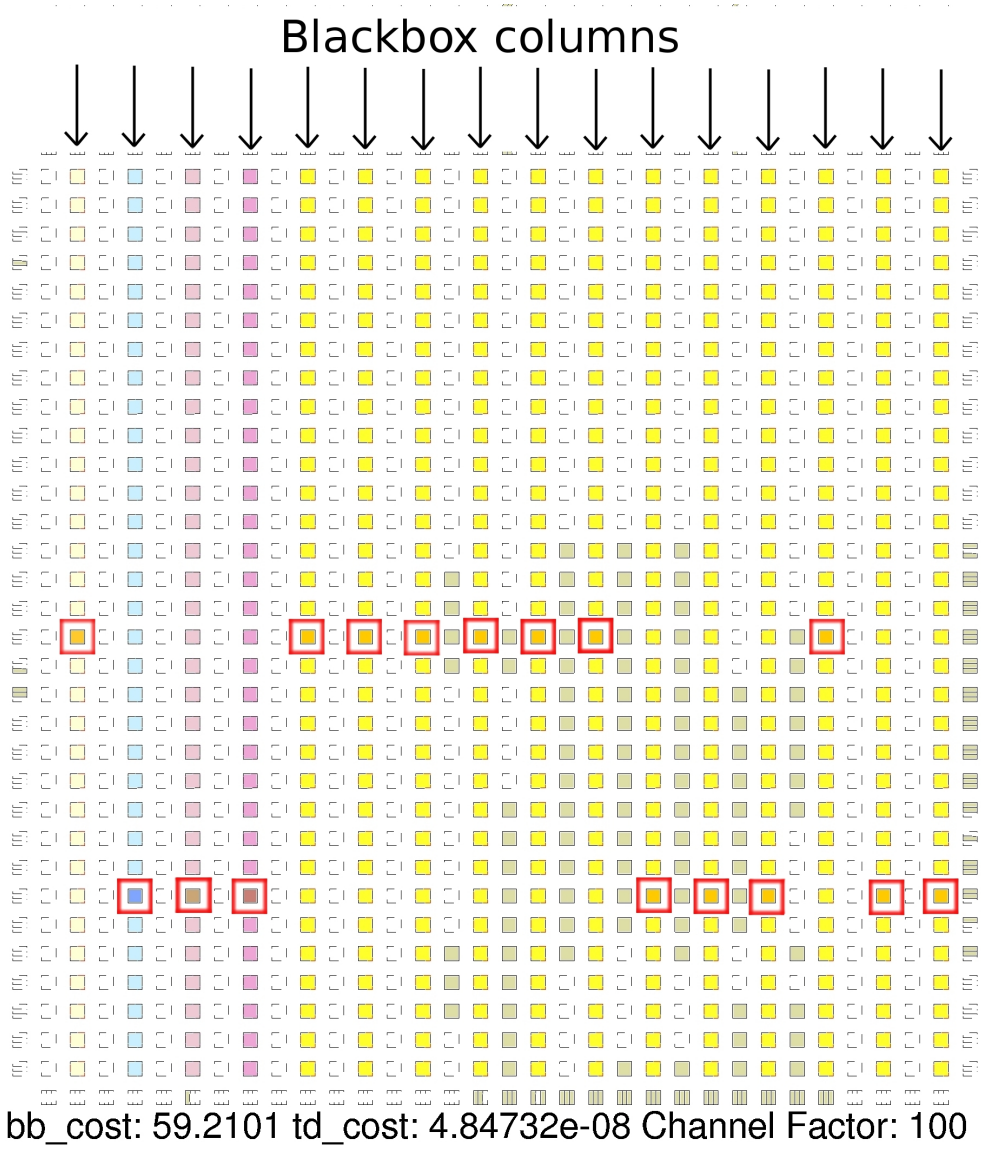

Figure 4.1: Oc_hdlc benchmark placement in HBVPR without Blackbox-aware technology mapping. The blackbox positions are marked with the red squares.

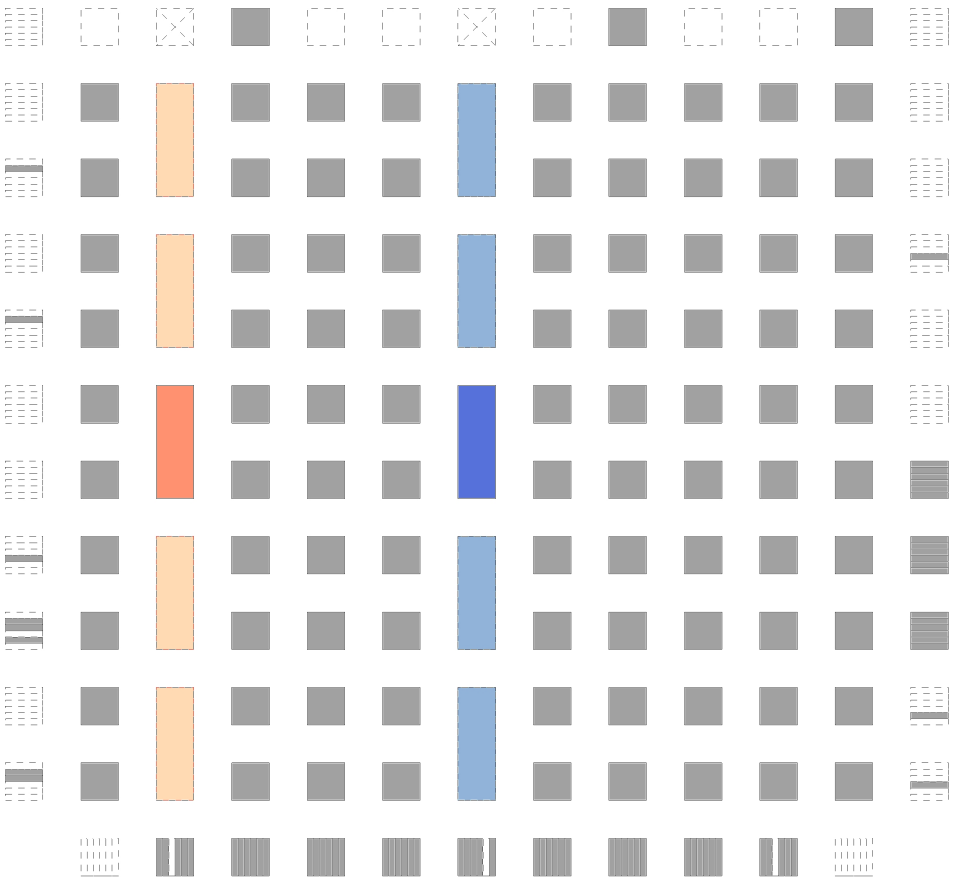

Figure 4.2: Oc_hdlc benchmark placement in HBVPR after level 1 Blackbox packing.

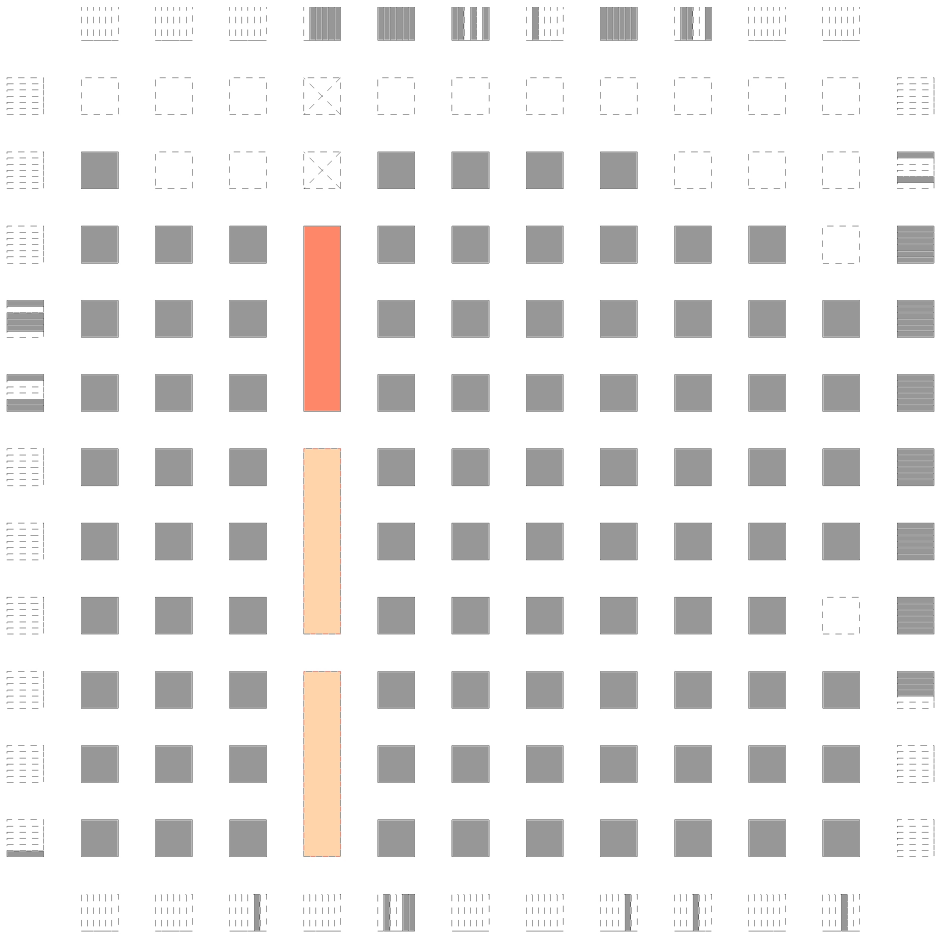

Figure 4.3: Oc_hdlc benchmark placement in HBVPR after level 2 Blackbox packing.

Benchmark	Estimated RAM bits each blackbox represents after Level 1 packing
oc_aes_core_inv	1×34,176
oc_ata_ocidec3	1×224
oc_hdlc	2×1,024
oc_minirisc	1×1,024
oc_oc8051	3×486, 1×972, 16×60, 1×678
os_blowfish	5×13,434 bits

Table 4.3: Estimated RAM bits each blackbox in each design has after Level 1 packing, the form is (No. of blackboxes× Ram bits)

Benchmark	No. of CLBs		Total no. of blackboxes pins	
	Initial	Extra	Initial	After multiplexing
oc_aes_core_inv	528	20	265	48
oc_ata_ocidec3	132	6	71	14
oc_hdlc	88	5	62	20
oc_minirisc	73	3	31	10
oc_oc8051	343	14	180	24
os_blowfish	565	37	362	67

Table 4.4: Pin multiplexing

As shown in table 4.2 and in figure 4.1 the number of blackbox and blackboxes I/O pins obtained from the .blif output from Quartus is unrealistic large and this is presenting a problem in Placement & Route step which will presented in the following section.

4.3 HBVPR results

The results presented in this section are output of HBVPR with the following options:

- The underlying FPGA architecture is STRATIX based, and the layout is set in auto, that is the minimum square array of clbs.
- Timing analysis of the routing.
- Path-timing-driven placement algorithm that focuses on minimizing both wire-length and the critical path delay.
- Detailed routing.
- Timing-driven router that focuses both on achieving a successful route and achieving good circuit speed.

The designs and different architectures are evaluated in the following three terms:

Area. Because there is no information about the size of the blackboxes only the minimum size of FPGA needed and the routing channel width[2] are used.

Delay. The Total logic delay, Total net delay, and critical path delay are used with the last, being the most crucial measurement.

Power. The power estimation results contain information about routing, logic block and clock: power, leakage power and energy.

The blackboxes in the benchmark designs have the following main features :

- They have height equal to one clb.
- The delay from any subblock input to the subblock output when this subblock is used in combinational mode is set to "2e-09" (sec).
- The delay from any subblock input pin to the FF storage element when this subblock is used in sequential mode is set to "1e-10" (sec)

[2]The minimum width required is increased by 20% in order to achieve lower delays

- The delay from the subblock storage element (FF) to the subblock output pin when this block is used in sequential mode is set to "1e-10" (sec)
- The number of tracks to which each block input pin connects in each channel bordering the pin is set to 0.25 of the channel width.
- The number of tracks to which each logic block output pin connects in each channel bordering the pin is set to "full" meaning it can use the whole channel width.
- Each type of blackbox can move (in placement) in one column, and the they have one empty column between them.

Benchmark	CLB Array size	Channel width
oc_aes_core_inv	-	-
oc_ata_ocidec3	64×64	36
oc_des_des3perf	-	-
oc_hdlc	32×32	22
oc_minirisc	16×16	28
oc_oc8051	-	-
os_blowfish	-	-
ucsb_152_tap_fir	-	-

Table 4.5: Area results benchmarks without blackbox-aware technology mapping.

Benchmark	Total logic delay	Total net delay	Critical path delay
oc_aes_core_inv	-	-	-
oc_ata_ocidec3	6.2733e-09	9.7107e-09	1.5984e-08
oc_hdlc	7.3548e-09	4.4894e-09	1.18442e-08
oc_minirisc	1.18706e-08	8.22e-09	2.00906e-08
oc_oc8051	-	-	-
os_blowfish	-	-	-

Table 4.6: Delay results (in seconds) of benchmarks without blackbox-aware technology mapping.

4.3.1 Non-packed

For comparisons reason (and in order to show the problems mentioned in section 3.5.1), the result presented in this section are of benchmarks without blackbox-aware technology mapping. The benchmarks which have "-" in the tables they couldn't run in HBVPR due to their enormous size ("Unable to malloc memory" error). The CMOS technology is 180nm.

In table 4.5 the area results are presented , the size of the minimum clb array in number of clbs and the required channel width.

In table 4.6 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.7 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.8 shows

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	-	-	-
oc_ata_ocidec3	1.11012	0.0120632	0.0686623
oc_hdlc	0.245012	0.0413755	0.0447689
oc_minirisc	0.0581201	0.00897621	0.00659914
oc_oc8051	-	-	-
os_blowfish	-	-	-

Table 4.7: Total power dissipation results (in Watts) of benchmarks without blackbox-aware technology mapping.

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	-	-
oc_ata_ocidec3	1.08555	0.00297538
oc_hdlc	0.151896	0.00214618
oc_minirisc	0.0374889	0.0015797
oc_oc8051	-	-
os_blowfish	-	-

Table 4.8: Leakage power dissipation results (in Watts) of benchmarks without blackbox-aware technology mapping.

the leakage power dissipation of the circuitry design and the table 4.9 shows the corresponding energy needed.

Lastly the results are summarized[3] in table 4.10.

As conclusion of this section we can say that there isn't an academic complete framework to support Heterogeneity from Synthesis step to Place & Route step of the design flow, for more complex designs. From six benchmark circuits, only the four smallest, HBVPR could process without memory overflow problem ("Cannot allocate memory").

[3]The average power and critical path delay and channel width are not accurate because the largest designs are missing

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	-	-	-
oc_ata_ocidec3	1.77441e-08	1.92819e-10	1.0975e-09
oc_hdlc	2.90198e-09	4.9006e-10	5.30252e-10
oc_minirisc	1.16767e-09	1.80338e-10	1.32581e-10
oc_oc8051	-	-	-
os_blowfish	-	-	-

Table 4.9: Energy consumption results (in Joules) of benchmarks without blackbox-aware technology mapping.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	-	-	-
oc_ata_ocidec3	1.19084	1.5984e-08	36
oc_hdlc	0.331157	1.18442e-08	22
oc_minirisc	0.0736955	2.00906e-08	28
oc_oc8051	-	-	-
os_blowfish	-	-	-
Average	**0.531**	**1.59e-08**	**28.6**

Table 4.10: Summarized results of benchmarks without blackbox-aware technology mapping.

4.3.2 Area and delay optimized architecture after level 1, 2 packing at 180nm

The intelligent FPGA Architecture Repository (iFAR) website [23] contains accurate area and timing estimates for the logic and routing of varied island-style FPGA architectures. Within this repository one can find areas and delays for architectures with varied logic block parameters, such as LUT size, and routing parameters such as segment length. The area and delay is determined through careful transistor sizing of each architecture. This is done for both a range of past, current and future implementation technologies (ranging from 22 nm to 180 nm CMOS) and a range of design objectives with varying emphasis on performance or area/cost.

Various FPGA architectures from iFAR were used for benchmarking the designs. These architectures used as templates in net2xml (see section 3.5.2.4) that created the specific for each design .xml architecture file. The parameters for all the architectures used were equally optimized for better area and delay results.

The results presented in this section are from a 180nm CMOS FPGA architecture.

All the benchmarks have passed through the blackbox-aware technology mapping step (Blackbox_Profiler tool). Every design had its blackboxes packed with level 1 and level 2 packing[4]. Also the pins of the blackboxes have been multiplexed.In the following results the benchmarks that their name is followed by "#FP" (Fully Packed) have been packed with level 2 blackbox packing and the rest with level 1.

In table 4.11 the area results are presented , the size of the minimum clb array in number of clbs, the required channel width and lastly the area savings in required area by packing the blackboxes, instead of leaving the as they are (section 4.3.1). This decrease is the percent decrease in the area of the FPGA array (in CLB area tiles).

In table 4.12 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.13 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.14 shows the leakage power dissipation of the circuitry design and the table 4.15 shows

[4]In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so these benchmarks they were tested with only level 1 packing

Benchmark	CLB Array size	Channel width	Decrease of Area
oc_aes_core_inv	24×24	46	-
oc_ata_ocidec3	13×13	40	95%
oc_hdlc	11×11	32	88%
oc_minirisc	10×10	36	60%
oc_oc8051	42×42	44	-
os_blowfish	28×28	50	-
oc_hdlc#FP	11×11	28	88%
oc_oc8051#FP	20×20	62	-
os_blowfish#FP	25×25	56	-

Table 4.11: Area results from benchmarks in 180nm optimized for area and delay FPGA architecture.

Benchmark	Total logic delay	Total net delay	Critical path delay
oc_aes_core_inv	5.027e-09	8.0692e-09	1.30962e-08
oc_ata_ocidec3	3.1175e-09	5.1387e-09	8.2562e-09
oc_hdlc	5.1827e-09	7.223e-10	5.905e-09
oc_minirisc	7.7444e-09	4.2147e-09	1.19591e-08
oc_oc8051	1.63509e-08	1.17537e-08	2.81046e-08
os_blowfish	1.89591e-08	1.9748e-09	2.09339e-08
oc_hdlc#FP	2.3278e-09	3.9957e-09	6.3235e-09
oc_oc8051#FP	1.20431e-08	1.64813e-08	2.85244e-08
os_blowfish#FP	1.89591e-08	1.7462e-09	2.07053e-08

Table 4.12: Delay results from benchmarks in 180nm optimized for area and delay FPGA architecture.

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	0.449013	0.213825	0.0202744
oc_ata_ocidec3	0.056261	0.0218882	0.0080378
oc_hdlc	0.10228	0.0832783	0.00572775
oc_minirisc	0.0362397	0.0146424	0.00282817
oc_oc8051	0.505805	0.0208085	0.0192158
os_blowfish	0.26368	0.0435005	0.012905
oc_hdlc#FP	0.100096	0.0780017	0.00534868
oc_oc8051#FP	0.134548	0.0207192	0.00474794
os_blowfish#FP	0.240735	0.0424196	0.0128237

Table 4.13: Total power dissipation results from benchmarks in 180nm optimized for area and delay FPGA architecture.

the corresponding energy needed.

The results are summarized in table 4.16.

In figure 4.4 an area comparison it is shown between benchmarks that haven't passed through blackbox profiling and packing are compared with those packed with packing level 1 and 2 at the same 180nm technology. The blue part of each bar is the percent area savings in terms of CLBs number of the area required by each non-packed design compared to the corresponding packed design. The FPGA array size (in CLB tiles) is decreased by a factor of 80%. If larger designs had completed the P&R step this factor would be even greater because the "minimum" width of the FPGA array is $Number - of - blackboxes + 1$ since each blackbox utilizes a column.

As shown also in figure 4.1 and in table 4.2 the hierarchical blif netlist, the only way until today solution to support heterogeneity, outputted from Quartus in Synthesis step doesn't have realistic results. This is mainly because BLIF format describes application's functionality at finer level. So for example, memories are reported at word level, each word is reflected in one blackbox. Something like this isn't realistic because commercial FPGAs have a small number of independent RAM blocks. So blackbox-aware technology mapping proposed in this thesis through Heterogeneity Support Toolset (section 3.5) is necessary for systematic and efficient grouping of blackboxes.

Every design is mapped in a certain amount of CLBs as resulted from

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	0.120238	0.0131264
oc_ata_ocidec3	0.0309331	0.00340505
oc_hdlc	0.0209909	0.0026209
oc_minirisc	0.0184509	0.00190929
oc_oc8051	0.468334	0.00839515
os_blowfish	0.206896	0.0154454
oc_hdlc#FP	0.0158092	0.00262174
oc_oc8051#FP	0.109309	0.00839512
os_blowfish#FP	0.172131	0.0141808

Table 4.14: Leakage power dissipation results from benchmarks in 180nm optimized for area and delay FPGA architecture.

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	5.88036e-09	2.80029e-09	2.65518e-10
oc_ata_ocidec3	4.64502e-10	1.80713e-10	6.63617e-11
oc_hdlc	6.03963e-10	4.91758e-10	3.38224e-11
oc_minirisc	4.33394e-10	1.7511e-10	3.38224e-11
oc_oc8051	1.42154e-08	5.84815e-10	5.40053e-10
os_blowfish	5.51984e-09	9.10636e-10	2.70151e-10
oc_hdlc#FP	6.32958e-10	4.93244e-10	3.38224e-11
oc_oc8051#FP	3.83789e-09	5.91004e-10	1.35432e-10
os_blowfish#FP	4.9845e-09	8.7831e-10	2.65518e-10

Table 4.15: Energy consumption from benchmarks in 180nm optimized for area and delay FPGA architecture.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	0.683112	1.30962e-08	46
oc_ata_ocidec3	0.086187	8.2562e-09	40
oc_hdlc	0.191286	5.905e-09	32
oc_minirisc	0.0537103	1.19591e-08	36
oc_oc8051	0.545829	2.81046e-08	44
os_blowfish	0.320085	2.09339e-08	50
oc_hdlc#FP	0.183446	6.3235e-09	28
oc_oc8051#FP	0.160015	2.85244e-08	62
os_blowfish#FP	0.295979	2.07053e-08	56
Average	**0.2799**	**1.597e-08**	**43.7**

Table 4.16: Summarized results from benchmarks in 180nm optimized for power FPGA architecture.

HBT-Vpack independently from FPGA size and blackbox-aware technology mapping. This means that the area penalty of unpacked blackboxes its utilized by logic elements that are not used by the benchmark design. Also the extra area causes the intercluster signals to travel greater lengths and pass through a larger number of switchboxes.

The increase in required wires and switchboxes that the intercluster signal have to travel through results in an increase of delay of each benchmark as shown in figure 4.5.

The increase of unused CLBs, wires and switchboxes, (due to the area penalty) results in larger leakage power and total routing power dissipation correspondingly so benchmarks that haven't passed through blackbox profiling and packing are more "power-hungry" than those packed with packing level 1 and 2. This is shown in figure 4.6.

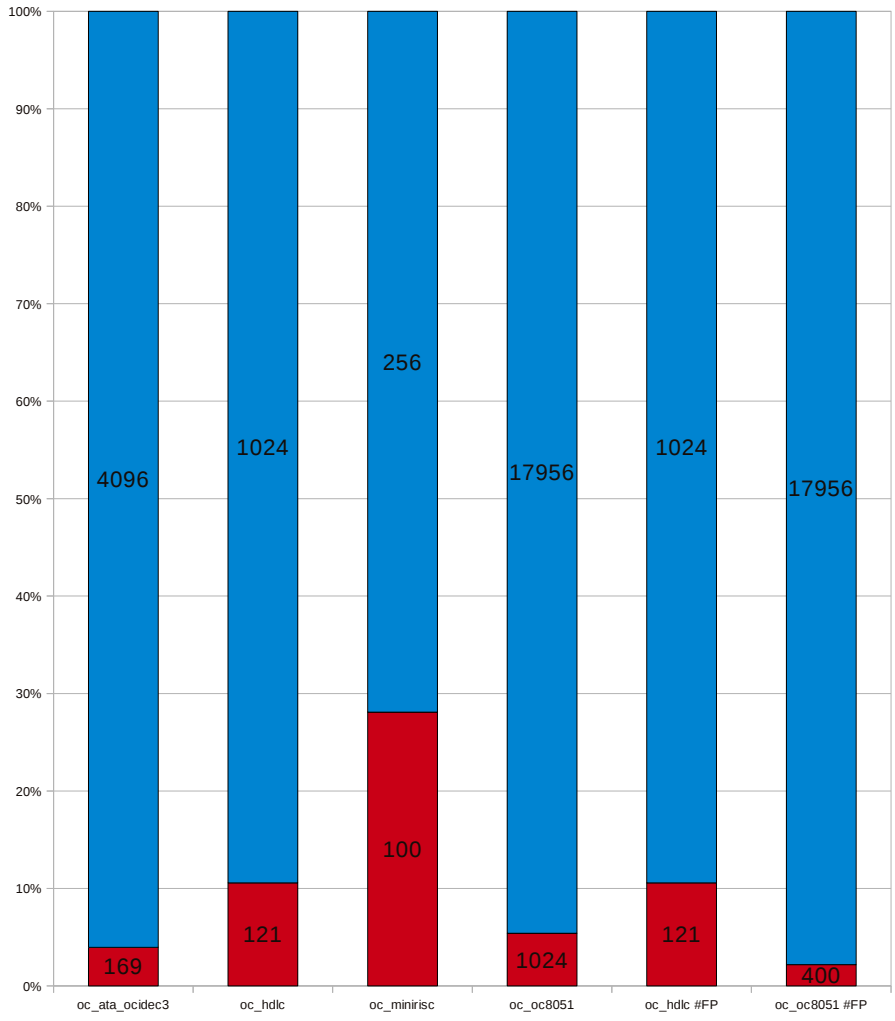

Figure 4.4: Comparison between non-packed and packed designs in terms of area requirement. The blue area of each bar represents the percent area savings of packed designs. The total area required by each design is shown in CLBs number, the number in the red area is for packed designs and the blue for non-packed

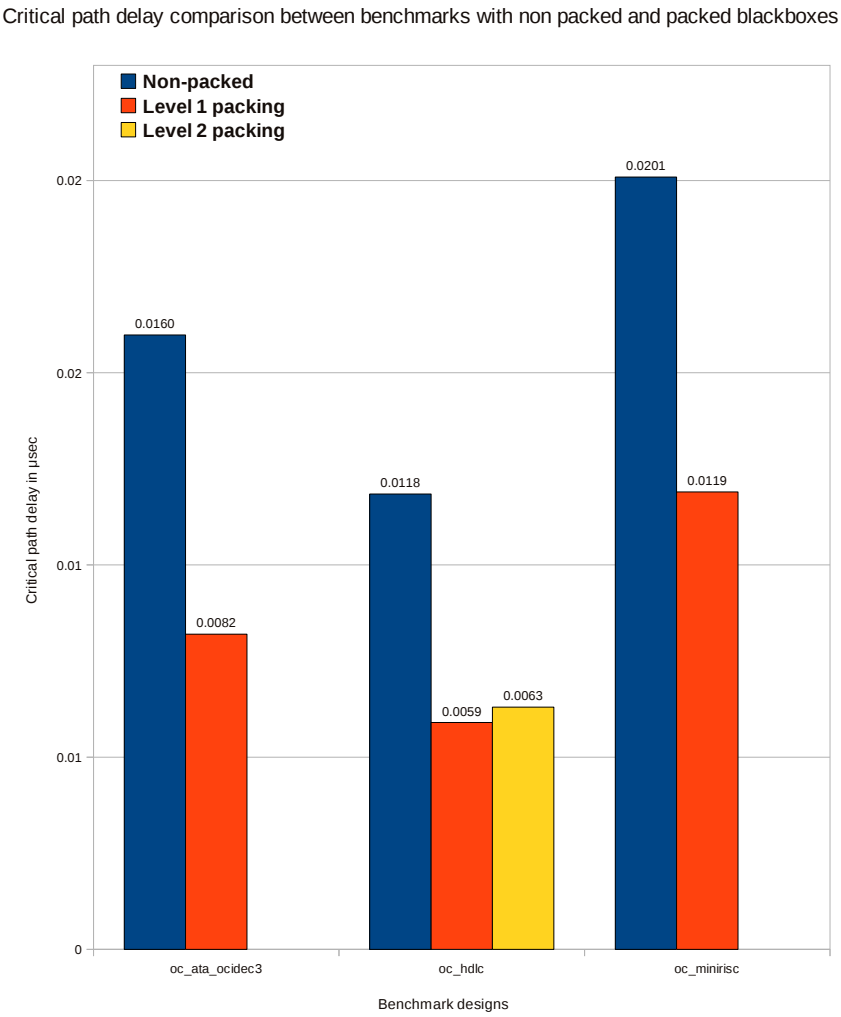

Figure 4.5: Comparison between non-packed and packed designs in terms of critical path delay.

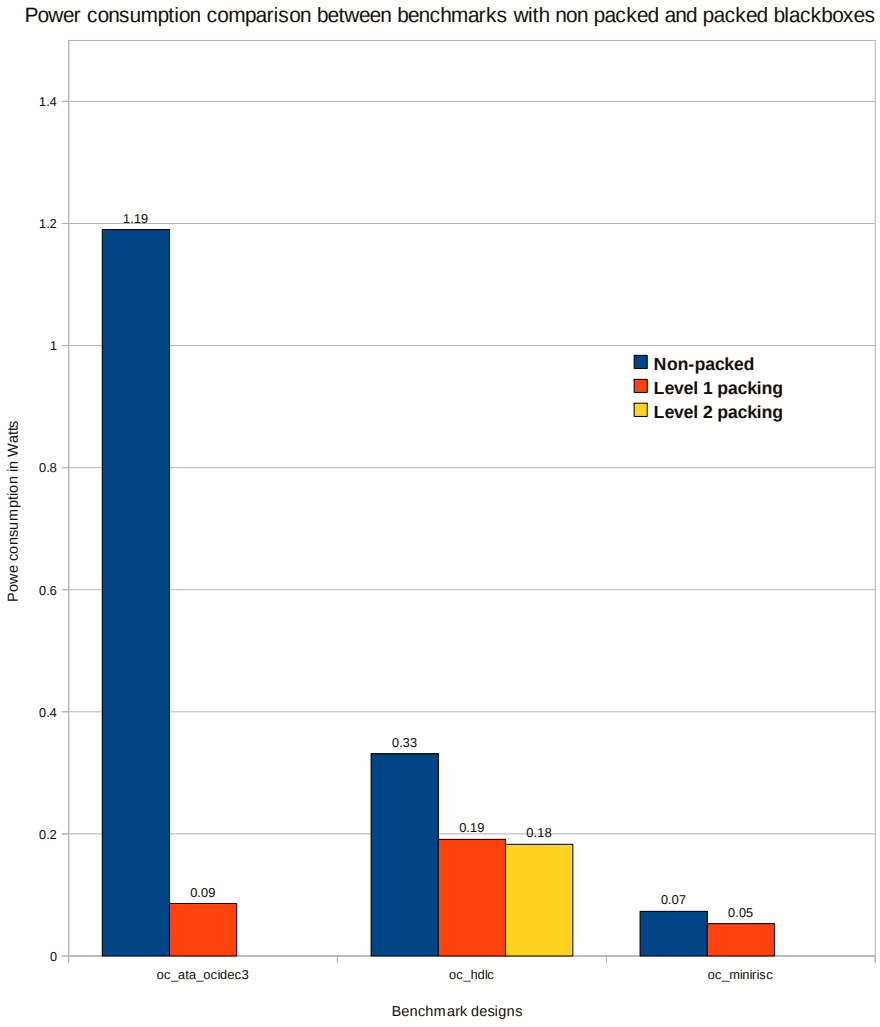

Figure 4.6: Comparison between non-packed and packed designs in terms of total power consumption.

Benchmark	CLB Array size	Channel width
oc_aes_core_inv	24×24	44
oc_ata_ocidec3	13×13	44
oc_hdlc	11×11	32
oc_minirisc	10×10	36
oc_oc8051	42×42	44
os_blowfish	28×28	56
oc_hdlc#FP	11×11	28
oc_oc8051#FP	20×20	60
os_blowfish#FP	25×25	58

Table 4.17: Area results from benchmarks in 130nm FPGA architecture.

4.3.3 Area and delay optimized architecture after level 1, 2 packing at 130nm

All the benchmarks in this section have passed through the blackbox-aware technology mapping step (Blackbox_Profiler tool). Every design had its blackboxes packed with level 1 and level 2 packing[5]. Also the pins of the blackboxes have been multiplexed.In the following results the benchmarks that their name is followed by "#FP" (Fully Packed) have been packed with level 2 blackbox packing and the rest with level 1.

The results presented in this section are from a 130nm CMOS FPGA architecture obtained from iFAR.

In table 4.17 the area results are presented, the size of the minimum clb array in number of clbs and the required channel width.

In table 4.18 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.19 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.20 shows the leakage power dissipation of the circuitry design and the table 4.21 shows the corresponding energy needed.

Lastly the results are summarized in table 4.22.

[5]In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so these benchmarks they were tested with only level 1 packing

Benchmark	Total logic delay	Total net delay	Critical path delay
oc_aes_core_inv	3.51636e-09	4.75289e-09	8.26925e-09
oc_ata_ocidec3	5.0558e-09	3.2083e-10	5.37663e-09
oc_hdlc	5.0558e-09	5.4574e-10	5.60154e-09
oc_minirisc	5.91904e-09	1.30427e-09	7.22331e-09
oc_oc8051	1.04055e-08	8.34816e-09	1.87537e-08
os_blowfish	1.20939e-08	1.35829e-09	1.34522e-08
oc_hdlc#FP	5.0558e-09	3.2083e-10	5.37663e-09
oc_oc8051#FP	1.07453e-08	5.62829e-09	1.63736e-08
os_blowfish#FP	1.20939e-08	1.13338e-09	1.32273e-08

Table 4.18: Delay results from benchmarks in 130nm FPGA architecture.

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	0.287672	0.174706	0.0167482
oc_ata_ocidec3	0.0368797	0.0167423	0.00643797
oc_hdlc	0.0465192	0.0458233	0.00314947
oc_minirisc	0.0232786	0.0119692	0.00244236
oc_oc8051	0.28407	0.0157366	0.0150207
os_blowfish	0.17215	0.0336424	0.0104751
oc_hdlc#FP	0.053521	0.048991	0.00328122
oc_oc8051#FP	0.0750406	0.0172036	0.00431439
os_blowfish#FP	0.147747	0.0325155	0.0104705

Table 4.19: Total power dissipation results from benchmarks in 130nm FPGA architecture.

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	0.0605833	0.00935204
oc_ata_ocidec3	0.0206187	0.00245484
oc_hdlc	0.0114238	0.00188863
oc_minirisc	0.0102589	0.00137818
oc_oc8051	0.256207	0.00601822
os_blowfish	0.13291	0.0110108
oc_hdlc#FP	0.00843055	0.00188857
oc_oc8051#FP	0.0583797	0.00601898
os_blowfish#FP	0.10084	0.0101076

Table 4.20: Leakage power dissipation results from benchmarks in 130nm FPGA architecture.

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	2.37883e-09	1.44468e-09	1.38495e-10
oc_ata_ocidec3	1.98288e-10	9.00171e-11	3.46146e-11
oc_hdlc	2.60579e-10	2.56681e-10	1.76419e-11
oc_minirisc	1.68149e-10	8.64576e-11	1.76419e-11
oc_oc8051	5.32735e-09	2.95119e-10	2.81694e-10
os_blowfish	2.31579e-09	4.52564e-10	1.40912e-10
oc_hdlc#FP	2.87763e-10	2.63407e-10	1.76419e-11
oc_oc8051#FP	1.22868e-09	2.81684e-10	7.0642e-11
os_blowfish#FP	1.95429e-09	4.30091e-10	1.38495e-10

Table 4.21: Energy consumption (in Joules) from benchmarks in 130nm FPGA architecture.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	0.479126	8.26925e-09	44
oc_ata_ocidec3	0.06006	5.37663e-09	44
oc_hdlc	0.095492	5.60154e-09	32
oc_minirisc	0.0376902	7.22331e-09	36
oc_oc8051	0.314827	1.87537e-08	44
os_blowfish	0.216268	1.34522e-08	56
oc_hdlc#FP	0.105793	5.37663e-09	28
oc_oc8051#FP	0.0965586	1.63736e-08	60
os_blowfish#FP	0.190733	1.32273e-08	58
Average	**0.176**	**1.040e-08**	**44.6**

Table 4.22: Summarized results from benchmarks in 130nm FPGA architecture.

4.3.4 Area and delay optimized architecture after level 1, 2 packing at 90nm

All the benchmarks in this section have passed through the blackbox-aware technology mapping step (Blackbox_Profiler tool). Every design had its blackboxes packed with level 1 and level 2 packing[6]. Also the pins of the blackboxes have been multiplexed.In the following results the benchmarks that their name is followed by "#FP" (Fully Packed) have been packed with level 2 blackbox packing and the rest with level 1.

The results presented in this section are from a 90nm CMOS FPGA architecture obtained from iFAR.

In table 4.23 the area results are presented, the size of the minimum clb array in number of clbs and the required channel width.

In table 4.24 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.25 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.26 shows the leakage power dissipation of the circuitry design and the table 4.27 shows the corresponding energy needed.

[6]In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so these benchmarks they were tested with only level 1 packing

oc_aes_core_inv	24×24	44
oc_ata_ocidec3	13×13	44
oc_hdlc	11×11	28
oc_minirisc	10×10	34
oc_oc8051	42×42	44
os_blowfish	28×28	50
oc_hdlc#FP	11×11	28
oc_oc8051#FP	20×20	62
os_blowfish#FP	25×25	58

Table 4.23: Area results from benchmarks in 90nm FPGA architecture.

Benchmark	**Total logic delay**	**Total net delay**	**Critical path delay**
oc_aes_core_inv	3.34686e-09	3.47617e-09	6.82303e-09
oc_ata_ocidec3	5.0442e-09	2.7317e-10	5.31737e-09
oc_hdlc	5.0442e-09	5.2657e-10	5.57077e-09
oc_minirisc	4.24383e-09	2.07843e-09	6.32226e-09
oc_oc8051	7.49437e-09	8.70637e-09	1.62007e-08
os_blowfish	1.04119e-08	1.09268e-09	1.15045e-08
oc_hdlc#FP	5.0442e-09	3.3652e-10	5.38072e-09
oc_oc8051#FP	9.56174e-09	4.73509e-09	1.42968e-08
os_blowfish#FP	1.04119e-08	1.02933e-09	1.14412e-08

Table 4.24: Delay results from benchmarks in 90nm FPGA architecture.

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	0.356626	0.179939	0.0172955
oc_ata_ocidec3	0.0421214	0.0150135	0.00554674
oc_hdlc	0.0537712	0.0397783	0.0026984
oc_minirisc	0.0263515	0.0117195	0.00237766
oc_oc8051	0.341419	0.015177	0.0148156
os_blowfish	0.185337	0.0336089	0.0104365
oc_hdlc#FP	0.0551673	0.0414802	0.00279371
oc_oc8051#FP	0.095756	0.0164531	0.00421016
os_blowfish#FP	0.176735	0.0318081	0.0103143

Table 4.25: Total power dissipation results from benchmarks in 90nm FPGA architecture.

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	0.07357	0.00867012
oc_ata_ocidec3	0.0248991	0.00226197
oc_hdlc	0.0105774	0.00174699
oc_minirisc	0.0110838	0.00127344
oc_oc8051	0.307173	0.00553185
os_blowfish	0.136839	0.010284
oc_hdlc#FP	0.0104559	0.00174598
oc_oc8051#FP	0.0750971	0.00553036
os_blowfish#FP	0.12051	0.00940561

Table 4.26: Leakage power dissipation results from benchmarks in 90nm FPGA architecture.

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	2.43327e-09	1.22773e-09	1.18008e-10
oc_ata_ocidec3	2.23975e-10	7.98321e-11	2.94941e-11
oc_hdlc	2.99547e-10	2.21596e-10	1.50322e-11
oc_minirisc	1.66601e-10	7.4094e-11	1.50322e-11
oc_oc8051	5.53124e-09	2.45879e-10	2.40024e-10
os_blowfish	2.13222e-09	3.86655e-10	1.20067e-10
oc_hdlc#FP	2.9684e-10	2.23193e-10	1.50322e-11
oc_oc8051#FP	1.36901e-09	2.35227e-10	6.0192e-11
os_blowfish#FP	2.02206e-09	3.63922e-10	1.18008e-10

Table 4.27: Energy consumption (in Joules) from benchmarks in 90nm FPGA architecture.

Lastly the results are summarized in table 4.28.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	0.55386	6.82303e-09	44
oc_ata_ocidec3	0.0626816	5.31737e-09	44
oc_hdlc	0.0962479	5.57077e-09	28
oc_minirisc	0.0404487	6.32226e-09	34
oc_oc8051	0.371411	1.62007e-08	44
os_blowfish	0.229382	1.15045e-08	50
oc_hdlc#FP	0.0994412	5.38072e-09	28
oc_oc8051#FP	0.116419	1.42968e-08	62
os_blowfish#FP	0.218858	1.14412e-08	58
Average	**0.196**	**9.20e-09**	**43.5**

Table 4.28: Summarized results from benchmarks in 90nm FPGA architecture.

4.3.5 Area and delay optimized architecture after level 1, 2 packing at 65nm

All the benchmarks in this section have passed through the blackbox-aware technology mapping step (Blackbox_Profiler tool). Every design had its blackboxes packed with level 1 and level 2 packing[7]. Also the pins of the blackboxes have been multiplexed.In the following results the benchmarks that their name is followed by "#FP" (Fully Packed) have been packed with level 2 blackbox packing and the rest with level 1.

The results presented in this section are from a 65nm CMOS FPGA architecture obtained from iFAR.

In table 4.29 the area results are presented, the size of the minimum clb array in number of clbs and the required channel width.

In table 4.30 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.31 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.32 shows the leakage power dissipation of the circuitry design and the table 4.33 shows the corresponding energy needed.

[7]In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so these benchmarks they were tested with only level 1 packing

Benchmark	CLB Array size	Channel width
oc_aes_core_inv	24×24	46
oc_ata_ocidec3	13×13	38
oc_hdlc	11×11	26
oc_minirisc	10×10	36
oc_oc8051	42×42	46
os_blowfish	28×28	50
oc_hdlc#FP	11×11	32
oc_oc8051#FP	20×20	62
os_blowfish#FP	25×25	56

Table 4.29: Area results from benchmarks in 65nm FPGA architecture.

Benchmark	Total logic delay	Total net delay	Critical path delay
oc_aes_core_inv	2.7714e-09	4.09254e-09	6.86394e-09
oc_ata_ocidec3	4.99396e-09	2.2785e-10	5.22181e-09
oc_hdlc	4.99396e-09	2.8129e-10	5.27525e-09
oc_minirisc	3.52677e-09	1.78345e-09	5.31022e-09
oc_oc8051	5.74936e-09	7.78282e-09	1.35322e-08
os_blowfish	8.67292e-09	9.114e-10	9.58432e-09
oc_hdlc#FP	4.99396e-09	2.2785e-10	5.22181e-09
oc_oc8051#FP	7.72561e-09	4.26987e-09	1.19955e-08
os_blowfish#FP	8.67292e-09	7.5108e-10	9.424e-09

Table 4.30: Delay results from benchmarks in 65nm FPGA architecture.

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	0.25168	0.149822	0.0144464
oc_ata_ocidec3	0.0262287	0.012889	0.0047461
oc_hdlc	0.0371527	0.0349259	0.00239442
oc_minirisc	0.0214984	0.0117852	0.00237866
oc_oc8051	0.251889	0.0149079	0.0149042
os_blowfish	0.1384	0.0321715	0.0105265
oc_hdlc#FP	0.0376617	0.0353386	0.00241893
oc_oc8051#FP	0.0707263	0.0158939	0.00421642
os_blowfish#FP	0.125898	0.0313164	0.010522

Table 4.31: Total power dissipation results from benchmarks in 65nm FPGA architecture.

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	0.0555407	0.0079329
oc_ata_ocidec3	0.013454	0.00206927
oc_hdlc	0.00650337	0.00159811
oc_minirisc	0.00854591	0.00116734
oc_oc8051	0.226265	0.0050822
os_blowfish	0.0938053	0.00934561
oc_hdlc#FP	0.00978357	0.00159794
oc_oc8051#FP	0.0517859	0.00508074
os_blowfish#FP	0.0789864	0.0085727

Table 4.32: Leakage power dissipation results from benchmarks in 65nm FPGA architecture.

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	1.72752e-09	1.02837e-09	9.91595e-11
oc_ata_ocidec3	1.36962e-10	6.73041e-11	2.47832e-11
oc_hdlc	1.9599e-10	1.84243e-10	1.26312e-11
oc_minirisc	1.14161e-10	6.25822e-11	1.26312e-11
oc_oc8051	3.4086e-09	2.01736e-10	2.01686e-10
os_blowfish	1.32647e-09	3.08342e-10	1.0089e-10
oc_hdlc#FP	1.96662e-10	1.84531e-10	1.26312e-11
oc_oc8051#FP	8.48395e-10	1.90654e-10	5.0578e-11
os_blowfish#FP	1.18646e-09	2.95126e-10	9.91595e-11

Table 4.33: Energy consumption (in Joules) from benchmarks in 65nm FPGA architecture.

Lastly the results are summarized in table 4.34.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	0.415949	6.86394e-09	46
oc_ata_ocidec3	0.0438639	5.22181e-09	38
oc_hdlc	0.0744731	5.27525e-09	26
oc_minirisc	0.0356622	5.31022e-09	36
oc_oc8051	0.281701	1.35322e-08	46
os_blowfish	0.181098	9.58432e-09	50
oc_hdlc#FP	0.0754192	5.22181e-09	32
oc_oc8051#FP	0.0908365	1.19955e-08	62
os_blowfish#FP	0.167736	9.424e-09	56
Average	**0.151**	**8.047e-09**	**43.5**

Table 4.34: Summarized results from benchmarks in 65nm FPGA architecture.

4.3.6 Area and delay optimized architecture after level 1, 2 packing at 45nm

All the benchmarks in this section have passed through the blackbox-aware technology mapping step (Blackbox_Profiler tool). Every design had its blackboxes packed with level 1 and level 2 packing[8]. Also the pins of the blackboxes have been multiplexed.In the following results the benchmarks that their name is followed by "#FP" (Fully Packed) have been packed with level 2 blackbox packing and the rest with level 1.

The results presented in this section are from a 45nm CMOS FPGA architecture obtained from iFAR.

In table 4.35 the area results are presented, the size of the minimum clb array in number of clbs and the required channel width.

In table 4.36 the delay results are presented : Total logic delay, Total net delay, and critical path in seconds.

Table 4.37 shows the total power dissipation of the circuitry design analysed in routing, logic block and clock power consumption, table 4.38 shows the leakage power dissipation of the circuitry design and the table 4.39 shows the corresponding energy needed.

[8]In some benchmarks the Blackbox_profiler resulted that all blackboxes are to be packed into one in the first level of packing, so these benchmarks they were tested with only level 1 packing

Benchmark	CLB Array size	Channel width
oc_aes_core_inv	24×24	44
oc_ata_ocidec3	13×13	40
oc_hdlc	11×11	28
oc_minirisc	10×10	32
oc_oc8051	42×42	46
os_blowfish	28×28	60
oc_hdlc#FP	11×11	26
oc_oc8051#FP	20×20	62
os_blowfish#FP	25×25	56

Table 4.35: Area results from benchmarks in 45nm FPGA architecture.

Benchmark	Total logic delay	Total net delay	Critical path delay
oc_aes_core_inv	4.34941e-09	1.72204e-09	6.07145e-09
oc_ata_ocidec3	4.98534e-09	2.7005e-10	5.25539e-09
oc_hdlc	4.98534e-09	3.9631e-10	5.38165e-09
oc_minirisc	3.38714e-09	3.89298e-09	7.28012e-09
oc_oc8051	5.72687e-09	8.79604e-09	1.45229e-08
os_blowfish	9.05448e-09	1.0802e-09	1.01347e-08
oc_hdlc#FP	4.98534e-09	3.9631e-10	5.38165e-09
oc_oc8051#FP	8.04476e-09	4.73819e-09	1.27829e-08
os_blowfish#FP	9.05448e-09	1.14333e-09	1.01978e-08

Table 4.36: Delay results from benchmarks in 45nm FPGA architecture.

Benchmark	Total Power (W)		
	Routing	Logic Block	Clock
oc_aes_core_inv	0.229306	0.113608	0.0109331
oc_ata_ocidec3	0.0258275	0.00864125	0.00315684
oc_hdlc	0.0328686	0.0232201	0.00157119
oc_minirisc	0.0150966	0.00616325	0.00116146
oc_oc8051	0.268165	0.010198	0.00929657
os_blowfish	0.165753	0.0221397	0.00666403
oc_hdlc#FP	0.0299404	0.0236438	0.00157119
oc_oc8051#FP	0.0700368	0.0110367	0.00264868
os_blowfish#FP	0.120167	0.0210552	0.00650919

Table 4.37: Total power dissipation results (in Watts) from benchmarks in 45nm FPGA architecture.

Benchmark	Leakage Power (W)	
	Routing	Logic Block
oc_aes_core_inv	0.0569171	0.00656244
oc_ata_ocidec3	0.0155596	0.0017025
oc_hdlc	0.00797102	0.00131098
oc_minirisc	0.00750365	0.000954651
oc_oc8051	0.247363	0.00419752
os_blowfish	0.132378	0.00772435
oc_hdlc#FP	0.00717569	0.00131029
oc_oc8051#FP	0.0566593	0.00419944
os_blowfish#FP	0.083589	0.00709208

Table 4.38: Leakge power dissipation results (in Watts) from benchmarks in 45nm FPGA architecture.

Benchmark	Energy in Joule		
	Routing	Logic Block	Clock
oc_aes_core_inv	1.39222e-09	6.89765e-10	6.63795e-11
oc_ata_ocidec3	1.35733e-10	4.54131e-11	1.65904e-11
oc_hdlc	1.76887e-10	1.24962e-10	8.45559e-12
oc_minirisc	1.09905e-10	4.48692e-11	8.45559e-12
oc_oc8051	3.89453e-09	1.48105e-10	1.35013e-10
os_blowfish	1.67986e-09	2.24379e-10	6.75378e-11
oc_hdlc#FP	1.61129e-10	1.27243e-10	8.45559e-12
oc_oc8051#FP	8.95276e-10	1.41081e-10	3.3858e-11
os_blowfish#FP	1.22545e-09	2.14717e-10	6.63795e-11

Table 4.39: Energy consumption (in Joules) from benchmarks in 45nm FPGA architecture.

Lastly the results are summarized in table 4.40.

Benchmark	Total Power (W)	Critical path delay (s)	Channel width
oc_aes_core_inv	0.353847	6.07145e-09	44
oc_ata_ocidec3	0.0376256	5.25539e-09	40
oc_hdlc	0.0576599	5.38165e-09	28
oc_minirisc	0.0224213	7.28012e-09	32
oc_oc8051	0.287659	1.45229e-08	46
os_blowfish	0.194557	1.01347e-08	60
oc_hdlc#FP	0.0551554	5.38165e-09	26
oc_oc8051#FP	0.0837221	1.27829e-08	62
os_blowfish#FP	0.147732	1.01978e-08	56
Average	**0.137**	**8.556e-09**	**43.7**

Table 4.40: Summarized results from benchmarks in 45nm FPGA architecture.

4.3.7 Comparison results

The following figures present the comparison between the different architectures. In figure 4.7 the average critical path delay of each of the previous FPGA architectures is presented. Figure 4.8 shows the average power consumption of each FPGA architecture. Because of the different critical path delay of each architecture its difficult to immediately compare the power dissipation of each FPGA architecture. Figure 4.9 shows the average power consumption of each architecture scaled to the same critical path delay, that of the area and delay optimized architecture at 180nm. The scaling is based to the fact that power is inverse proportional to the time element of a design.

Delay

As shown in figure 4.7, generally the decrease in CMOS size results in decrease in delay of the circuitry. This is a result of the decrease in delay of:

- A signal to go from a routing track to a logic block input.
- A signal to pass through a switchbox
- A logic block to process input signals
- A signal to go from the output of a logic block to the routing track.

The FPGA architecture at 45nm has critical path delay 10% increased compared to that of 65nm. This is natural because the same benchmark in different architectures may have slightly different placement, routing, FPGA array size and channel width. The cause of this variation is that Placement and Routing are timing-driven.

Power

As shown in figure 4.8, in most cases the decrease in technology scaling results to a decrease of the average total power consumption of the circuitry, mainly because of the decrease of supply Voltage (V_{dd}). Power consumption is proportional to the square of supply Voltage, so little decreases in V_{dd} result in large power benefits. On the other hand when CMOS technology is decreasing in size the delay of the circuit which is inverse proportional to power consumption is also decreased. Figure 4.9 shows that if we extract from the parameters the delay of the design the scaling of CMOS size and supply voltage is reflected clearly to the average power consumption of each architecture.

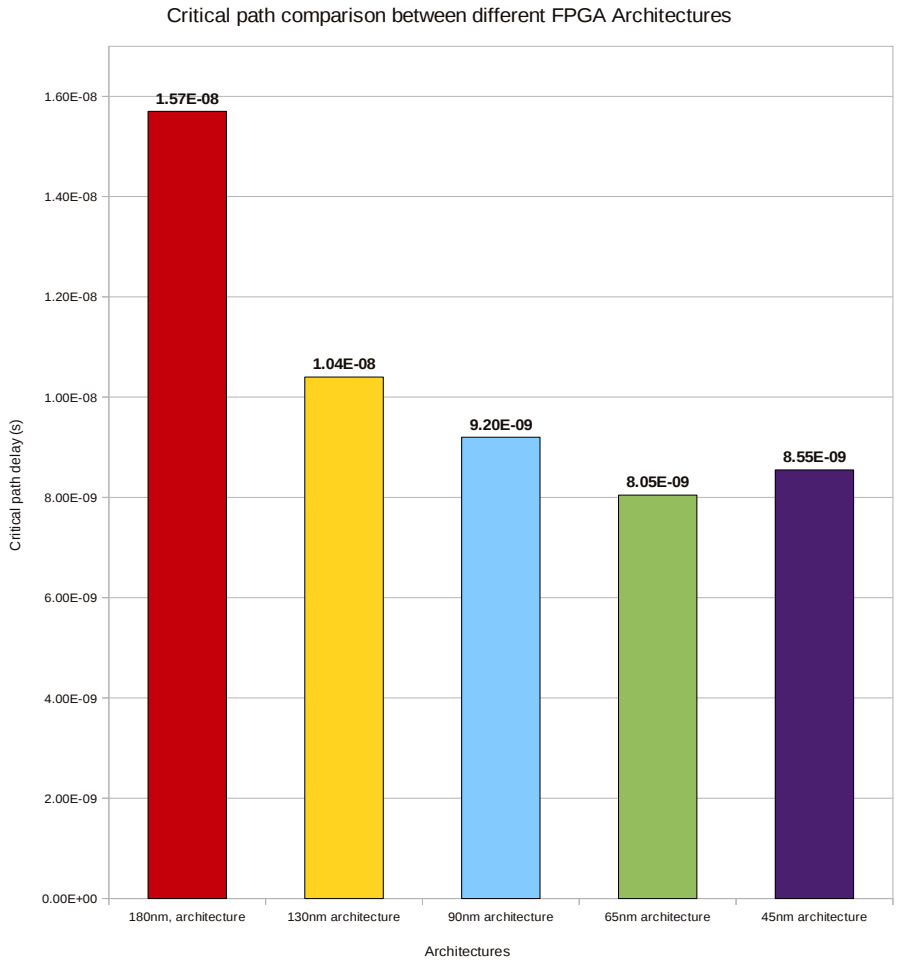

Figure 4.7: Comparison of different technologies in terms of critical path delay.

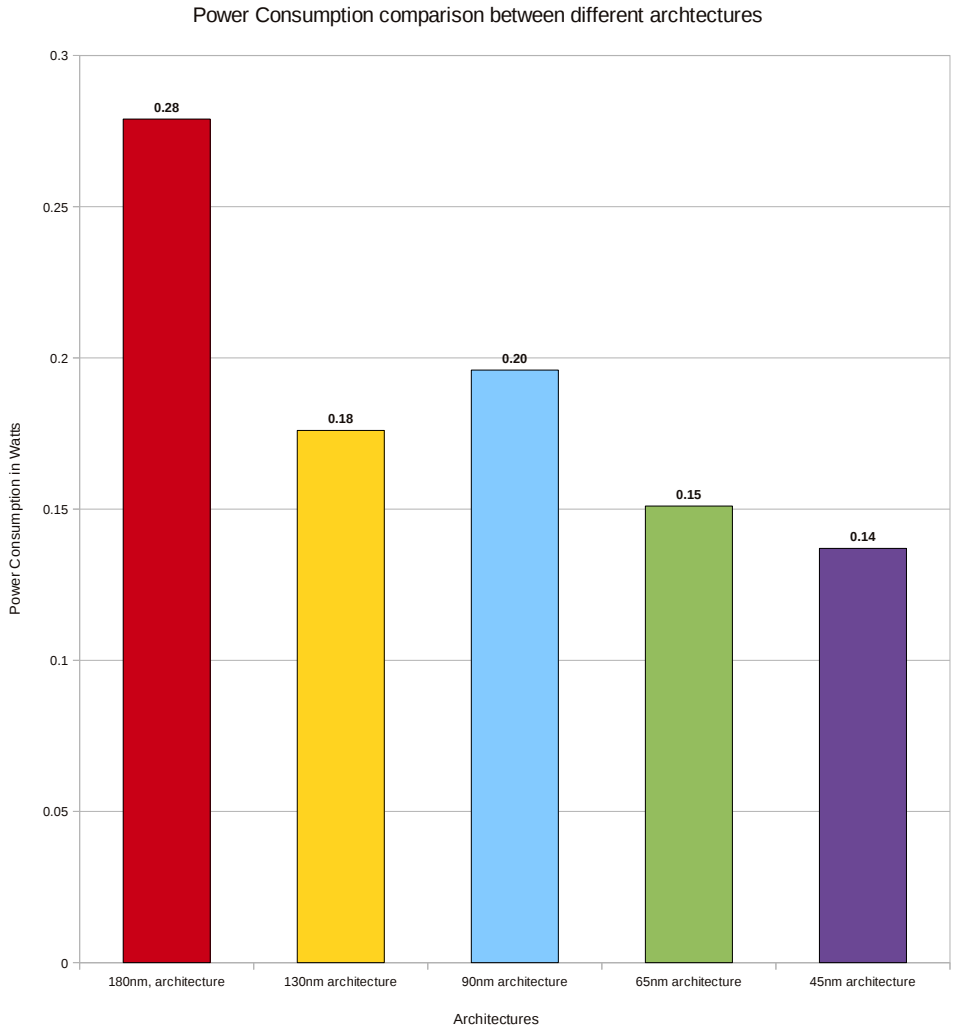

Figure 4.8: Comparison of different technologies in terms of power consumption.

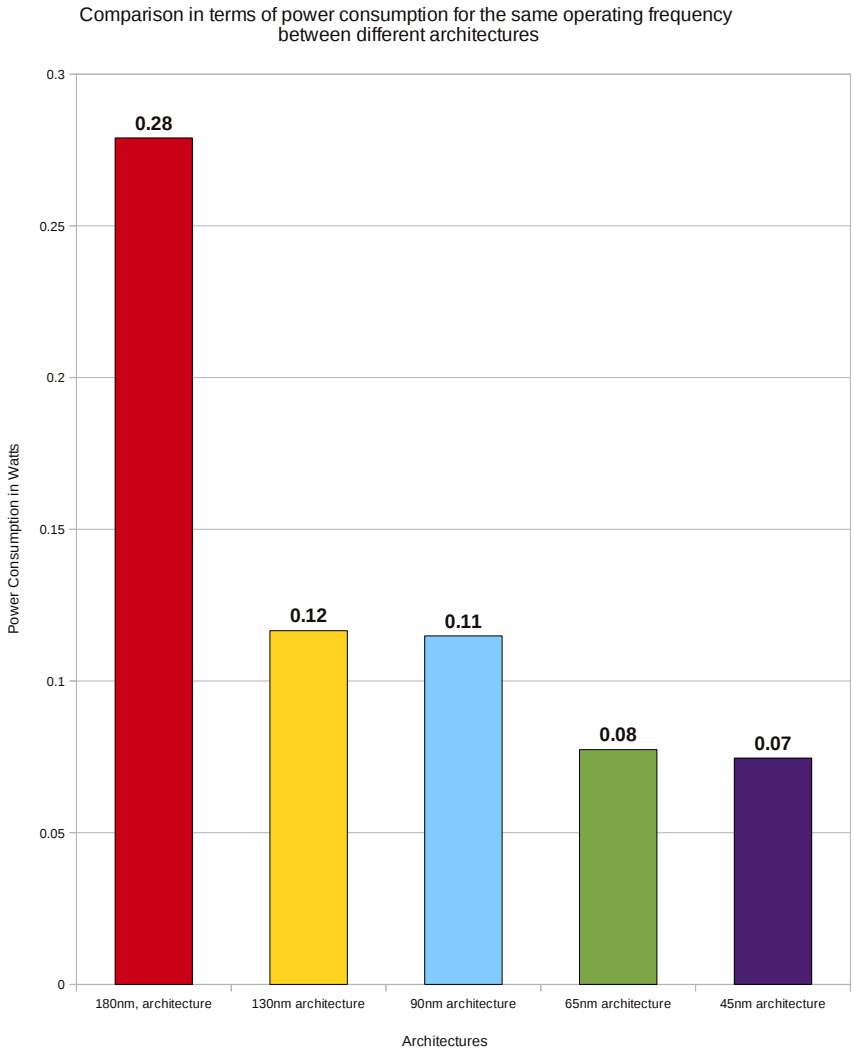

Figure 4.9: Comparison in terms of power consumption for the same operating frequency.

Chapter 5

Conclusions

Reconfigurable systems and special FPGAs have received much attention during the last years. The research is expanded in many fields of interest, FPGA Fabric, power consumption, heterogeneity, CAD tools etc. This thesis addresses the lack of a complete, free, academic design framework of CAD tools that supports power estimation in Heterogeneous FPGAs.

In the first chapter, the fundamental principles of an FPGA are analysed. In the second chapter complete frameworks and standalone tools of FPGA design both academic and commercial are analysed, and the lack of tools for power estimation in Heterogeneous FPGAs is shown.

The third chapter presents the proposed framework NAROUTO. The practical problems occurred in development and every step in the design flow from VHDL circuit description to FPGA programming is thoroughly described. New tools, a Heterogeneity Support Toolset and modified existing tools are presented.

The framework is tested in chapter four , in which the results are presented from eight benchmarks. Firstly the results of Blackbox profiling, packing and pin multiplexing are shown. Then with tool HBVPR we take results concerning area, power and delay of those benchmarks mapped in 6 different FPGA architectures. For comparison reasons the first architecture is in 180nm and the designs haven't passed the Blackbox-aware technology mapping step. The second architecture is in 180nm and its calibrated for decreased power consumption. The other four architectures are in 45nm, 65nm, 90nm, and 130nm calibrated for area and delay optimization.

Chapter 6

Future work

As this thesis has shown, today's academic research CAD tools are not completely updated to support Heterogeneous FPGA designs. Current tools need to be modified or new tools to be created in order : a) to support multiple different Heterogeneous FPGA architectures b) to expand the exploration space to include power estimation, a critical design parameter. This thesis proposed a complete framework for power estimation in heterogeneous FPGAs that includes both modified existing tools, and new tools to meet the goals mentioned above.

Blif format structure its restrictive concerning heterogeneity as seen in this thesis and requires a lot of conversions in order to fully support heterogeneous blocks. Edif file format is the standard output of every commercial and of many academic tools, so further research must be done in clustering and/or P&R tools to support this file format.

In the proposed framework research can be done in :

- Building libraries and an identifier to recognize the blackboxes and supply accurate power and delay values in HBVPR.
- Modifying HBVPR to make power-aware placement and routing.
- Modifying the framework to support 3D architectures.

Also research can be done in merging the Meander framework (see section 2.2.1) with NAROUTO design framework. Figure 6.1 shows the two frameworks side by side and the middle arrow shows a possible connection of the design flows.

Lastly a Graphical User Interface it would be a useful feature.

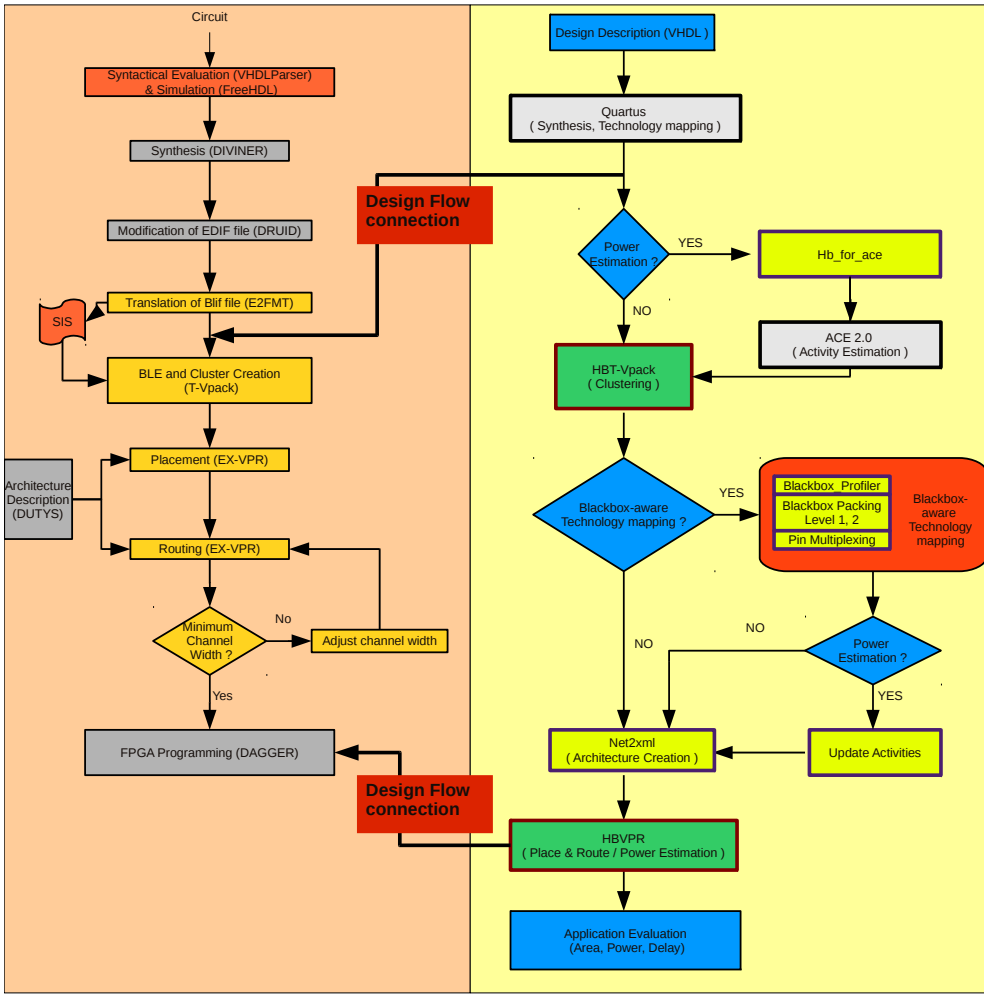

Figure 6.1: Meander and NAROUTO design frameworks.

Bibliography

[1] V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, ISBN 0-7923-8460-1

[2] Altera AN 311: Standard Cell ASIC to FPGA Design Methodology and Guidelines, April 2009

[3] Scott Hauck and Andre DeHon, Reconfigureable Computing, The Theory and practice of FPGA-BASED computation, ISBN 978-0-12-370522-8

[4] `http://embedded.eecs.berkeley.edu/mvsis/`

[5] Wayne Wolf, FPGA-Based System Design, 2004 PRENTICE HALL `www.phptr.com` ISBN: 0-13-142461-0

[6] `http://www.eecs.berkeley.edu/~alanmi/abc/`

[7] `http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html`

[8] `http://www.eecg.utoronto.ca/vpr/`

[9] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, and Jonathan Rose, VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling, *International Symposium on Field Programmable Gate Arrays (FPGA) 2009*, pp. 133-142

[10] Peter Jamieson, Jonathan Rose Edward S. Rogers, A VERILOG RTL Synthesis tool for Heterogeneous FPGAs, *15th International Conference on Field Programmable Logic and Applications*, August, 2005

[11] `http://www.ece.ubc.ca/~stevew/powermodel.html`

[12] Julien Lamoureux, Modeling and Reduction of Dynamic Power in Field-Programmable Gate Arrays, The University of British Columbia 2003, Phd Thesis

[13] Kara K.W. Poon Steven J.E. Wilton and Andy Yan, A Detailed Power Model for Field Programmable Gate Arrays , *ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 10, Issue 2, April 2005*, pp. 279-302.

[14] `http://www.users.muohio.edu/jamiespa/vpr_5_pow.html`

[15] `http://proteas.microlab.ntua.gr/meander/index.html`

[16] `http://vlsi.ee.duth.gr/amdrel/`

[17] KOSTAS SIOZIOS, "Design of a Configurable Logic Block and The Development of Supporting Toolset for embedded FPGAs", Master Thesis, Democritus University of Thrace, Greece, 2003.

[18] Introduction to the Quartus II Software Altera Corporation `www.altera.com`

[19] Quartus II Handbook Version 9.1 Volume 1: Design and Synthesis Altera Corporation `www.altera.com`

[20] `http://www-asim.lip6.fr/recherche/alliance/`

[21] Berkeley Logic Interchange Format (BLIF) University of California Berkeley February 22, 2005 `www.cs.uic.edu/~jlillis/courses/cs594/spring05/blif.pdf`

[22] Synthesis Design Flows Using the Quartus University Interface Program (QUIP) Version 1.1 February 6, 2008

[23] `http://www.eecg.utoronto.ca/vpr/architectures/`

[24] VPR and T-VPack User's Manual Summer 2008 VPR 5.0 Full Release, July 29, 2009

[25] Xiaoxiang Shi, FPGA CAD Research: An Introduction Report 1: 6/4/05 13/4/05