LEBANESE AMERICAN UNIVERSITY

POWER AND TEMPERATURE AWARE FUNCTIONAL UNIT BINDING IN HIGH LEVEL SYNTHESIS

By

LAYALE BASSIL

A project

Submitted in partial fulfillment of the requirements

for the Degree of Master of Science in Engineering

School of Engineering

June 2011

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

LEBANESE AMERICAN UNIVERSITY

School of Engineering and Architecture - Byblos Campus

Project Approval Form

Student Name: Layale Bassil I.D. #: 200101242

Project Title : Power & Temperature Aware Functional Unit Binding using Simulated Annealing

Program / Department: Master of Science in Computer Engineering

School : Engineering and Architecture

Approved by:

Project Advisor : Dr.Iyad Ouaiss

Committee Member : Dr. Zahi Nakad

Date : June 8, 2011

PROJECT COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY

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

Name: Layale Bassil

Signa [redacted] Date: June 8, 2011

PLAGIARISM POLICY COMPLIANCE STATEMENT

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

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

Name: Layale Bassil

Signatu

Date: June 8, 2011

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Iyad Ouaiss for his guidance throughout this project work. Thanks are also due to Dr. Zahi Nakad for being on my project committee.

I would like to express my sincere gratitude to the Lebanese American University whose financial support during my graduate studies made it all possible.

Finally, I would like to thank my friends and family for their long and patient support.

To my loving parents

Power and Temperature Aware Functional Unit Binding in High Level Synthesis

Layale Bassil

Abstract

This project elaborates on the possibility of optimizing the power consumed by the functional units by optimizing the functional unit binding technique. Functional unit binding maps the operations in each control step to specific functional units. However, the mapping between the operations and the available functional units has a profound effect on the power consumed. Hence, by optimizing the functional unit binding algorithm, it is possible to reduce the power consumption of the functional units which comprises a large fraction of the overall power of the design. The optimized power-aware functional unit binding methodology focus on reducing the switching activity of the functional units by minimizing the transitions of their input operands; this is done by trying to bind operations having one of its inputs remaining the same between two consecutive control steps, to the same functional unit.

The second part of this project tackled temperature reduction. The same methodology used for power reduction was used for temperature reduction by optimizing the functional unit binding technique. The optimized temperature-aware functional unit binding focus on reducing the temperature of the functional units by following a parabola-like cost function; the cost is the temperature dissipated by the functional unit for every two consecutive switching at its inputs. This will lead to a change in the binding of operations to functional units giving each functional unit the time to cool down between any two successive operations.

Keywords: High-Level Synthesis, Binding, Dynamic Power Consumption, Temperature Dissipation, Switching Activity, Leakage Current.

TABLE OF CONTENTS

CHAPTER ONE INTRODUCTION 1
CHAPTER TWO LITERATURE SURVEY 5
2.1 High-Level Synthesis 5
2.1.1 Control Data Flow Graph (CDFG) 6
2.1.2 Scheduling 6
2.1.3 Module Assignment 8
2.1.4 Register Assignment 11
2.2 Power Research 16
2.2.1 Power in CMOS circuit 16
2.2.1.2 Power Dissipation Equations 19
2.2.1.3 Switching Activity (SA) 20
2.2.1.4 Power Consumption Reduction 21
2.3 Temperature Research 25
CHAPTER THREE HLS TOOL 26
3.1 Input File 26
3.2 Resource bag 27
3.3 HLS Tool GUI 27
3.3.1 Input Tab 27
3.3.2 Output Tab 29
3.3.3 FUB Optimization Tab 29
3.3.4 Power Optimization Tab 30
3.3.5 Temperature Optimization Tab 31
CHAPTER FOUR 33
POWER AND TEMPERATURE AWARE FUNCTIONAL UNIT BINDING USING SIMULATED ANNEALING 33
4.1 Definitions and Problem Formulation 34
4.2 Switching Activity Calculation 35
4.2.1 Definition 35
4.2.2 Calculation 36
4.3 Power Awareness 37
4.3.1 Algorithm Description 37
4.3.2 Decisions & Restrictions 38
4.3.3 Motivation examples: 39
4.4 Temperature Awareness 50

4.4.1 Temperature Aware Functional Unit Binding 52
CHAPTER FIVE EXPERIMENTAL RESULTS 62
5.1 Power Results 62
5.2 Temperature Results 88
5.3 Analysis of the Results 97
CHAPTER SIX CONCLUSION 98
References 101

LIST OF FIGURES

Figure 1: Unscheduled CDFG Example 6
Figure 2: Resource Bag 1 8
Figure 3: Scheduled CDFG using List Scheduling and Resource Bag 1 8
Figure 4: Function Unit Binding 9
Figure 5: Multiplexers' distribution after Functional Unit Binding 10
Figure 6: Lifetime of the variables 12
Figure 7: Incompatibility graph 13
Figure 8: Compatibility Graph 13
Figure 9: Overall multiplexers' distribution 16
Figure 10: CMOS Inverter Mode for Static Power Consumption 17
Figure 11: Sources of Power Consumption in CMOS 19
Figure 12: Opportunities for Power Savings [25] 23
Figure 13: Example Node 26
Figure 14: HLS Tool GUI 28
Figure 15: Output Tab 29
Figure 16: FUB Optimization Tab 30
Figure 17: Power Optimization Tab 31
Figure 18: Temperature Optimization Tab 32
Figure 19: SA Calculation 36
Figure 20: Motivation example CDFG 39
Figure 21: Optimized overall multiplexers' distribution 44
Figure 22: Graph of the parabola 56
Figure 23: Temperature awareness DFG 58
Figure 24: Schedule of the Temperature aware DFG 59

LIST OF TABLES

Table 1: Schedule 45
Table 2: Register Binding 45
Table 3: ASAP Schedules & Resource Bags 63
Table 4: PAFUB - 4pDCT 65
Table 5: PARB - 4pDCT 65
Table 6: PAFUB&PARB - 4pDCT 66
Table 7: PAFUB - AR 66
Table 8: PARB - AR 67
Table 9: PAFUB&PARB - AR 68
Table 10: PAFUB - DCT1 69
Table 11: PARB - DCT1 70
Table 12: PAFUB&PARB - DCT1 71
Table 13: PAFUB - DIFFEQ2 72
Table 14: PARB - DIFFEQ2 72
Table 15: PAFUB&PARB - DIFFEQ2 73
Table 16: PAFUB - DIFFEQ4 74
Table 17: PARB - DIFFEQ4 75
Table 18: PAFUB&PARB - DIFFEQ4 76
Table 19: PAFUB - EWF 77
Table 20: PARB - EWF 77
Table 21: PAFUB&PARB - EWF 78
Table 22: PAFUB - FIR 78
Table 23: PARB - FIR 79
Table 24: PAFUB&PARB - FIR 79
Table 25: PAFUB - IIR 80
Table 26: PARB - IIR 80
Table 27: PAFUB&PARB - IIR 81
Table 28: PAFUB - LMS 82
Table 29: PAFUB&PARB - LMS 82
Table 30: PAFUB - nestor2 83
Table 31: PARB - nestor2 84
Table 32: PAFUB&PARB - nestor2 84
Table 33: PAFUB - wavelet 85
Table 34: PARB - wavelet 86
Table 35: PAFUB&PARB - wavelet 86
Table 36: All Techniques - All Benchmarks 87
Table 37: Maximum, minimum, and average improvements 87
Table 38: Temperature - 4pDCT 88
Table 39: Temperature - AR 88
Table 40: Temperature - DCT1 89
Table 41: Temperature - DIFFEQ2 90
Table 42: Temperature - DIFFEQ4 91
Table 43: Temperature - EWF 92
Table 44: Temperature - FIR 92
Table 45: Temperature - IIR 93

Table 46: Temperature - LMS 93
Table 47: Temperature - nestor2 94
Table 48: Temperature - wavelet 95
Table 49: Temperature - All Benchmarks 96

CHAPTER ONE
INTRODUCTION

According to Moore's law, the number of transistors, per unit chip area, would be doubled every two years. Driven by this fact, the exponential growth of circuit capacity and performance prompts new and critical challenges in terms of the power consumption of the integrated circuit and pauses new constraining factors in the design flow [24].

Nowadays, optimizing power consumption is necessary when designing a VLSI system. According to [23], optimizing power can start at the system level stage going down to the gate and transistor level stage. This wide range in the design stages affects the design decisions and the quality of the product and the improvement in the power consumed. Consequently, in order to get the highest improvement in the power consumption, designer has to start considering power as early as possible in the design stages. In the research done in [25], it is stated that power optimization techniques tackling the system and behavior level can achieve a power reduction in the range 40-70 %, which is the second highest possible improvement.

High-level synthesis transforms a circuit's behavioral description into a register-transfer level (RTL) design consisting of a datapath and a control unit. The resultant datapath will be composed of three different types of components: ALUs and multipliers as functional units, registers and memory as storage units, and finally buses and multiplexers as interconnection units. The three central synthesis tasks in a typical high-level synthesis system are: scheduling, allocation, and binding (or module assignment). Scheduling determines the sequence in which operations execute; allocation sets the

appropriate number of needed resources in terms of functional units, registers, and interconnection units; binding assigns operations to functional units and values to storage units, and interconnects those components to form a complete data path.

In this work, the objective was to optimize the dynamic power of the design. In order to achieve this objective, the focus was on the functional unit binding task of the behavioral synthesis. The reason behind this decision is that functional unit binding is considered the governing power source for modern VLSI circuits. The solution proposed here explores techniques to reduce spurious switching activity at the functional unit binding level with the use of a stochastic search algorithm, namely simulated annealing.

The other main focus of this work is the reduction of the temperature dissipated by the available functional units. The main idea was to reduce the temperature dissipated by each functional unit in order to avoid hot spots in the design. This was done by calculating the lifetime and the activity of each functional unit, and creating a cost function that will compute the temperature of each functional unit and will guide the simulated annealing decisions. A parabola curve was used for the cost function after calculating the optimal distance between any two consecutive switching couples at the inputs of any functional unit. More details about this method will be shown later on in this report.

The proposed approaches target the binding of operations to the available functional units and examine the room for improvement in the conventional functional binding techniques used in this work before applying the optimization. Optimizing functional unit binding for power and temperature reduction can be done using an efficient

swapping of operations over the available resources without violating the resource constraint. Knowing that simulated annealing tries to find an optimal solution, for the cost function used in the annealing process; when each simulation ends, a guaranteed better solution is reached. The new solution is most of the times very close to the optimal solution which is the global minima (in terms of power). One should keep in mind that simulated annealing doesn't guarantee an optimal solution, nevertheless it does guarantee a better solution and in the worst case, a final solution similar to the initial one.

The remainder of this report is organized as follows: Chapter 2 explores the literature for high-level synthesis and a description of the techniques implemented in terms of scheduling, and binding algorithms. It also explains, in details, the sources and types of the power dissipated in a digital CMOS circuit. It also describes the types of switching activities along with detailed explanation on the occurrence of each type. A focus on the spurious switching activity type is conducted because here resides the room for power improvement resides. This chapter is concluded by a summary on the previous research work done in the power optimization field. Chapter 3 introduces the HLS tool that was developed in order to implement and test the techniques used in this project. This chapter describes the implementations of the techniques developed in that tool with detailed examples and screenshots of the visual user interface. Chapter 4 presents, in details, the power aware register binding technique implemented in [1], and the work done in this project in order to improve this technique and get less power consumed by the functional units available. In addition to that, this chapter explains in details the other developed temperature aware functional unit binding technique. Chapter 5 displays the experimental results of the two techniques used (power & temperature

aware functional unit binding) in addition to an analysis of each output. Chapter 6 concludes the project and proposes some basis for future research.

CHAPTER TWO
LITERATURE SURVEY

In this chapter, we will introduce the techniques that are used in this work. First, a brief introduction about high-level synthesis is presented; then an explanation about the three major tasks of the high-level synthesis process which are: scheduling, allocation, and binding is also presented. This chapter also gives a detailed description about the power consumed in a digital CMOS circuit, gives an explanation about the power dissipation equations, how can it be estimated, and how can it be reduced. In addition to that, a research was also conducted about temperature and how it can be estimated in a circuit and how it can be reduced as part of this work.

2.1 High-Level Synthesis

High-level synthesis is the process of converting a behavioral description of a circuit into a structural description of that circuit. The outcome of this high-level synthesis process is a combination of data path and control logic. In other words, the process of synthesis starts from a description of the behavior of a given system in the form of inputs, functions, and outputs. The behavioral description of the system is then converted into a control data flow graph (CDFG). After having a CDFG representing the system, scheduling takes places where the operations of CDFG are scheduled in clock cycles. After scheduling, we get a scheduled data flow graph (SDFG) which will be used for module assignment where operations are assigned to operators or functional units. After module assignment is completed, register assignment takes place where all variables (input and output) are assigned to registers and then, as a final step,

interconnections between all the resultant components is achieved in order to form a complete datapath ([27], [35]).

2.1.1 Control Data Flow Graph (CDFG)

The behavior of a system design needs to be represented in a flow diagram. A useful flow diagram is the Control Data Flow Graph (CDFG) that represents the inputs to the design, the operations, and the outputs along with any necessary control sequences. The unscheduled CDFG is a group of nodes (operations) that depend on each other. An example of such a CDFG is presented in Figure 1.

Example: Suppose we want to represent the arithmetic expression: G = A*B + C*D + E*F into a CDFG.

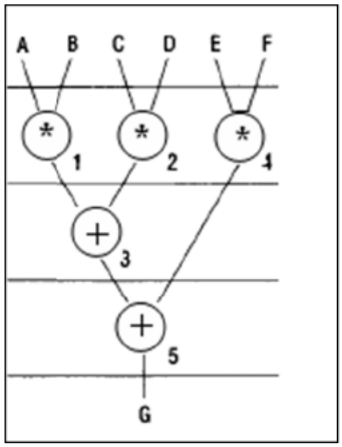


Figure 1: Unscheduled CDFG Example

In this project, each node is represented as shown in Figure 1, a circle with x inputs and y outputs, and an operation type in the center of the circle.

2.1.2 Scheduling

Scheduling involves determining the execution order of operations and the total number of control steps needed for the design. So, after scheduling, each operation is given a specific sequence step, which must satisfy the original dependencies of the graph. For

constrained scheduling, i.e. scheduling with a resource bag constraint (e.g. List Scheduling) which puts a limit on the number of available resources of each type, the maximum number of concurrent operations of any given type at any control step of the schedule is determined by the number of available resources of that type. While as for unconstrained scheduling (e.g. ASAP, ALAP), the maximum number of concurrent operations of any given type at any control step is a lower bound on the number of required hardware resources of the type.

In the unscheduled graph of Figure 1, node 2 cannot be scheduled before node 1 since the output of node 2 depends on the output of node 1, and same thing applies to the remaining nodes.

Different Algorithms that schedule a Control Data Flow Diagram (CDFG) based on different characteristics exist. As Soon As Possible (ASAP), As Late As Possible (ALAP), List Scheduling (resource constraint), and Force Directed (priority based) are some examples.

Module Bag contains the functional units available to implement a design. The number of resources needed by a design can be less or more than the number of resources available. The resources available for each design are read from a file. When the resources are limited in number, some of the resources are reused. To make that possible, the functions needing the same resource are scheduled at different control step [28].

The scheduled graph of the CDFG shown in Figure 1 using the list scheduling algorithm with the resource bag shown in Figure 2 is shown in Figure 3.

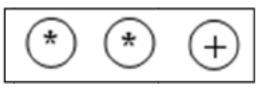


Figure 2: Resource Bag 1

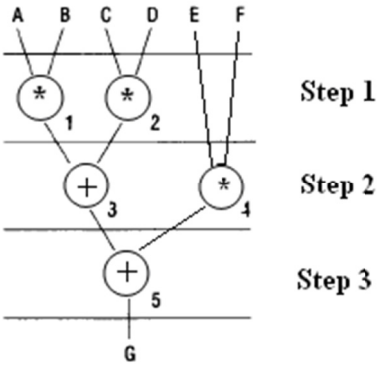


Figure 3: Scheduled CDFG using List Scheduling and Resource Bag 1

2.1.3 Module Assignment

After scheduling is completed, module assignment takes place; it assigns a module (functional unit) to each operation in the SDFG (Scheduled DFG). While performing scheduling, we can easily compute the minimum number of modules needed by counting the number of operations that exist in the same control step. If a resource constraint scheduling algorithm is used, the maximum possible parallelism, at each control step, is conditioned by the dependencies in the DFG and the number of available resources of each type. If unconstraint scheduling is used, the dependencies in the graph are the only constraint for the maximum parallelism, and hence the minimum number of resources needed of each type. Often, scheduling and module assignment occurs simultaneously to provide better results [27].

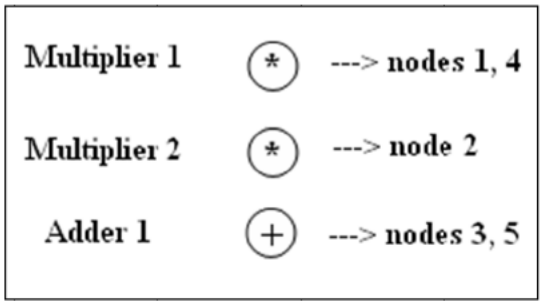
The algorithm used here starts from the first control step and assigns nodes to functional units sequentially. The List scheduling used here assures that the number of nodes assigned to functional units assigned in each control step does not exceed the number of

f unctional units availa ble. When all the nodes on a control step are assigned to f unctional units, the next control step is considered [28].

So, as we can see, the list scheduling technique does almost 90 % of the wor : since it a ssures that the number of require l resources does not exceed the available number for e ach step, so the functional unit inding tec nique used in this project does only the ork of specifying the nodes that are bound to the same functional unit by going over a l the nodes and assigning th em rando nly without caring for the number of ultiplexers that is nee led.

The result of the functi nal unit bi ding technique on the scheduled graph of Figure 3 is s own in the below Fig ure 4.

ear each resource, the numbers of the node s assigned to it are stated. Those numbers a e taken from the sche luled graph shown in igure 3.



Figur e 4: Functio Unit Binding

F rom the functional uni binding results, we c n conclude that more than one resource is u sed by more than one operation, so in order to be able to use the same resource by ore than one operation, and be ible to select the correct inputs at each control step, ultiplexers are neede l at the in ut of the resources and are used to select the right i puts at the right time.

In the scheduled example shown in Figure 3, the distribution of multiplexers at the input of the three different resources shown in the resource bag of Figure 2 is shown in Figure 5.

First, the outputs of the scheduled operations of Figure 3 are named based on the number of node, as follows:

Node number	Output name
1	O1
2	O2
3	O3
4	O4
5	G

So, from the binding result shown in Figure 4, a multiplexer is needed at the inputs of Multiplier1 and Adder1 because they both are used by more than one operation.

Four multiplexers are needed in this case, and their distribution is shown in Figure 5.

Note that the number of multiplexers will change after doing register binding because different nodes can be bound to the same register, and the need for a multiplexer could vanish, so this is not the final distribution for the multiplexers, the final one will be after doing register binding.

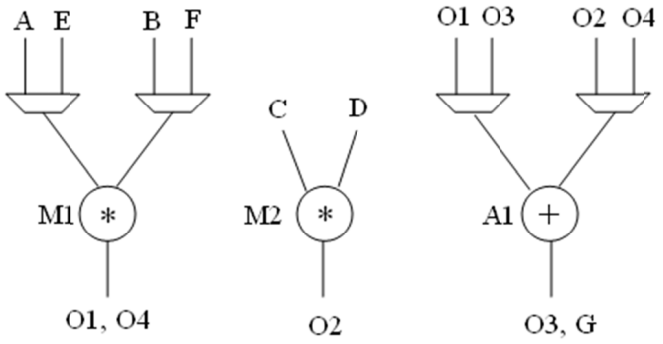


Figure 5: Multiplexers' distribution after Functional Unit Binding

2.1.4 Register Assignment

Figure 3 shows a DFG to which scheduling and module assignments have been applied. Horizontal lines in the SDFG (Scheduled Data Flow Graph) denote clock cycle boundaries or control step boundaries. In the register assignment task, the number of registers is specified along with the variables bound to each register. A register should be assigned to each input or output variable on a clock boundary, which is called register assignment [27].

Different techniques exist to bind variables to registers such as Left-Edge algorithm [29] and Clique partitioning algorithm [30]. The technique used in this project is the Clique partitioning.

Clique partitioning algorithm is an approach based on the heuristic proposed by Tseng and Siewiorek in [30]. It can be applied to any assignment; we used it in this work for the register assignment task only.

Clique partitioning is based on heuristic, thus, it does not guarantee optimal solutions [27], but it tries to reduce the number of registers used. It produces the minimum number of registers needed to save the values of the variables.

Clique partitioning needs the variable Lifetime. The lifetime of a variable is the time in which the variable is used or is saved to be used in subsequent steps [28].

For each variable the lifetime is determined by traversing the circuit and looking at the first occurrence of the variable. If a variable is given as an input and not used until control step x, then that variable needs to have a used sign for all those steps.

If a variable is written in control step x, and read in control steps y and z, where z>y, then the variable needs to remain in the register starting from time x all the way to time z [28].

The Lifetime of the variables of the graph shown in Figure 3 is shown in Figure 6.

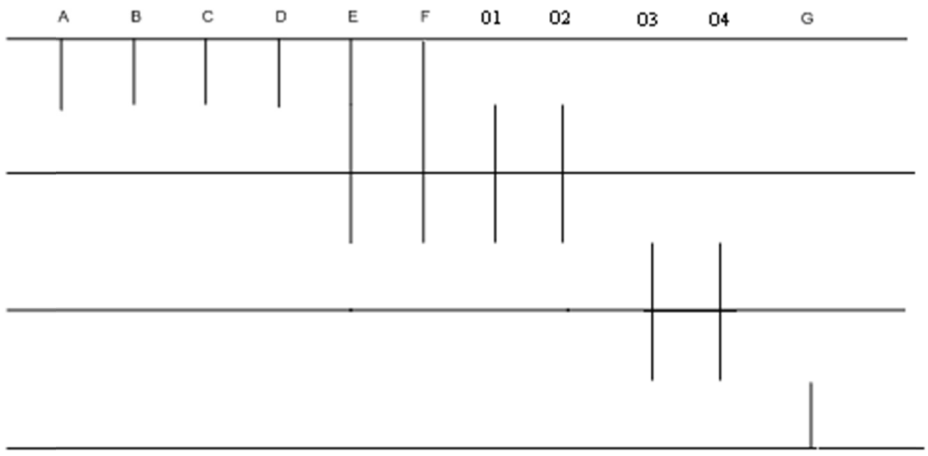


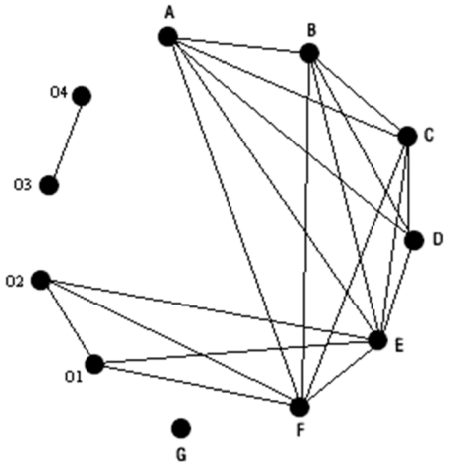
Figure 6: Lifetime of the variables

ariables should be alive in registers as long as they are needed in the execution of the circuit's operations. As mentioned earlier, the clique partitioning algorithm was used for the register assignment task. For this purpose, the compatibility graph of the variables in the DFG has to be created in order to create the minimum number of cliques which will be the number of needed registers. In order to create the compatibility graph, we have to find compatible variables that don't have an overlap in their lifetimes. Two variables are said to be ***compatible*** if there is no overlap at any clock boundary and there is no common region in their lifetimes. If those conditions are not satisfied, the two variables are ***incompatible***, and hence cannot share the same register.

Figure 7: Incompatibility graph

shows the incompatibility graph of input and output variables for the SDFG in Figure 3. Each variable of the graph (input and output) is represented by a vertex, and any two ***incompatible*** vertices (variables) are connected by an edge. The compatibility graph of variables is the opposite of the incompatibility graph and can be easily derived from the incompatibility graph. Figure 8 shows the ***compatibility*** graph of Figure 3 where, in

t his case, an edge conn ects now two ***compatible*** vertices (variables) which ca n also be b ound to the same regis ter.



Figur e 7: Incompatibility grap h

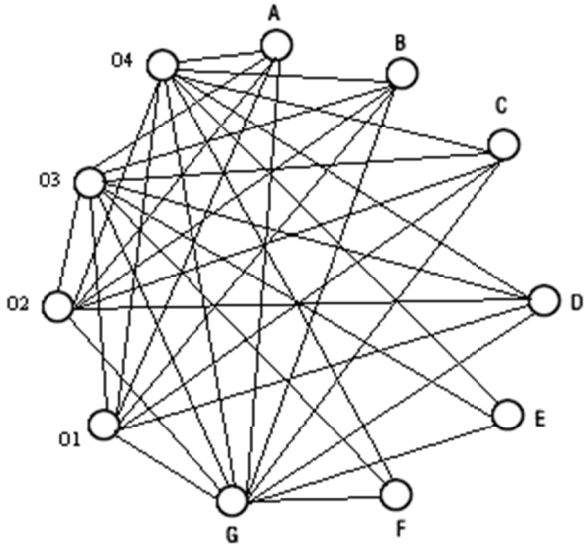


Figure 8: Compatibility Grap h

Once we get the co mpatibility graph of a certain DFG, the clique pa titioning a gorithms starts.

A complete graph is a graph in which in edge exists between every pair of vertices. A clique of a graph is a complete sub-graph (that follows the same definition of a complete graph). The size of a clique is the number of vertices in the clique; e.g. if a clique is composed of 5 vertices, this means that the size of the clique is 5. The clique portioning algorithms consists of partitioning the compatibility graph into a disjoint set of cliques. The objective of this algorithm is to get the ***maximum clique partition*** of the compatibility graph, and hence the smallest number of cliques or registers for this specific purpose. Clique partitioning uses two measurements as follows: a vertex v is called ***common neighbor*** of a sub-graph when the vertex v is not contained in the sub-graph and is connected by an edge to every vertex of the sub-graph. The second measurement is the ***non-common edge***; an edge is considered as such when it connects a vertex v, which is not a ***common neighbor*** of a sub-graph, but has an edge with at least one vertex of the sub-graph.

Steps to follow in the clique partitioning algorithm [31] using the above two measurements:

1. Locate the edge having the most number of common edges.
2. The two nodes of the edge are combined into one super-node.
3. Delete all the edges that link the two nodes, and add edges between the new super-node and all the other super-nodes that are common neighbors to the two nodes.
4. Repeat this process until no more edges exits, which means that no more nodes that don't overlap with each other’s exist.
5. The number of cliques found at the end, is the number of registers needed to save all the variables.

After the register assignment part is done, the number of multiplexers needed for the design changes from that shown in Figure 5. The result of the clique partitioning algorithm affects the number of multiplexers needed.

For the same example shown in Figure 3, the result of the clique partitioning technique is:

C

D

E

F

A, o1, o3

B, o2, o4, G

Number of Registers needed: 6, each one is used to store the variables in each of the six lines above.

After the register binding technique is done, the number of multiplexers needed in the design is:

mux 2-1: 2

mux 3-1: 1

mux 4-1: 1

The distribution of the multiplexers, before and after the functional units is shown in Figure 9.

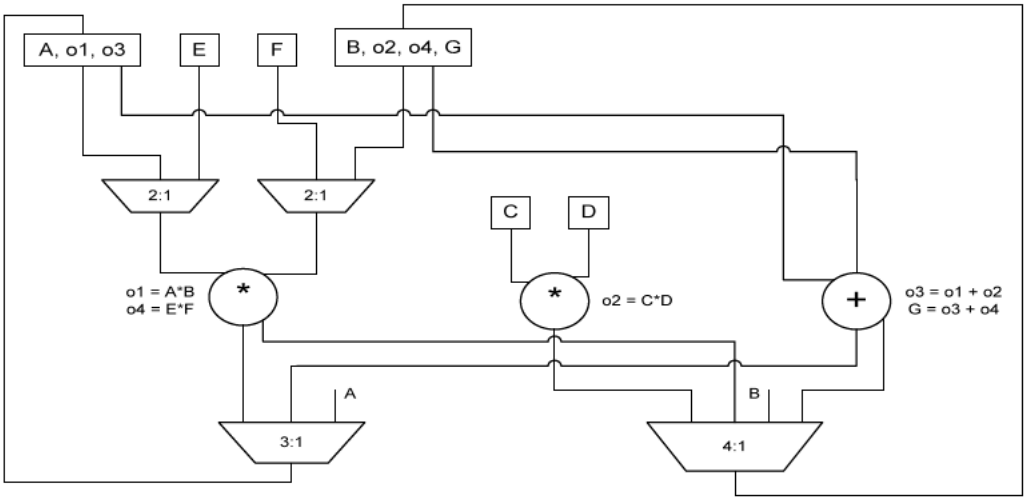


Figure 9: Overall multiplexers' distribution

2.2 Power Research

2.2.1 Power in CMOS circuit

2.2.1.1 Static & Dynamic Power Consumption

Traditionally, when designing VLSI systems, designer used to only consider the following two metrics only: performance (speed) and area. Lately, a new major metric is being taken into consideration which is power consumption. In order to measure the power consumption of a certain circuit, [2] stated that one should count the switches caused by the transitions occurring at the inputs of the circuit. This will be the theory used for the computation of the power consumption in this work later on.

Figure 11 shows the sources of power consumption in a CMOS circuit. As we can see, two major sources of power consumption exist: *dynamic power* and *static power*.

Knowing that static power consumption is usually very low, we will ignore the static power consumption in the power consumption calculation throughout this project and we will only consider the dynamic power consumption which contributes significantly

t the total power consumption of the whole design when the circuit's inputs are switching at a high frequency.

1. Static Power Consumption

Static power can also be called leakage power because it is the result of leakage current. Static power is consumed when all the inputs of the circuit are invariable, in other words, are kept at some valid logic level and the circuit is not in charging states [14]; to sum up, this type of power occurs when input isn't switching. In order to elaborate more on the static power consumption, we will refer to the behavior and the power consumed in a CMOS inverter. We considered the CMOS inverter in this elaboration because, in real life low-voltage devices, as it is the case here, an inverter exists at the input and output stage. Figure 10 shows the CMOS inverter modes.

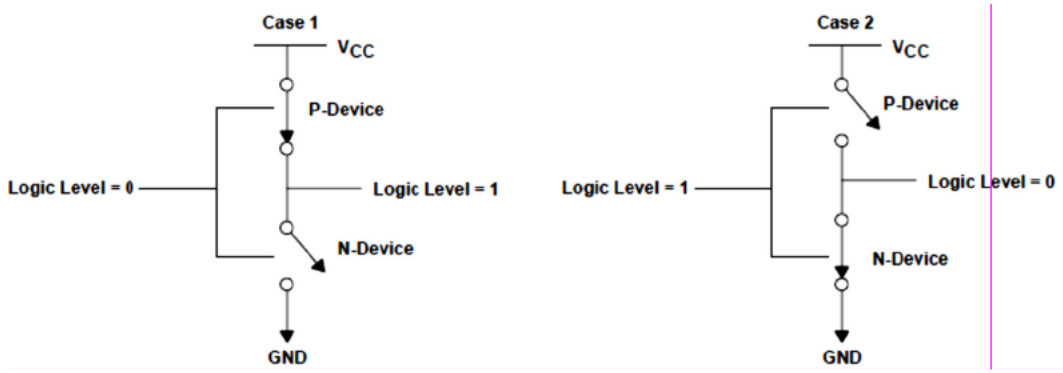


Figure 1 : CMOS Inverter Mode for Static Power Consumption

From Figure 10, we can conclude that when the input is at logic 0, as shown in Case 1, the N-Device is OFF and the P-Device is ON. The output voltage is V_{CC} or at logic level 1. This is also true when the input is at logic 1, in this case, the N-Device is ON and the P-Device is off and the output voltage is GND or at logic level 0 ⇒ one of the transistors (N-Device and P-Device) is always off and the other one is on when the gate is in either of these logic states (logic level 0 or logic level 1). Since no current flows

into the gate terminal, and there is no dc current path from V_{CC} to GND, the resultant quiescent current is zero and hence, the static power consumption (Pq) is zero. However, static power consumption is not completely zero because there is a small amount that is consumed due to reverse-bias leakage between diffused regions and the substrate [14].

Static power consumption in a CMOS device is caused by the leakage current I_{CC} (current into a device), along with the supply voltage. It is given by:

$P_S = V_{CC} \times I_{CC}$ (eq.1)

Where V_{CC} is the supply voltage and I_{CC} is the current into the device (sum of leakage currents)

2 Dynamic Power Consumption

Dynamic power consumption occurs when the inputs at the circuit are switching. In this case, the outputs of the gates are changing and transitions of the signals take place. In terms of capacitance, load capacitance (C_L) is charging and discharging.

The formula used to calculate the dynamic power consumption is:

$$P_{dynamic} = \frac{1}{2} * C * S * f * V_{cc}^2$$

Where the parameters of this formula are:

C = effective capacitance and

V_{cc} = supply voltage and

f = operating frequency and

S = switching activity of the circuit

Total power consumption is the sum of static and dynamic power consumption:

$$\boldsymbol{P_{total} = P_{dynamic} + P_{static}}$$

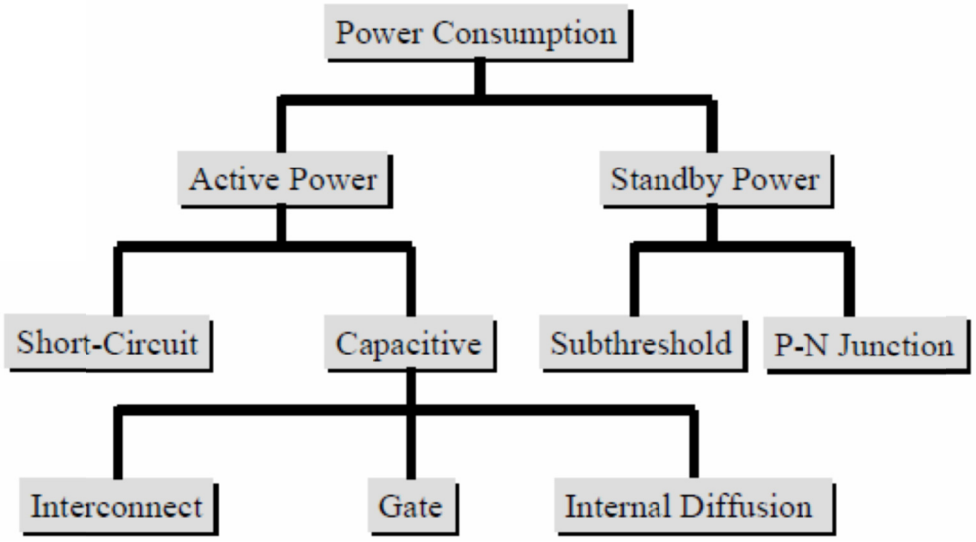


Figure 11: Sources of Power Consumption in CMOS

2.2.1.2 Power Dissipation Equations

a. Energy dissipated equations

We will compute the equations of the energy dissipated in a CMOS device because this is how we will compute the power consumed. The idea behind the power consumption was to count the switching occurring at the inputs of the functional units, this is justified as follows: during any switching at the input of the CMOS device, the energy stored in the load capacitor is dissipated. This applies to both, the N-Device and the P-Device, because in both cases, one of them will be conducting and the other in cutoff mode. The energy dissipated in the N-Device is given by (output switches from 1 to 0):

$$E_N = \frac{1}{2} C_L V_{DD}^2$$

Similarly, the energy dissipated in the P-Device device is given by (output switches from 0 to 1):

$$E_P = \frac{1}{2} C_L V_{DD}^2$$

The total energy dissipated during one switching is the summation of E_N and E_P:

$$E_T = E_P + E_N = \frac{1}{2} C_L {V_{DD}}^2 + \frac{1}{2} C_L {V_{DD}}^2 = C_L {V_{DD}}^2$$

The power dissipated, in terms of switching frequency, can be written as:

$$E_T = P * T \Rightarrow P = \frac{E_T}{T} \text{ where f} = \frac{1}{T} \Rightarrow P = f E_T = f C_L {V_{DD}}^2$$

The formula of the power dissipated in the CMOS inverter implies it is directly proportional to switching frequency f and the supply voltage ${V_{DD}}^2$.

b. Dynamic capacitive power

The formula for dynamic power is given by:

$$P_{dynamic} = f C_L {V_{DD}}^2$$

Observations:

From the above formula of the dynamic power, one can conclude that the dynamic power does not depend on the sizes of the device or on the switching delay. But it does depend on the load capacitance C_L, on the clock frequency f or switching frequency, and on the supply voltage V_{DD}.

Conclusion: the dynamic power is not a function of transistor sizes, but is a function of switching activity!

2.2.1.3 Switching Activity (SA)

Switching Activity *SA* can also be considered as the activity of a functional unit *fu*, whenever there is a change in its input operands. As a result, the operand variability of the functional unit's inputs, or its SA, affects its power consumption.

In this work, we will consider only the power consumed by the available functional units due to their large contribution to the power consumption of the data-path.

The switching activities within the functional unit accounts for the majority of the power dissipation of a multiplier, as given in the following equation:

$P_{switching} = \alpha C_L V^2_{DD} f_{clk}$

Where C_L is the loading capacitance, V_{DD} is the supply voltage and f_{clk} is the operating frequency or switching frequency and finally the parameter α is the switching activity [5].

2.2.1.4 Power Consumption Reduction

Continuous design challenges faced by IC designers are becoming more and more critical due to the increase in the designs' complexity, in addition to the use of deep sub-micron 90 nm processes. Subsequently, being able to compromise between performance and power is becoming essential but more difficult.

Higher power consumption requirements make designers suffer if they want to benefit from the performance and features that current FPGAs provide. As mentioned earlier, efficient power optimization techniques are a must for a design having maximum performance and minimum power consumption [32].

During design, one can achieve power consumption minimization in a number of ways: according to [14], minimum-size devices is advantageous when trying to minimize the static power consumption because, as mentioned earlier, this source of power consumption is directly proportional to the area of diffusion for being directly proportional to the leakage current. One way to apply this solution is by using low-power devices (having a supply voltage in the range 1.5 to 3.3 V).

On the other hand, dynamic power consumption is proportional to the supply voltage, load capacitance and the clock frequency. Consequently, [14] also proposed a solution in order to decrease the dynamic power consumption. This solution can be the reduction of any of the terms to which this source of power is proportional which are the supply voltage (V_{DD}), switched capacitance (C) and the clock frequency (f).

The solution proposed in [23] is close to that proposed in [14]. [23] also proposed a reduction in the factors to which the dynamic power is proportional with examples on how to reduce each factor. For example, in order to reduce the load capacitance, [23] suggested the use of a smaller number of resources. In his work, [23] suggested decreasing the supply voltage to which the dynamic power is proportional. This can be done by using variable supply voltages that drive the operations in the design. Hence, lower supply voltage can be supplied to the operations that don't belong to the critical paths. This will reduce the dynamic power consumed without negatively affecting the performance of the design [23].

A reduction in power consumption provides several benefits. One of the benefits is that less power consumed = less heat is generated, and hence less problems associated with high temperature, such as the formation of on-die hotspots, or the need for heatsinks. On the other hand, high temperature requires the use of more effective cooling and packaging solutions in order to avoid all the drawbacks of high temperature, causing a noteworthy increase in packaging and cooling cost. Subsequently, less generation of heat offers to the user a product that costs less on one hand, and that is more reliable due to lower-temperature stress gradients on the device [14].

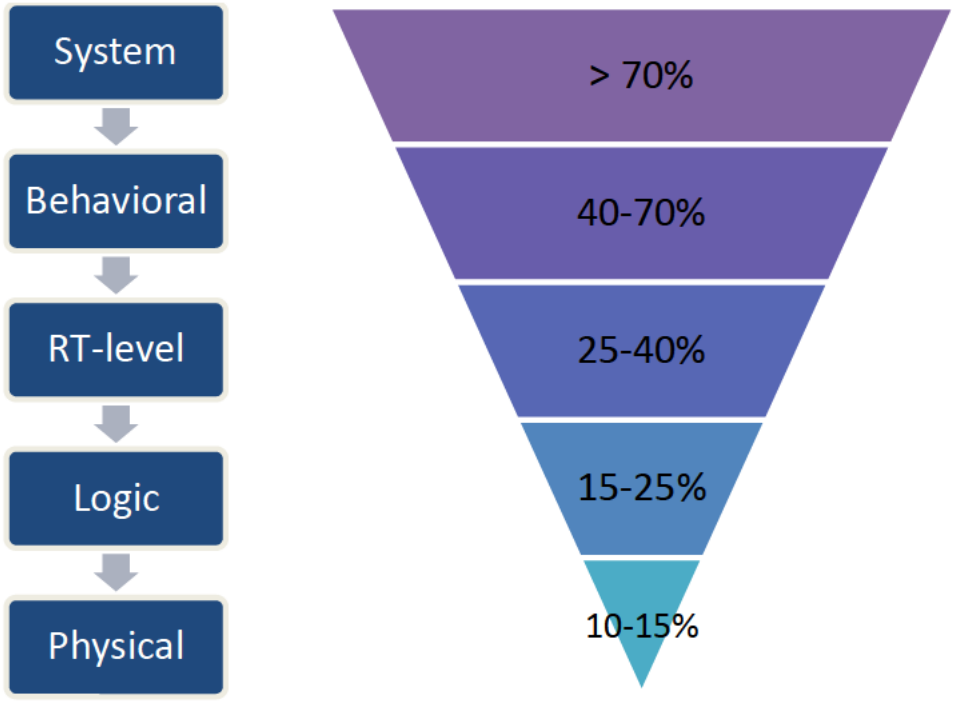


Figure 12: Opportunities for Power Savings [25]

The work done in [32] also explains the possible various techniques to optimize both static and dynamic power consumptions. This was done by tackling the architecture, software, and design methodology areas. From the architectural point of view, it is important to choose the right design architecture and to employ the right design techniques aiming at reducing the power consumption keeping a balance with the performance and the cost of the design. From the software point of view, dynamic power can also be reduced when using synthesis tools that will map the functions to the general logic in an intelligent way after using the specialized dedicated blocks.

And finally, the design methodology can minimize the dynamic power consumption of a design by reducing needless and unintended switching. This can be achieved by optimizing design algorithms. An additional room for improvement is by reducing the switching on clock networks. Dynamic power consumption can also be significantly

minimized using this methodology because, clocks in a circuit, have a high switching activity and long paths, and hence consume a good amount of power [32].

In addition to the above stated works that were developed to reduce the power consumption, some references try to save significant power consumption of a VLSI design by developing an optimized version of the multiplier resource. [3] developed a low power high performance multiplier that uses an SPST adder in the addition phase of the multiplication and this will avoid the unwanted addition and thus minimize the switching power dissipation. Also, [5] developed a low power shift and add multiplier. According to [5], the multiplier is one among the functional components of many digital systems, the reduction of power dissipation in multipliers should be as much as possible.

For getting the low power low area architecture, the modifications made to the conventional architecture consist of the reduction in switching activities of the major blocks of the multiplier, which includes the reduction in switching activity of the adder and counter. This architecture avoids the shifting of the multiplier register. The simulation result for 8 bit multipliers shows that the proposed low power architecture lowers the total power consumption by 35.25% and area by 52.72 % when compared to the conventional architecture. Also the reduction in power consumption increases with the increase in bit width [5].

Another reference gave special attention to the adder/subtractor resource. As per [4], a spurious power suppression technique (SPST) was proposed, leading to the use of a low-power adders/subtractors in the design in order to reduce the total power consumed by the whole HLS process. The proposed SPST separates the adders/subtractors into two parts: MSP = Most Significant Part and LSP = Least Significant Part. Whenever the

most significant part doesn't affect the computation results, its circuits are turns off. The solution proposed in [4] results in a power reduction of 31.9% with an area overhead of 20.9%.

Figure 12 shows the opportunities for power savings. As we can see, optimizing the "Behavioral" phase of the flow, which consists of scheduling, binding, pipelining, and behavioral transformations, has the second highest impact on the power consumption where it constitutes 40-70% of the consumed power.

In this project, we will focus on reducing the switching activity (*SA*) of the available functional units in order to minimize the total power of the design.

2.3 Temperature Research

Power and temperature optimization in high-level synthesis has been an active research area in the recent past [11].

Power and temperature are very much related. In other terms, high spatial and temporal power densities (power dissipation per unit area) lead to high temperatures, which result in an increase in leakage power consumption.

In addition to the above, the count of transistors in today's electronic systems is steadily increasing. Heat dissipation, being proportional to the transistor counts, will rise steeply as a consequence of this trend [18].

Moreover, temperature heavily influences the fault processes. For example, high temperature leads to electromigration, dielectric breakdown, and power–thermal cycling that result in a large number of IC permanent faults [11].

To sum up this section, it became obvious that the optimization of the thermal properties of integrated circuits is a crucial step to take for a reliable system having a low power and a good performance.

CHAPTER THREE

HLS TOOL

This project was developed in a java environment. The input files, the resource bags, and the output files have the same properties as done by [33].

3.1 Input File

An input file starts by the number of nodes in the graph, and then each node is written with the type of operation, the number of outputs, their names, and finally the number of inputs and their names. The input file looks as shown below. In this representation, only one node shown in Figure 13 is represented, but the rest of the file has same format for all the remaining 34 nodes.

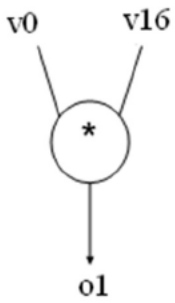


Figure 13: Example Node

```
                35      → number of nodes
                1       → number of outputs
                o1      → output name
1 Node          *       → function type
                2       → number of inputs
                v0      → input name
                v16     → input name
```

3.2 Resource bag

A resource bag file starts by the number of the different resource types, and then each resource type is represented by the available number and the name of the resource. A resource bag example is shown below. In this example, we represent a resource bag of two adders and two multipliers.

2 → number of resource types

2 → number of resource of the succeeding type

+ → resource type

2

*

3.3 HLS Tool GUI

This is a continuation of the GUI done by [33]. Additional Features were added to simulate the implemented power aware functional unit binding technique. These are demonstrated in the following snapshots from the tool.

3.3.1 Input Tab

In this tab, the user selects the input file and the resource bag following the formats specified above, and then selects the desired technique that he wishes to use.

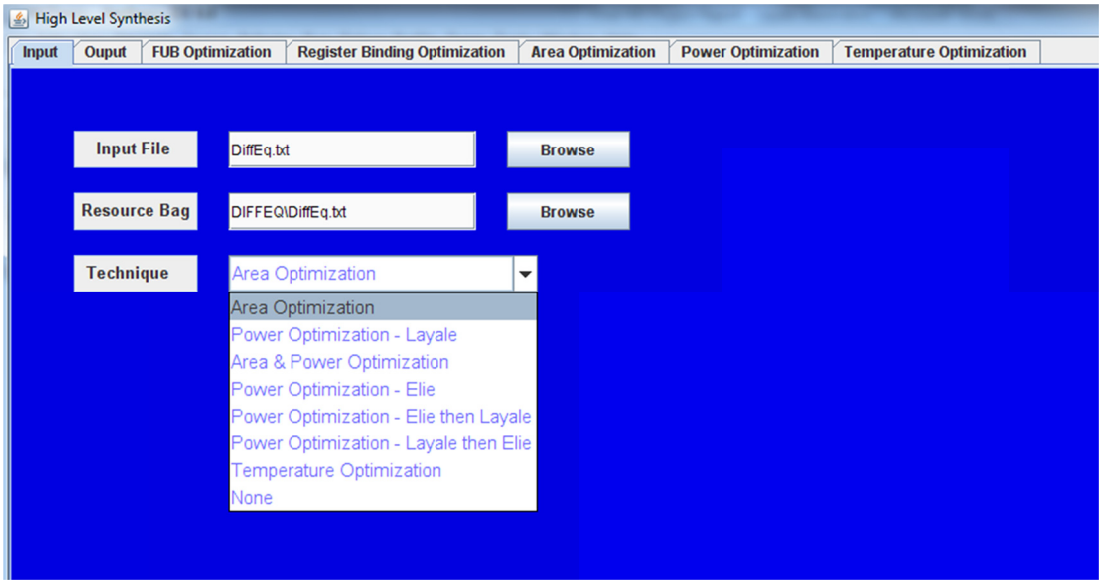


Figure 14: HLS Tool GUI

Figure 14 shows a complete view for the GUI of the HLS tool. Some changes were made to the GUI done by [33][28]. First, the four buttons shown at the bottom were removed. A complete tab, related to power was added in order to test the result of the selected technique. In the first tab, the user can now select the desired technique. The first selection only executes the improved functional unit binding technique implemented by [28] to reduce the area. The second selection in the drop down executes the power aware functional unit binding developed in this work. A combination of the work done in this project and the power aware register binding technique developed by [1] was also made available through the fifth and the sixth sections in the drop down. Those selections execute both the improved register binding technique implemented by [28] and the improved functional unit binding technique proposed here simultaneously. The seventh selection in the drop down executes the temperature aware functional unit binding developed in this work. Additional details about the outcome of each selection will be given in chapter 4.

3.3.2 Output Tab

In the output tab, the result of the selected technique will be displayed. As we can see in Figure 15, we can see the resultant schedule using the list scheduling algorithm, the life times of the variables in our design, and the register bound set result using the clique partitioning algorithm.

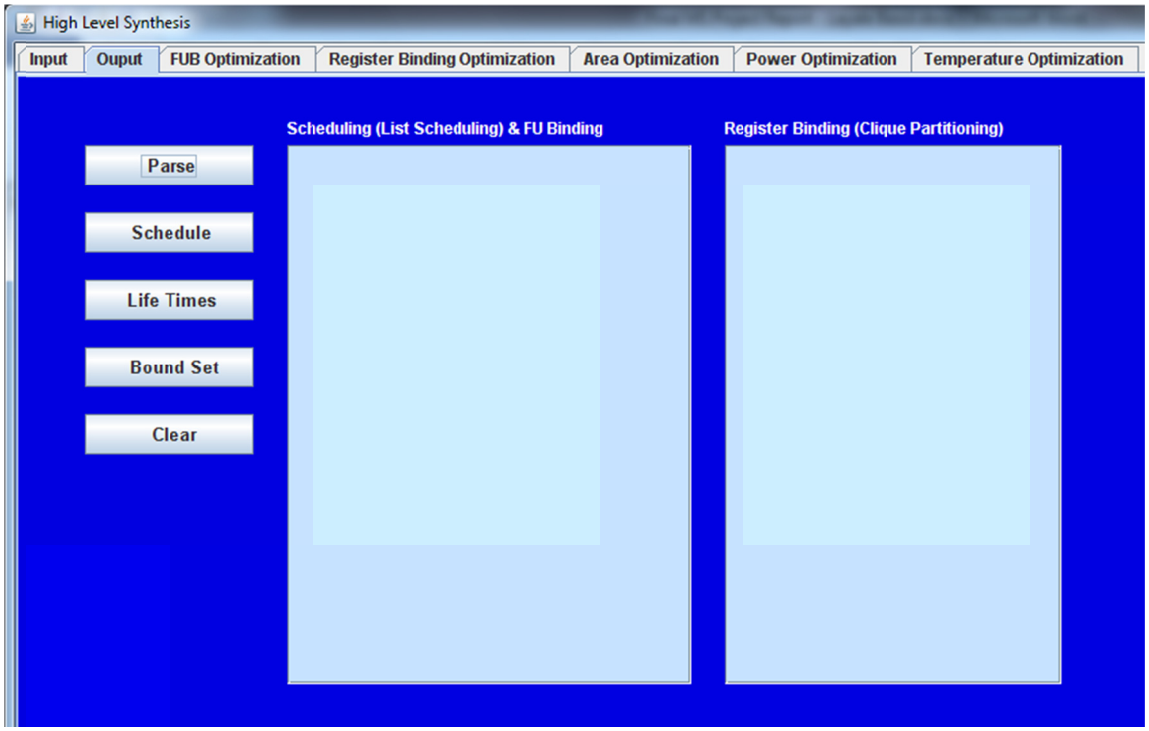


Figure 15: Output Tab

3.3.3 FUB Optimization Tab

In this tab, the user can see the result of the optimized functional unit binding technique used here to improve the power of the design. In Figure 16, the user can list the initial functional unit binding outcome, which means how operations were distributed over the available functional units, and the in the optimized distribution of operations taking into consideration a power aware biding.

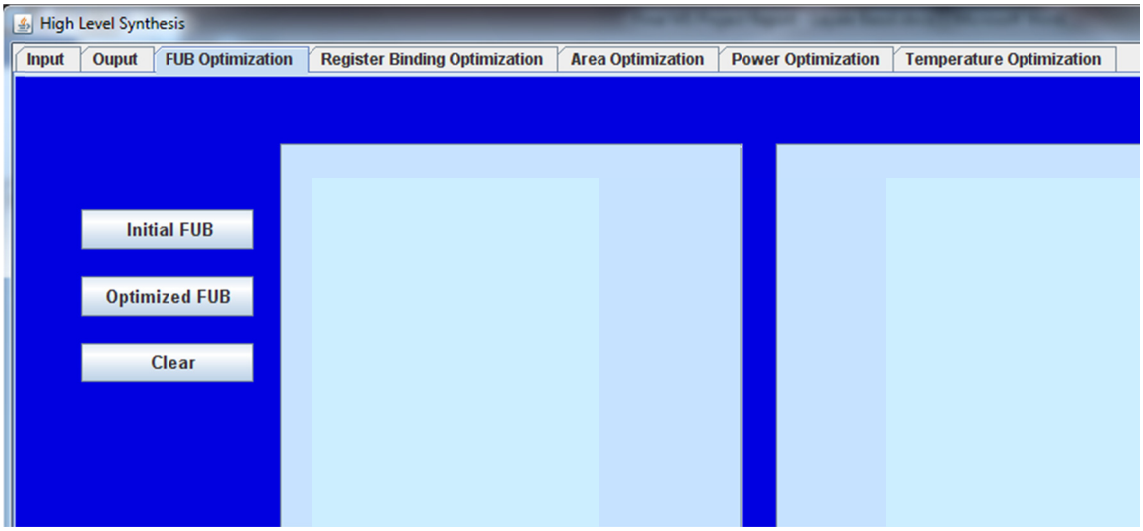


Figure 16: FUB Optimization Tab

3.3.4 Power Optimization Tab

In this tab, the user gets a detailed explanation on the calculation of the power. As shown in

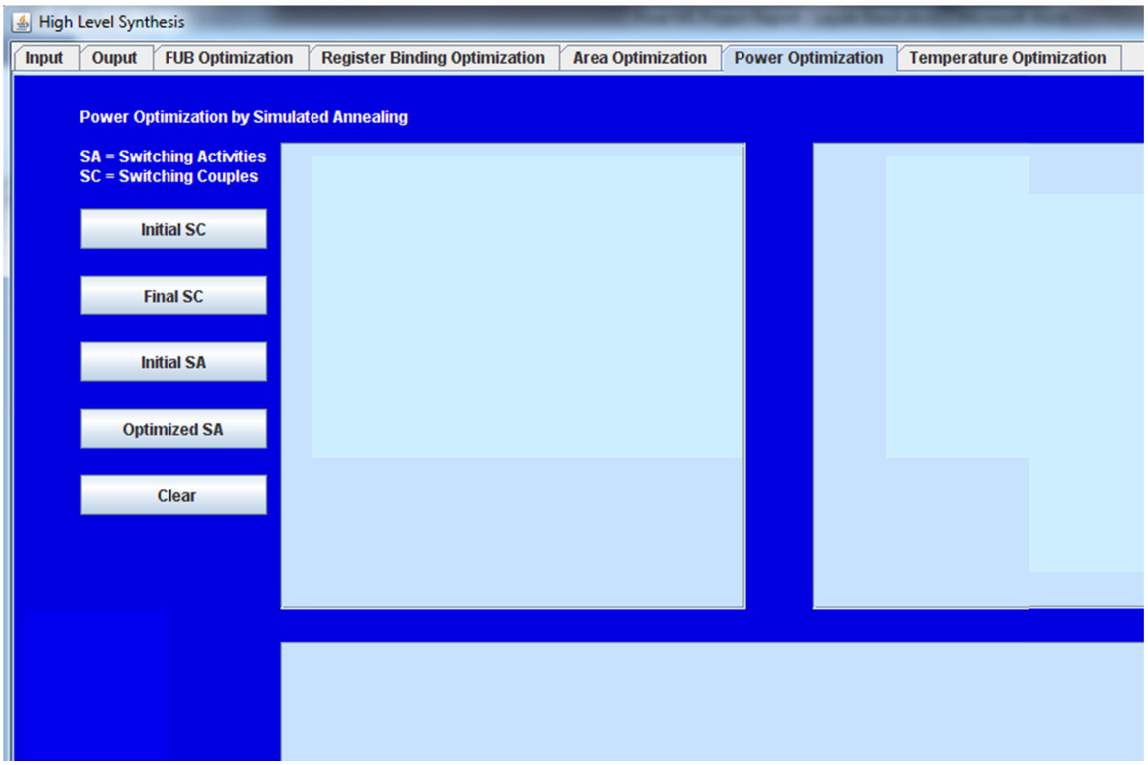


Figure 17, the user can list the initial and final switching couples, which means before and after the optimization technique. And he can get the improvement in terms of SA by

printing it before and after the optimization technique with a percentage of improvement.

3.3.5 Temperature Optimization Tab

In this tab, the user gets a detailed explanation on the calculation of the temperature. As shown in Figure 18, the user can list the initial and final temperature occupied by all the available functional units in the design, which means before and after the optimization technique. And he can get the improvement in terms of temperture by printing it before and after the optimization technique with a percentage of improvement.

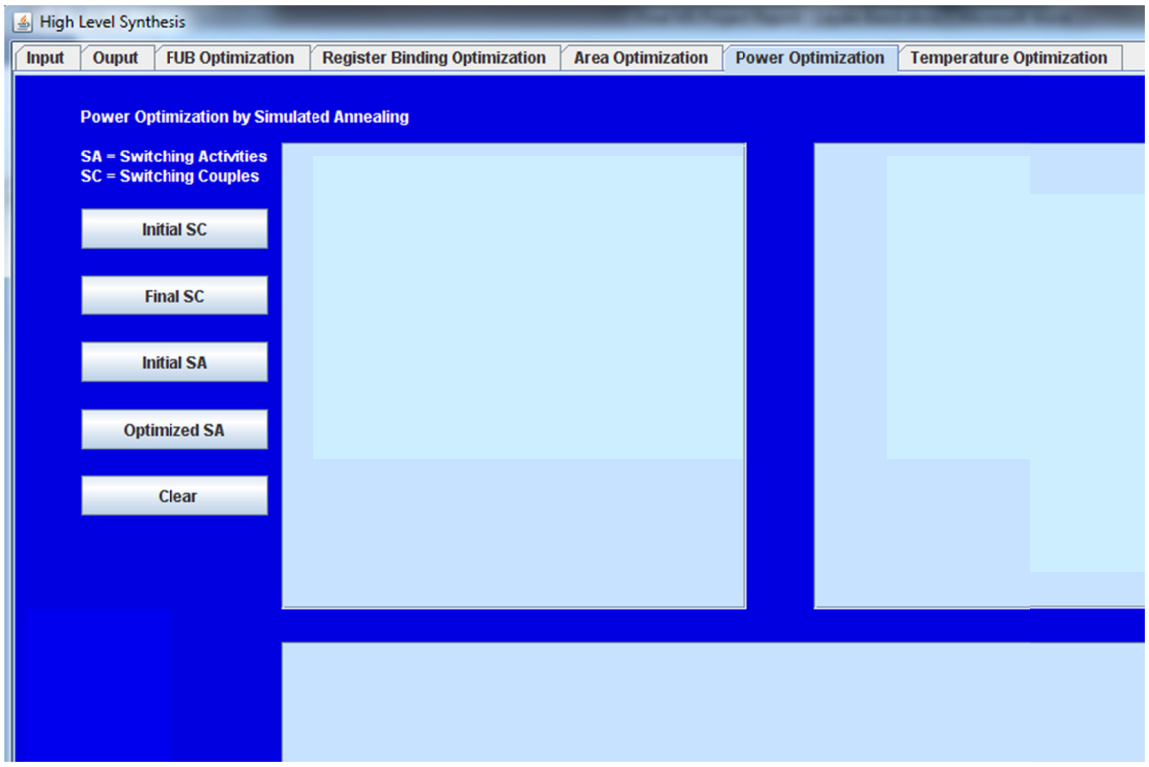


Figure 17: Power Optimization Tab

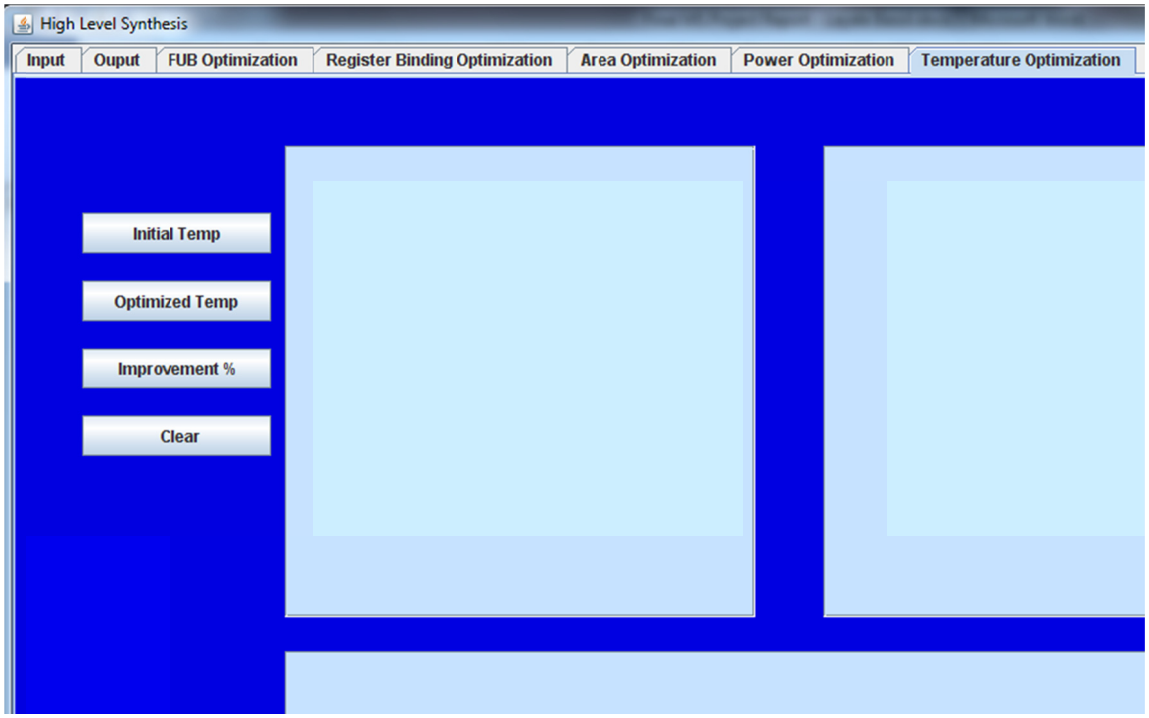


Figure 18: Temperature Optimization Tab

CHAPTER FOUR

POWER AND TEMPERATURE AWARE FUNCTIONAL UNIT BINDING USING SIMULATED ANNEALING

Thermal concerns have gained significant attention nowadays due to the continuing decrease in the feature size and the resultant increase in the chip density and the clock frequency of VLSI circuits. Thus, low power and temperature chip design is becoming a must [34].

[1] has worked on optimizing the register binding in order to decrease the Spurious Switching Activities. This optimization was done by switching 2 variables or by switching one variable with a set of variables. What has been done here is the work on optimizing the functional unit binding technique, and then apply the optimized register binding technique and see the improvement when using the optimized function unit binding alone, and then when using both optimized techniques simultaneously.

What was also done in this work is the incorporation of the area cost with the use of the optimization technique developed by [1] as stated in the future work section. Add to this, the area was also incorporated in the work developed in this project, which is the optimization of functional unit binding technique in order to improve the power of the design, and no area overhead resulted.

According to the work done in [8], power consumption can be reduced by increasing the number of registers in the design, but this solution is not area-friendly and increases the area of registers and hence the total area of the design. The best case in terms of power

is to assign each variable to a separate register, but on the other hand, this solution leads to the worst case in terms of area.

The goal of the optimized power aware resource-binding algorithm is to reduce power consumption in the FUs once the scheduling and FU-binding tasks have been done without having any area overhead.

4.1 Definitions and Problem Formulation

In this work, we directed our attention to the functional unit binding task of the high-level synthesis flow, which highly impacts the activity on a resource.

In this project, we will consider only the power consumed by the available functional units. This decision was taken due to the large contribution of the functional units due to in the total power consumption of the data-path. The power consumption of a functional unit depends on the operand variability of its inputs. Hence, a functional unit consumes both useful (non-spurious) and useless (spurious) power. Useful power is consumed when the functional unit is executing an operation while useless power is consumed when the functional unit is idle and there is an input operand transition [7]. In order to reduce the useless power, we will try to minimize the number of input changes at the idle units.

A significant fraction of the total power consumption of the available functional units occurs when the functional units produce useless outputs, hence spurious power is consumed. Knowing that the power consumed by the functional units dominates the overall power consumption of the design, optimizing it is a good room for improving the overall power consumed by the design [26].

4.2 Switching Activity Calculation

4.2.1 Definition

Two types of switching activities can be minimized: the intra−transition SA (occurring during the propagation of a single input vector) and the inter−transition SA (occurring between different input vectors) [23]. In this work, we will focus on the intra-transition switching activities.

According to [1], it can be observed that unnecessary switching exists in the following cases:

1. At the first operation executed in a functional unit.

   This case occurs when the operation is the first operation bound to the functional unit. There might be a switching between a "junk" value and another variable bound to the registers in case a correct value was not bound to the register until a certain control step.

2. At intermediate operations executed in a functional unit.

   The second case of spurious switching occurs in between operations and can also be called intra-transition switching activity. This happens frequently because, usually, operands (variables) of an operation have different life times; the life time of a variable is the difference between its end time and start time. Sometimes, even if operands have similar start times, the registers that store the operands of the operation and are connected to the inputs of the functional unit do not get changed through multiplexer routing. This cause all the variables bound to a certain register to be available at the inputs of

that functional unit and result in spurious switching between the available variables when the functional unit executes useless operations.

3. At the final operation executed in a functional unit.

   This case of spurious switching is quite similar to the first type of spurious switching. After the last operation, the selectors of the multiplexers at the inputs of the resource will not change. As new values are written to the registers, these variables will be directly consumed by the resource and will therefore cause the resource to switch spuriously.

4.2.2 Calculation

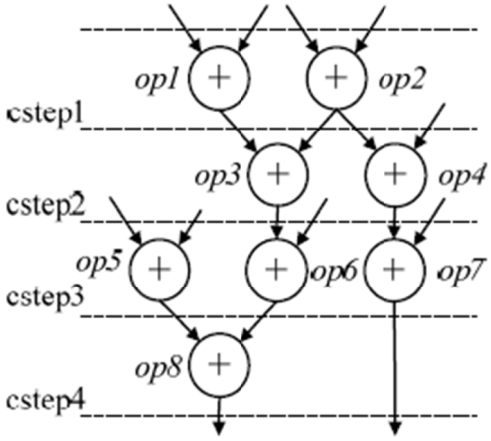


Figure 19: SA Calculation

In the example shown in Figure 19, the SDFG needs three control steps. The resource bag contains 3 adders; this was the result of the maximum number of adders needed in step3. When a new input vector *PI* arrives at the primary inputs, after four cycles, all the corresponding outputs for the design are computed.

If we want to compute the switching activities of adder1: there is switching on the adder1's input ports when there is a switch in the operations that this adder is executing. So, a switching occurs at the input ports of adder1 in control step 1 executing op1,

control step 2 executing op3, and finally control step 3 executing op5 when the first vector of inputs, PI^1, propagates through the design. The type of switching activity occurring at the input ports of adder1, in one complete iteration to propagate PI^1, is called the **intra-transition SA**. The second type of switching activity is known as **inter-transition SA**, and occurs when another vector of inputs is feeding the design or across iteration boundaries. For the case of adder1, when the primary inputs are switching from PI^1 to PI^2, the inputs at adder1 will change according to the new vector PI^2 and the operation executed by adder1 will actually switch from op5 back to op1 in order execute the new vector. This is when inter-transition switching activity takes place [23].

4.3 Power Awareness

4.3.1 Algorithm Description

This algorithm is an optimization of the normal functional unit binding aiming at reducing the total power consumed by the available resources in the design by minimizing the switching activities (SA) of the functional units. According to the motivation examples shown in section 4.3.3, room for improvement is there and this was the reason for the development of this power aware functional unit binding technique. As we will see, swapping the operations bound to the available functional units, can improve the power consumed by reducing the switching activity.

As we all know, hardware sharing may cause an increase in the switching activity thereby increasing the power. This can be explained by the additional need of multiplexers and registers. However, one instance of hardware sharing which reduces power consumed is when the same functional unit performs two operations with one of its inputs remaining the same [36].

After the synthesis flow ends, which means after scheduling, resource binding, register binding, and the interconnections are specified for a given DFG, the optimized functional unit binding is run on top of the final result trying to improve the power. This algorithm is composed of a simulated annealing having a cost function the summation of the switching activity of the available functional unit binding. The simulated annealing runs for a good amount of time, trying to swap the binding of operations over the available resources. A possible swap between two nodes (operations) has to verify the below conditions:

1. More than one available functional unit of the same type as the one needed by this node.
2. There should be another node, having the same functional unit type but bound to another functional unit (by checking the names of the functional units) and is scheduled on the same sequence step.
3. Two nodes abiding by the above two conditions can be paired together as a possible swap.
4. So all the possible pairs of nodes are collected and will be passed to the simulated annealing trying to optimize the binding for a better power.
5. Also, if a node is alone on a sequence step, but there is more than one functional unit of the same type as this node's operation, so this node is added to another vector to reconsider for switching between all the available functional units.

4.3.2 Decisions & Restrictions

4.3.2.1 Decisions

This work consists of a technique that reduces the dynamic power of the design without affecting performance. For this purpose, a power-aware functional unit binding

algorithm was developed. The goal of this algorithm is to increase the potential for a functional unit (FU) to reuse an operand. This goal was achieved by a wise binding of operations to the available functional units by trying to maximize operand reutilization. A detailed example will be given in the motivation section next. The design for low power tries to spot cases when an operand is reused by two operations consecutively executed in the same FU and bind them to the same functional unit.

"A functional unit does not perform a spurious operation when both of its inputs are unaltered" [13].

4.3.2.2 Restrictions

1. No possible improvement exists in the case of one resource of any type; i.e. if only one MULT, or ADDER, or SUB, etc…exit, this means there will be room for swapping bound operations.

4.3.3 Motivation examples:

4.3.3.1 Example 1:

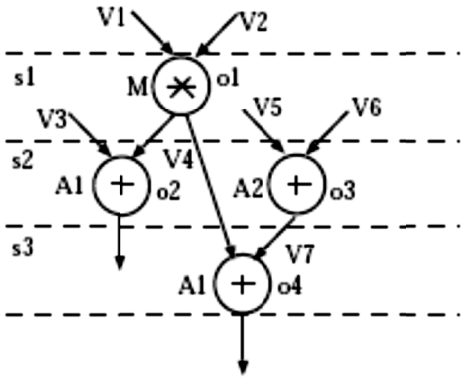
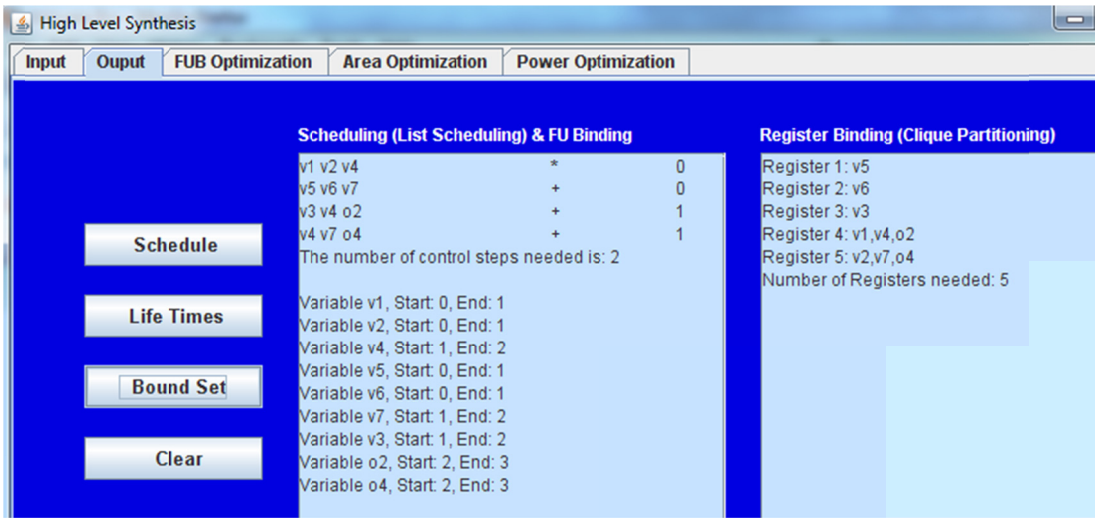
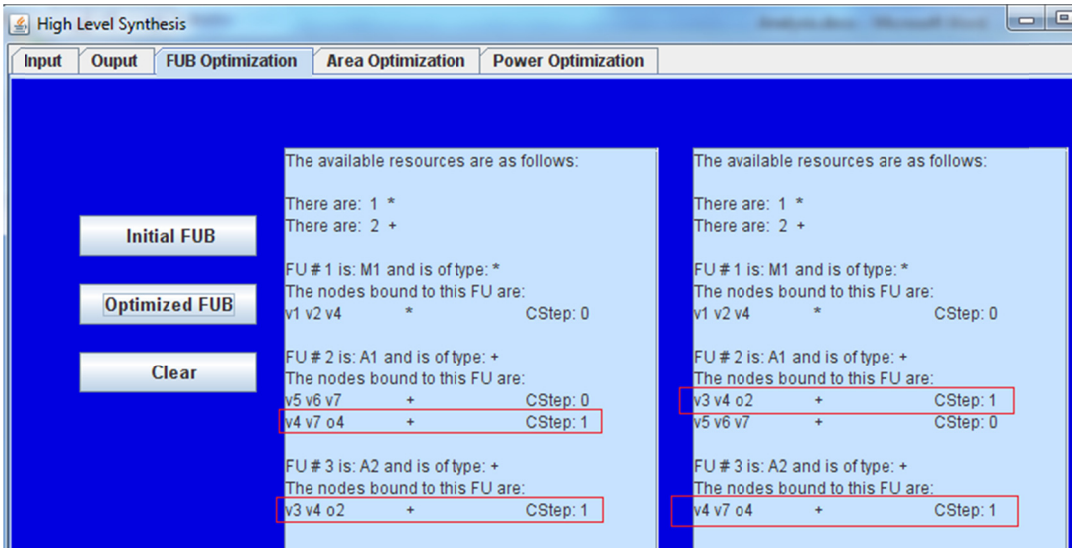


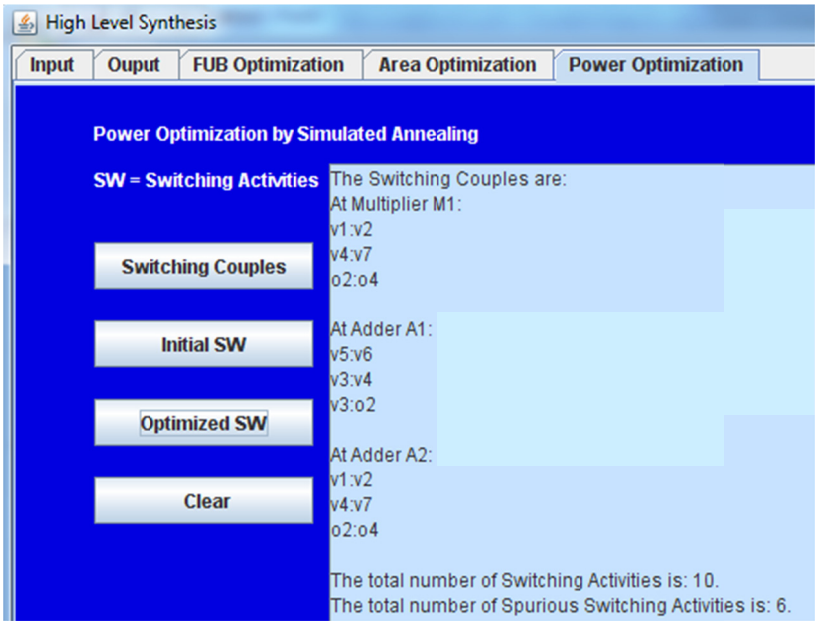
Figure 20: Motivation example CDFG

The implementation of the SDFG shown in Figure 20 requires two adders (A1 and A2) and one multiplier M1. Having only one addition operation in control step s3 and two available adders, so o4 can be bound to either A1 or A2. On the other hand, we can see that the two addition operations o2 and o4 have one common input V4. If we assign o4 to the second adder A2, both input operands of A2 (V4 and V7) will have to change at s3. Whereas, if we assign o4 to the first adder A1, in other words, the two addition operations o2 and o4, that share one input operand, are assigned to the same adder A1, only one input operand at A1 will change between the two controls steps s2 and s3 which is V3 to V7. In this case, assigning o4 to adder A1 is power efficient and reduces the total switching activity by one by removing the useless switching at adder A2.

Note that we have tried this example using the developed synthesis tool, the results were as follows:







----------The execution time is eq ıal to: 133686 ms ----------

The initial percentage o f Spurious Switching Activities is: 60.000 %.

The final percentage of Spurious S witching A ctivities is: 55.556 %.

The percentage of improvement in the Spurio ıs Switchin ɜ Activities is: 4.444 %.

----------Statistics----------

The total percentage of improvement in the S urious Switching Activities is: 16.667 %.

Initial Spurious Switching Activities = 6 and Final Spurious Switching Activities = 5 and Total Spurious Switching Activities = 11.

Initial Switching Activities = 10 and Final Switching Activities = 9 and Total Switching Activities = 19

The total percentage of improvement in the total Switching Activities is: 10.000 %.

Analysis of the results:

An analysis of the CDFG shown in Figure 3, after scheduling, binding, and allocation is done.

Functional unit	Bound operations	Control Step
Multiplier 1	o1 = AxB o4 = ExF	Step 1 Step 2
Multiplier 2	o2 = CxD	Step 1
Adder 1	o3 = o1 + o2 G = o3 + o4	Step 2 Step 3

Variable Start and End Times

Variable	Start Time	End Time
A	0	1
B	0	1
C	0	1
D	0	1
E	0	2
F	0	2
o1	1	2
o2	1	2
o3	2	3
o4	2	3
G	3	4

If we analyze the switching activities of the functional units, we find the following:

Functional Unit	Switching couples	Is Spurious	Step
Multiplier 1	A:B E:F	No No	1 2
Multiplier 2	C:D No switching	No No	1 2 & 3
Adder 1	A:B o1:o2 o3:o4 o3:G	Yes No No Yes	1 2 3 Last

A total of 7 switching activities; among which, 2 switching activities are spurious.

After applying the optimized functional unit binding, operation o1 is assigned to the second multiplier, and operations o2 and o4 are assigned to the first multiplier. The optimized multiplexers' distribution saves one 4-1 multiplexer and replaces it by a 3-1 multiplexer as shown in Figure 21.

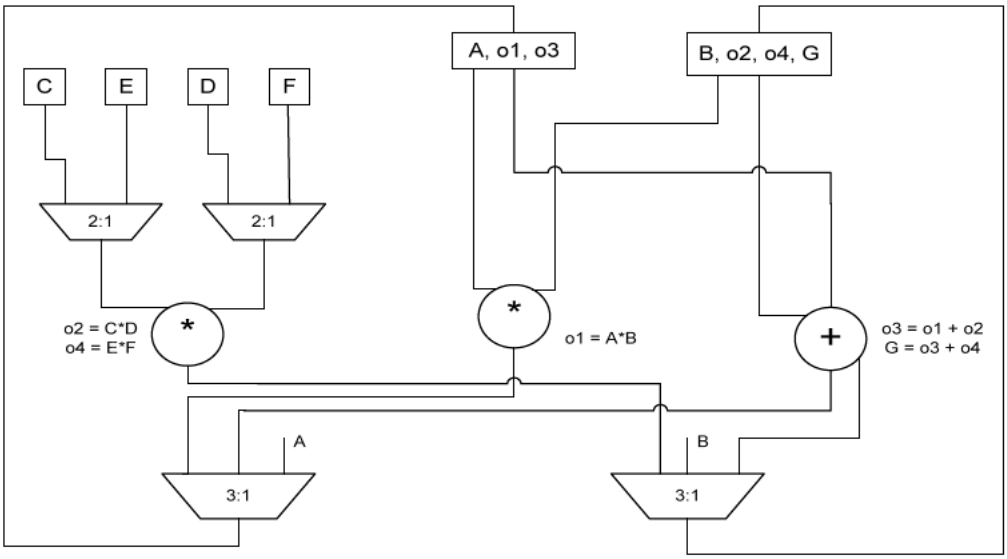


Figure 21: Optimized overall multiplexers' distribution

Functional Unit	Switching couples	Is Spurious	Step
Multiplier 1	C:D	No	1
	E:F	No	2
Multiplier 2	A:B	No	1
	o1:o2	Yes	2
	o3:o4	Yes	3
	o3:G	Yes	Last
Adder 1	A:B	Yes	1
	o1:o2	No	2
	o3:o4	No	3
	o3:G	Yes	Last

A total of 10 switching activities; among which, 5 switching activities are spurious.

Detailed Differential Equation Example:

- Resource Bag: DI7.txt (2 +;4 *;2 -;1 >;)
- Schedule: (List Scheduling)

Table 1: Schedule

v1 v2 o1	*	**0**
v3 v4 o2	*	**0**
v5 v6 o3	*	**0**
v1 v2 o4	*	**0**
v2 v4 o5	+	**0**
o1 o2 o6	*	**1**
v2 o3 o7	*	**1**
v6 o4 o8	+	**1**
v7 o5 o9	>	**1**
o6 v1 o10	-	**2**
o10 o7 o11	-	**3**

The number of control steps needed is: 4

- Register Binding: (Clique Partitioning)

Table 2: Register Binding

Register 1: v1
Register 2: v6
Register 3: o4
Register 4: o5
Register 5: v7
Register 6: v2,o6,o10,o11
Register 7: o1,v3,o7
Register 8: o2,v4,o8
Register 9: o3,v5,o9
Number of Registers needed: 9

Initial & Optimized Functional Unit Binding:

Initial Functional Unit Binding	**Optimized Functional Unit Binding**
The available resources are as follows:	The available resources are as follows:
There are: 2 +	There are: 2 +
There are: 4 *	There are: 4 *
There are: 2 -	There are: 2 -
There are: 1 >	There are: 1 >
FU # 1 is: A1 and is of type: +	FU # 1 is: A1 and is of type: +
The nodes bound to this FU are:	No nodes are bound to this FU!
v2 v4 o5 + CStep: 0	
	FU # 2 is: A2 and is of type: +
FU # 2 is: A2 and is of type: +	The nodes bound to this FU are:
The nodes bound to this FU are:	v2 v4 o5 + CStep: 0
v6 o4 o8 + CStep: 1	v6 o4 o8 + CStep: 1
FU # 3 is: M1 and is of type: *	FU # 3 is: M1 and is of type: *
The nodes bound to this FU are:	The nodes bound to this FU are:
v1 v2 o1 * CStep: 0	v1 v2 o1 * CStep: 0
o1 o2 o6 * CStep: 1	o1 o2 o6 * CStep: 1
FU # 4 is: M2 and is of type: *	FU # 4 is: M2 and is of type: *
The nodes bound to this FU are:	The nodes bound to this FU are:
v3 v4 o2 * CStep: 0	v1 v2 o4 * CStep: 0
v2 o3 o7 * CStep: 1	v2 o3 o7 * CStep: 1
FU # 5 is: M3 and is of type: *	FU # 5 is: M3 and is of type: *
The nodes bound to this FU are:	The nodes bound to this FU are:
v5 v6 o3 * CStep: 0	v3 v4 o2 * CStep: 0
FU # 6 is: M4 and is of type: *	FU # 6 is: M4 and is of type: *
The nodes bound to this FU are:	The nodes bound to this FU are:
v1 v2 o4 * CStep: 0	v5 v6 o3 * CStep: 0
FU # 7 is: S1 and is of type: -	FU # 7 is: S1 and is of type: -
The nodes bound to this FU are:	No nodes are bound to this FU!
o6 v1 o10 - CStep: 2	
	FU # 8 is: S2 and is of type: -
FU # 8 is: S2 and is of type: -	The nodes bound to this FU are:
The nodes bound to this FU are:	o6 v1 o10 - CStep: 2

o10 o7 o11 - CStep: 3 **FU # 9 is: C1 and is of type: >** **The nodes bound to this FU are:** **v7 o5 o9 > CStep: 1**	o10 o7 o11 - CStep: 3 FU # 9 is: C1 and is of type: > The nodes bound to this FU are: v7 o5 o9 > CStep: 1

Switching Couples:

The below table will list all the switching couples. Beside each pair of variables being switched, a flag is added to tell whether this switch is spurious or not. Being non-spurious means that this switch was needed to execute a certain operation at the inputs of the functional unit.

Initial Switching Couples	**Final Switching Couples**
The Initial Switching Couples are: **At Adder A1 (5 couples): 4 S & 1 NS** **v2:v4(NS)** **v2:o2(S)** **o6:o8(S)** **o10:o8(S)** **o11:o8(S)** **At Adder A2 (2 couples): 1 S & 1 NS** **v6:junk(S)** **v6:o4(NS)** **At Multiplier M1 (4 couples): 2 S & 2 NS** **v1:v2(NS)** **v3:v4(S)** **o1:o2(NS)** **o7:o8(S)** **At Multiplier M2 (6 couples): 4 S & 2 NS** **v3:v4(NS)** **v2:v5(S)** **v2:o3(NS)**	The Final Switching Couples are: At Adder A2 (2 couples): **2 NS** v2:v4(NS) v6:o4(NS) At Multiplier M1 (4 couples): **2 S & 2 NS** v1:v2(NS) v3:v4(S) o1:o2(NS) o7:o8(S) At Multiplier M2 (6 couples): **4 S & 2 NS** v1:v2(NS) v2:v5(S) v2:o3(NS) o6:o9(S) o10:o9(S) o11:o9(S) At Multiplier M3 (3 couples): **2 S & 1 NS** v3:v4(NS) o1:o2(S) o7:o8(S)

o6:o9(S) **o10:o9(S)** **o11:o9(S)** **At Multiplier M3 (3 couples): 2 S & 1 NS** **v5:v6(NS)** **o3:v6(S)** **o9:v6(S)** **At Multiplier M4 (4 couples): 3 S & 1 NS** **v1:v2(NS)** **v1:o6(S)** **v1:o10(S)** **v1:o11(S)** **At Subtractor S1 (4 couples): 3 S & 1 NS** **v2:v1(S)** **o6:v1(NS)** **o10:v1(S)** **o11:v1(S)** **At Subtractor S2 (5 couples): 4 S & 1 NS** **v2:v3(S)** **v2:o1(S)** **o6:o7(S)** **o10:o7(NS)** **o11:o7(S)** **At Comparator C1 (2 couples): 1 S & 1 NS** **junk:junk(S)** **v7:o5(NS)**	At Multiplier M4 (3 couples): **2 S & 1 NS** v5:v6(NS) o3:v6(S) o9:v6(S) At Subtractor S2 (5 couples): **3 S & 2 NS** v2:v1(S) o6:v1(NS) o6:o7(S) o10:o7(NS) o11:o7(S) At Comparator C1 (2 couples): **1 S & 1 NS** junk:junk(S) v7:o5(NS)
Total of 35 Switching Activities **Total of 24 Spurious Switching Activities**	Total of 25 Switching Activities Total of 14 Spurious Switching Activities
A total of 41.66 % improvement in the number of the spurious switching activities	

Analysis of switching activities of A1 & A2:

By simply moving operation o5 = v2 + v4 from A1 to A2, we saved 4 spurious switching activities occurring at the inputs of A1. We also saved one spurious switching activity at the inputs of A2 where v6 will switch with junk at step 0 until o4 is ready at step1. By moving o5 to A2, this operation will result in a useful switching at the inputs of A2 to compute o5 at step0 and then another switching to compute o8 = v6 + o4 at step1.

Analysis of the results:

According to the above results, we realized that the dynamic power of a design can be reduced by switching the operations assigned to functional units in order to reduce the spurious switching activities occurring at the inputs of the functional units. This improvement has some overhead in the time execution of the design; this overhead is achieved by the simulated annealing that runs in order to search for the optimal solution in terms of switching activities. No overhead in the area of the design is enforced.

4.4 Temperature Awareness

Circuit performance is greatly affected by a high working temperature for it affects the switching speed of the transistor through the carrier mobility. In addition to performance degradation, very high temperature also affects circuit reliability. A region with excessive heat is referred to as a hot spot. Hot spots put at risk the correct execution of the design by simply causing temporary as well as permanent faults which lead to incorrect operations [18]. In addition to that, a circuit may be identified as faulty when it is tested at a very high temperature, but it is good at normal working temperature ➔ this leads to higher yield loss.

The fact that temperature has an additive nature is what differentiates between switching power and temperature is that temperature; in other words, if we want to measure the temperature of the design at a certain point in time, we should consider the entire history of activity because temperature depends on the entire history of activity in the past and not on the current state only.

Since the decisions, at any of the three high-level synthesis tasks: scheduling, allocation, and binding, taken during the high-level synthesis flow will greatly impact the activity of the available functional resources, temperature increase can be effectively controlled by performing temperature-aware scheduling, resource allocation (functional units and registers), and binding tasks. Failing to do so, heat dissipated in the design will get accumulated to a point where it exceeds the maximum limit and might cause undesired hot spots, as explained earlier. Having an additive nature, it is understandable to have a high amount of heat dissipated even though the average activity of the resources in the design seems to be well bounded [18].

The solution to this problem is a functional unit binding algorithm that will minimize the maximum temperature reached by all the resources in the design while following the resource constraint given in the selected resource bag. So after applying the normal scheduling and functional unit binding techniques to the graph, the algorithm is executed trying to optimize the functional unit binding decisions in order to minimize the temperature of the available resources. Note that the resource bag is predefined and no increase in the number of resources is acceptable.

- Or maybe, while implementing the functional unit binding to bind operations to modules, we can use weighted edges between operations where the weight will be the number of switching activities if these two operations were bound to the same resource consecutively, and decide on the distribution of operations over the available resources based on this criterion.

By analyzing the impact of a temperature-aware functional unit binding technique onto the power consumption of a design, we learnt that, controlling the maximum temperature reached by the available resources in the design comes at the cost of some power overhead sometimes. Whenever we try to optimize the temperature, there might be an overhead on power. For example, the "RC-TEMP-MIN" binding technique presented in [18] tries to minimize the maximum temperature by using a larger number of resources. By splitting the operations on more resources, the maximum temperature reached by every resource becomes smaller because it will have more time to cool down between an operation and its successor. But on the other hand, using a larger number of resources causes an increase in the total power consumption of the design because more

resources are being used, hence more resources are consuming power and henceforth the summation of the power consumed by every resource used becomes larger.

If hot spot prevention is of highest priority due to the severe reliability requirements of the design, a temperature-aware binding scheme would be used. This depends on the tolerable power overhead.

4.4.1 Temperature Aware Functional Unit Binding

A temperature aware functional unit binding was developed in order to reduce the temperature of a certain design. The main idea of this optimized algorithm is to reduce the temperature consumed by every resource used in the design, and hence reduce the total temperature in order to avoid hot spots.

This optimized will run after the whole synthesis flow is done. The same concept used for the optimization of power, is applied here. Simulated annealing is used to try to get the optimal temperature of the design by swapping the operations bound to functional units to other possible available functional units of same type when applicable. The possible swaps are collected following the same criteria used for the power as explained in section 4.3.1 and follow the same restrictions as in section 4.3.2.2.

4.4.1.1 Motivations

The temperature of a module is associated with its activity. In other words, as the number of tasks assigned to a module increases, that same module is more active and hence generates more heat and subsequently the temperature of the module is more likely to increase leading to a higher temperature of the whole design [18].

In addition to what was previously said, the relative timing or control steps of the operations that are executing on a particular module are important and has an impact its temperature. This is affected by the schedule of the DFG following the resource

bag constraint and the result of the functional unit binding task that will assign the operations scheduled at a certain control step to the available resources. Let us consider the following two scenarios:

1- In a certain interval of time, a module M1 executes a set of tasks where the assigned tasks are very close (in time) to each other. In this case, the rise in temperature is expected to be steep, since the module has less opportunity to relieve any heat buildup during execution.
2- M1 executes the same set of tasks across the same period of time, but where there are more inactive time intervals in between the executed operations. In this case, the temperature is expected to be lower than in case 1 because the module has more time to cool down between any operation and its successor.

The solution to this problem is by having a temperature-aware assignment of tasks to the available modules. This solution should take into consideration the heat buildup that is affected by continuous activity of the module and will help control the rise in temperature.

According to the conclusion regarding the correlation between the switching activity of a resource and its temperature, one could argue that rise in temperature can be bound by minimizing total switching activity ⇒ minimizing switching activity of the functional units helps keep the total temperature low.

4.4.1.2 Algorithm Description

The purpose of the algorithm is to evenly distribute the operations bound to functional units over the lifetime of the functional unit.

The lifetime of the functional unit is the duration in which the functional unit is active, which means, from the first occurring switching couple at the inputs of the functional unit to the last switching couple.

The activity of the functional unit is the total count of switching occurring at its inputs. The total count is the sum of spurious and non-spurious switching activity.

The temperature of a functional unit is a function of its lifetime and its activity. In order to compute the temperature of a functional unit, which will be the main variable in the cost function of the simulated annealing, a summation over all the dissipated temperature needed for each switching couple at the inputs of the functional unit.

Optimal Distance:

This is the optimal distance (in terms of control steps) that should separate two consecutive activities on a certain functional unit. This distance gives time for the functional unit to cool down after each operation in order to avoid any excess in the heat dissipated. The objective is to try to get as close as possible to this optimal distance. Knowing that a small distance between operations will not give time to a functional unit to cool down, same as a far distance, for it will lead to small distance for the remaining operations.

In order to compute this optimal distance, we compute the lifetime of the functional unit, and its total activity, and then we divide them:

OptimalX_{FU} = lifetime/activity.

Cost:

In order to compute the cost, which is the temperature of the functional unit, a parabola is used. This is due to the above explanation where the optimal distance will be the

vertex of the parabola. And very small distances have high weight, same as very far distances.

Get the equation of a parabolic curve having the minimum value $y = a(x - m)^2 + n$, then the vertex is the point V(m, n) .

For $| a | > 1$ the parabola will be "skinny", because it grows more quickly. Let us consider the case where a = 5, the parabola grows five times as fast.

For $| a | < 1$, the parabola will be "fat", because it grows more slowly.

Because we need a fat curve in this project, we considered the case for a = 1/5 which will lead to a parabola curve which will have the following equation:

$y = 1/5(x - m)^2 + n$ where the vertex has the following coordinates (m, n). As mentioned earlier, the “m” coordinate of the vertex will be equal to the OptimalX. We are left with its “n” coordinate; we selected the value “1” to start with.

So the final equation of the parabola is: $y = 1/5(x - \text{OptimalX})^2 + 1$.

For any given x, we can compute y which will be the weight or the cost for a given computation at a certain functional unit distant a distance x from the previous computation.

For the special case where the optimal x, which the vertex x, is 2 and the minimum cost is 1, the graph of the parabola will be shown below:

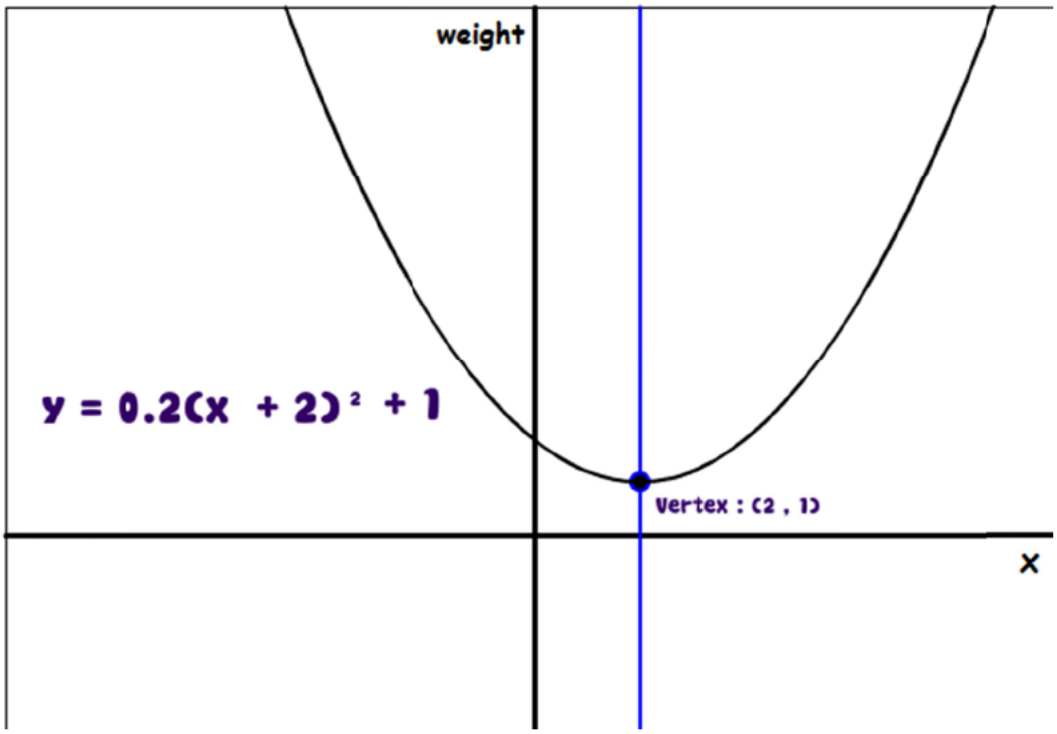


Figure 22: Graph of the parabola

Let us consider those two cases:

Case 1: a functional unit FU_1 is computing 10 computations during a period of 20 control steps following this distribution:

Case 2: another functional unit FU_2 is computing 10 computations during a period of 20 control steps following this distribution:

In both cases, the two functional units, FU_1 & FU_2, consume the same amount of power in terms of switching activity because both of them have a "10" as total activity. The difference will be in the dissipated temperature. It is obvious that the first functional unit consumes much more temperature than the second because the first 9 operations are very close to each other, and the last one is very far.

On the contrary, the second functional unit follows the optimal distribution where the distance is optimal which is in this case: 20/10 = 2 control steps between any two consecutive operations. The role of the temperature aware functional unit binding is to redistribute the operations bound to the functional units, without breaking any constraint, in a way to have the most normal distribution of the distance between any two switching activities.

The more we can decrease the number of switching activities, the less the temperature reached by the available resources. (According to Reason 2 above)

We will try to separate the operations bound to the same functional unit and leave a time interval between the executed tasks in order to give a bigger opportunity to the module to relieve any heat buildup during execution. (According to Reason 1 above)

Pseudo code:

- After the functional unit binding is completed, we compute the total number of switching activities on each resource (spurious and non-spurious).
- Compute the lifetime of each functional unit.
- Compute the optimal distance for each functional unit by dividing the lifetime over the total activity.
- Compute the total cost (temperature) of the current resource binding by adding the temperature of each functional unit.
    - To get the temperature of each functional unit, we get all the switching couples at its input, loop over them and add the cost for each switching following the parabola equation as explained above, with the vertex coordinates as calculated in optimal distance:

(optimalX, 1) and the x in the equation will be the distance between any two consecutive switching.

- Compute all the possible swaps of operations.
- Run the simulated annealing to try to swap any two possible operations over the available resources and try to get the optimal solution in terms of temperature.

Let us consider the following simple example in order to elaborate more in the work in this algorithm. Consider the CDFG shown in Figure 23.

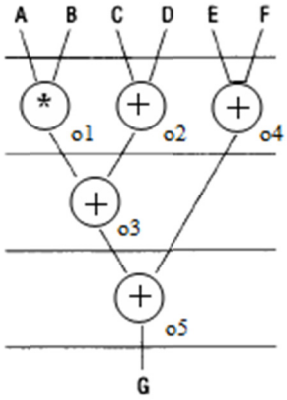


Figure 23: Temperature awareness DFG

Scheduling this DFG shown above using a resource bag containing 2 adders (A1, A2) and one multiplier (M1), the schedule is constituted of 3 control steps and shown in Figure 24.

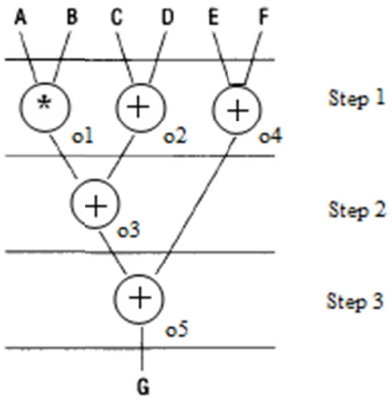


Figure 24: Schedule of the Temperature aware DFG

aving two adders and the sched ıle requires two adders, so we can conclude that the o ɔerations bound to the available a dders can have differe ıt distributions as follows:

Option 1	o2, o3, o5 are bound to A1 and o4 to A2
Option 2	o2: A1 o3: A1 o4: A2 o5: A2
Option 3	o2: A2 o3: A2 o4: A1 o5: A1
Option 4	o2: A1 o3: A2 o5: A1 o4: A1

The above 4 propose l options are all cor rect, but dissipate different am ɔunts of t mperatures. In option 1, one of the adders is used to compute the three sim ltaneous a ldditions; this will result in the hi ghest amount of temperature. In options 2 a ıd 3, one a lder is used to comp ıte two simultaneous additions, and the se ond one is used to c ɔmpute the other two. In terms o ` temperatu re, each of the options dissipates a certain a nount of temperature which is ılso depen lent on the register bi nding result which a fects the switching ac ivities at t e inputs of the functio ıal units.

Using numbers, we can reduce the temperature from 10 to 7 by simply using the optimal binding as option 4:

Initial Functional Unit Binding	**Final Functional Unit Binding**
The available resources are as follows: **There are: 2 +** **There are: 1 *** **FU # 1 is: A1 and is of type: +** **The nodes bound to this FU are:** **C D o2 + CStep: 0** **o1 o2 o3 + CStep: 1** **FU # 2 is: A2 and is of type: +** **The nodes bound to this FU are:** **E F o4 + CStep: 0** **o3 o4 G + CStep: 2** **FU # 3 is: M1 and is of type: *** **The nodes bound to this FU are:** **A B o1 * CStep: 0**	The available resources are as follows: There are: 2 + There are: 1 * FU # 1 is: A1 and is of type: + The nodes bound to this FU are: C D o2 + CStep: 0 o1 o2 o3 + CStep: 1 o3 o4 G + CStep: 2 FU # 2 is: A2 and is of type: + The nodes bound to this FU are: E F o4 + CStep: 0 FU # 3 is: M1 and is of type: * The nodes bound to this FU are: A B o1 * CStep: 0

Initial Switching Couples	Final Switching Couples
The Initial Switching Couples are: **At Adder A1 (4 couples):** **C:D(NS)** **o1:o2(NS)** **o3:o2(S)** **G:o2(S)** **At Adder A2 (4 couples):** **E:F(NS)** **o1:o4(S)** **o3:o4(NS)** **G:o4(S)** **At Multiplier M1 (2 couples):** **A:B(NS)** **o2:o4(S)**	The Initial Switching Couples are: At Adder A1 (4 couples): C:D(NS) o1:o2(NS) o3:o4(NS) G:o4(S) At Adder A2 (1 couples): E:F(NS) At Multiplier M1 (2 couples): A:B(NS) o2:o4(S)

Initial Temperature	Final Temperature
The Initial Functional Units Temperatures are: **Temperature of the FU: A1 is:4.0.** **Temperature of the FU: A2 is:4.0.**	The Final Functional Units Temperatures are: Temperature of the FU: A1 is:4.0. Temperature of the FU: A2 is:1.0.

Temperature of the FU: M1 is:2.0	Temperature of the FU: M1 is:2.0.
Improvement	
The initial total temperature is: 10.0 . **The final total temperature is: 7.0 .** **The percentage of improvement in the temperature of FUs is: 30 % .**	

In the above case, adder A1 is computed 3 consecutive operations, but this didn't increase its temperature because in the initial binding, A1 was computing two operations, but its inputs were switching 4 times, 2 of which were spurious. In the optimized binding, A1's activity is still the same with 4 switching, but only one of which is spurious. This means that the temperature at A1 is better now because it is computing only one spurious activity instead of two. On the other hand, A2 was computing also two operations, but having 4 switching couples at its inputs, two of those switching were spurious. Now, A2 is computing only one operation, o4, and only switching once because the inputs of o4, which are E & F, happen to be placed in separate registers that contain no other variables.

It is clear now, how were able to reduce the total temperature consumed in the design without increasing the temperature consumed by every functional unit.

After running this algorithm on 10 benchmarks, the results give a maximum of 44.4% improvement and an average of 9.3% reduction in the total temperature of the design.

CHAPTER FIVE
EXPERIMENTAL RESULTS

5.1 Power Results

In order to validate the effectiveness and correctness of the developed approaches, 11 benchmarks of various complexities were synthesized; power optimized, and finally temperature optimized. The power related results have shown that an average of 8.7% reduction in the total power was achieved in terms of switching activity and an average of 15.4% was achieved in terms spurious switching activity.

In this chapter, we will provide the simulation results for all the benchmarks showing initial switching activity with no power management and annealed results with power management. A comparison is also conducted between the results got by applying the method developed by [1] alone, or applying the work implemented in this project alone, or applying both methodologies simultaneously.

A wide selection of resource bags was adopted. For a wise resource bag selection, an ASAP schedule was conducted on each benchmark, in order to find the maximum needed resources to schedule the CDFG with its fastest schedule. After getting the maximum required number of each resource, a set of resource bag was created starting from a resource bag having a quantity of 1 from each resource type, to a resource bag having the maximum quantity that was got from the ASAP schedule. The jump in the number of resources of each type was proportional to the maximum number. If the maximum number is not very big, an increment of 1 resource for each type is used to create the next resource bag. Sometimes a jump of 2 is used. Other times, a jump of 4 in each type of resources is done. The average number of resource bag is 20.

A batch file was created in order to loop over all the benchmarks and all the resources for each benchmark, and the results were written to a file.

Note that the execution time needed to get the final results was also saved. Knowing that the execution time and the final results are widely affected by the simulated annealing parameters, in terms of cooling rate, the number of iterations, the trial and temperature factor, etc… some changes in those parameters might give better results.

The below table shows the outcome of the ASAP scheduling algorithm for all the available 11 benchmarks in order to get the upper bound on the needed functional units:

Table 3: ASAP Schedules & Resource Bags

Benchmark	ASAP	Resource Bag
4pDCT (15 nodes)	4 control steps	(4 *; 2 +; 2 -)
AR (28 nodes)	8 control steps	(8 *; 4 +)
DCT1 (35 nodes)	6 control steps	(16 *; 8 +)
DCT2 (70 nodes)	6 control steps	(32 *; 16 +)
DIFFEQ2 (11 nodes)	4 control steps	(1 +; 4 *; 1 -; 1 >;)
DIFFEQ3 (16 nodes)	5 control steps	(2 +; 6 *; 1 -; 1 >;)
DIFFEQ4 (17 nodes)	5 control steps	(3 +; 6 *; 2 -;)
EWF (34 nodes)	14 control steps	(2 *; 4 +)
FDCT (42 nodes)	6 control steps	(8 *; 4 +; 4 -)
FIR (13 ndoes)	7 control steps	(7 *; 1 +)
IIR (14 nodes)	8 control steps	(6 *; 2 +)
LMS (17 nodes)	7 control steps	(4 *; 4 +)
wavelet (16 nodes)	6 control steps	(5 *; 2 +; 1 -)
nestor2	6 control steps	(8 *; 4 +; 4 -)

The below set of tables show the results of the optimized power using the technique presented in this work as PAFUB (Power Aware Functional Unit Binding), the technique developed in [1] as PARB (Power Aware Register Binding), and finally the combination of both techniques running simultaneously one after the other. The labels shown in the tables are explained as follows: %SA is the percentage of improvement in terms of switching activities, and the %SSA is the percentage of improvement in terms

of spurious switching activities, %TArea is the percentage of improvement in the total area and finally %MArea is the percentage of improvement in the area of muxes.

Benchmark: 4pDCT (15 nodes)

Table 4: PAFUB - 4pDCT

PAFUB				
Resource Bag	%SA	%SSA	% TArea	% MArea
4D2.txt (2 *;1 +;1 -;)	0.000	0.000	0.000	0.000
4D3.txt (3 *;1 +;1 -;)	0.000	0.000	0.000	0.000
4D4.txt (4 *;1 +;1 -;)	0.000	0.000	0.000	0.000
4D5.txt (1 *;2 +;1 -;)	12.500	33.333	1.790	3.316
4D6.txt (2 *;2 +;1 -;)	4.348	12.500	2.580	6.897
4D7.txt (3 *;2 +;1 -;)	10.000	20.000	2.034	7.407
4D8.txt (4 *;2 +;1 -;)	6.061	11.111	3.302	14.815
4D9.txt (1 *;1 +;2 -;)	10.345	21.429	0.000	0.000
4D10.txt (2 *;1 +;2 -;)	4.348	12.500	0.000	0.000
4D11.txt (3 *;1 +;2 -;)	7.407	16.667	0.000	0.000
4D12.txt (4 *;1 +;2 -;)	9.677	18.750	0.000	0.000
4D13.txt (1 *;2 +;2 -;)	21.212	38.889	5.572	11.047
4D14.txt (2 *;2 +;2 -;)	0.000	0.000	0.000	0.000
4D15.txt (3 *;2 +;2 -;)	9.375	17.647	1.007	3.704
4D16.txt (4 *;2 +;2 -;)	11.765	21.053	0.951	4.865
Average	**7.136**	**14.925**	**1.149**	**3.470**

Benchmark: 4pDCT (15 nodes)

Table 5: PARB - 4pDCT

PARB				
Resource Bag	**%SA**	**%SSA**	**% TArea**	**% MArea**
4D2.txt (2 *;1 +;1 -;)	20.000	50.000	-5.365	-14.815
4D3.txt (3 *;1 +;1 -;)	20.690	42.857	-3.179	-12.500
4D4.txt (4 *;1 +;1 -;)	21.212	38.889	-4.303	-21.739
4D5.txt (1 *;2 +;1 -;)	8.333	22.222	0.056	0.104
4D6.txt (2 *;2 +;1 -;)	8.696	25.000	-6.610	-17.672
4D7.txt (3 *;2 +;1 -;)	23.333	46.667	-3.051	-11.111
4D8.txt (4 *;2 +;1 -;)	24.242	44.444	-0.825	-3.704
4D9.txt (1 *;1 +;2 -;)	31.034	64.286	-3.960	-7.710
4D10.txt (2 *;1 +;2 -;)	13.043	37.500	-2.648	-7.407
4D11.txt (3 *;1 +;2 -;)	14.815	33.333	0.986	3.320
4D12.txt (4 *;1 +;2 -;)	12.903	25.000	-0.103	-0.463
4D13.txt (1 *;2 +;2 -;)	27.273	50.000	-3.656	-7.250
4D14.txt (2 *;2 +;2 -;)	25.000	53.846	-2.649	-7.692
4D15.txt (3 *;2 +;2 -;)	18.750	35.294	-2.140	-7.870

4D16.txt (4 *;2 +;2 -;)	20.588	36.842	-3.383	-17.297
Average	**19.328**	**40.412**	**-2.722**	**-8.921**

Benchmark: 4pDCT (15 nodes)

Table 6: PAFUB&PARB - 4pDCT

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
4D2.txt (2 *;1 +;1 -;)	20.000	50.000	20.000	50.000
4D3.txt (3 *;1 +;1 -;)	20.690	42.857	20.690	42.857
4D4.txt (4 *;1 +;1 -;)	21.212	38.889	21.212	38.889
4D5.txt (1 *;2 +;1 -;)	16.667	44.444	16.667	44.444
4D6.txt (2 *;2 +;1 -;)	8.696	25.000	13.043	37.500
4D7.txt (3 *;2 +;1 -;)	26.667	53.333	26.667	53.333
4D8.txt (4 *;2 +;1 -;)	27.273	50.000	24.242	44.444
4D9.txt (1 *;1 +;2 -;)	34.483	71.429	31.034	64.286
4D10.txt (2 *;1 +;2 -;)	13.043	37.500	13.043	37.500
4D11.txt (3 *;1 +;2 -;)	14.815	33.333	14.815	33.333
4D12.txt (4 *;1 +;2 -;)	19.355	37.500	16.129	31.250
4D13.txt (1 *;2 +;2 -;)	42.424	77.778	42.424	77.778
4D14.txt (2 *;2 +;2 -;)	25.000	53.846	25.000	53.846
4D15.txt (3 *;2 +;2 -;)	21.875	41.176	21.875	41.176
4D16.txt (4 *;2 +;2 -;)	20.588	36.842	23.529	42.105
Average	**22.186**	**46.262**	**22.025**	**46.183**

Benchmark: AR (28 nodes)

Table 7: PAFUB - AR

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
A2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
A3.txt (1 +;3 *;)	2.174	5.556	-0.809	-2.260
A4.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
A5.txt (1 +;5 *;)	0.000	0.000	0.000	0.000
A6.txt (1 +;6 *;)	0.000	0.000	0.000	0.000
A7.txt (1 +;7 *;)	1.563	2.778	-0.515	-2.611
A8.txt (1 +;8 *;)	0.000	0.000	0.000	0.000
A9.txt (2 +;1 *;)	12.000	23.077	-0.100	-0.167
A10.txt (2 +;2 *;)	2.128	5.263	-1.059	-2.135
A11.txt (2 +;3 *;)	2.000	4.545	1.646	4.307

A12.txt (2 +;4 *;)	0.000	0.000	0.000	0.000
A13.txt (2 +;5 *;)	3.030	5.263	2.515	8.916
A14.txt (2 +;6 *;)	1.408	2.326	1.279	5.063
A15.txt (2 +;7 *;)	1.370	2.222	-0.060	-0.273
A16.txt (2 +;8 *;)	0.000	0.000	0.000	0.000
A17.txt (3 +;1 *;)	12.281	21.212	2.799	4.542
A18.txt (3 +;2 *;)	3.922	8.696	2.243	4.531
A19.txt (3 +;3 *;)	6.250	8.333	3.690	10.068
A20.txt (3 +;4 *;)	2.941	5.000	2.827	8.449
A21.txt (3 +;5 *;)	4.412	7.500	3.900	13.710
A22.txt (3 +;6 *;)	1.282	2.000	0.523	2.135
A23.txt (3 +;7 *;)	2.410	3.636	-1.442	-6.598
A24.txt (3 +;8 *;)	6.667	9.677	1.726	8.637
A25.txt (4 +;1 *;)	19.118	29.545	5.517	9.084
A26.txt (4 +;2 *;)	3.390	6.452	2.085	4.307
A27.txt (4 +;3 *;)	8.451	11.628	-1.854	-5.453
A28.txt (4 +;4 *;)	5.405	8.696	2.806	8.570
A29.txt (4 +;5 *;)	4.000	6.383	1.900	6.720
A30.txt (4 +;6 *;)	0.000	0.000	0.000	0.000
A31.txt (4 +;7 *;)	3.333	4.839	0.912	4.198
A32.txt (4 +;8 *;)	4.950	6.849	-2.665	-14.402
Average	**3.693**	**6.177**	**0.899**	**2.237**

Benchmark: AR (28 nodes)

Table 8: PARB - AR

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
A2.txt (1 +;2 *;)	2.500	8.333	4.636	9.813
A3.txt (1 +;3 *;)	4.348	11.111	1.040	2.906
A4.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
A5.txt (1 +;5 *;)	3.509	6.897	1.274	4.988
A6.txt (1 +;6 *;)	3.226	5.882	0.575	2.476
A7.txt (1 +;7 *;)	1.563	2.778	-1.015	-5.142
A8.txt (1 +;8 *;)	0.000	0.000	0.000	0.000
A9.txt (2 +;1 *;)	8.000	15.385	-1.606	-2.664
A10.txt (2 +;2 *;)	6.383	15.789	2.416	4.870
A11.txt (2 +;3 *;)	0.000	0.000	0.000	0.000
A12.txt (2 +;4 *;)	5.263	10.345	1.452	4.351
A13.txt (2 +;5 *;)	4.545	7.895	-0.135	-0.480
A14.txt (2 +;6 *;)	2.817	4.651	0.538	2.132

A15.txt (2 +;7 *;)	1.370	2.222	0.000	0.000
A16.txt (2 +;8 *;)	0.000	0.000	0.000	0.000
A17.txt (3 +;1 *;)	8.772	15.152	0.047	0.076
A18.txt (3 +;2 *;)	7.843	17.391	2.146	4.334
A19.txt (3 +;3 *;)	1.563	2.778	0.888	2.422
A20.txt (3 +;4 *;)	4.412	7.500	0.685	2.046
A21.txt (3 +;5 *;)	2.941	5.000	2.466	8.669
A22.txt (3 +;6 *;)	3.846	6.000	-2.869	-11.708
A23.txt (3 +;7 *;)	2.410	3.636	0.481	2.199
A24.txt (3 +;8 *;)	4.444	6.452	0.000	0.000
A25.txt (4 +;1 *;)	4.412	6.818	0.046	0.076
A26.txt (4 +;2 *;)	5.085	9.677	5.375	11.104
A27.txt (4 +;3 *;)	2.817	4.651	-1.825	-5.369
A28.txt (4 +;4 *;)	2.703	4.348	2.077	6.343
A29.txt (4 +;5 *;)	2.667	4.255	1.216	4.301
A30.txt (4 +;6 *;)	1.205	1.818	-1.812	-8.140
A31.txt (4 +;7 *;)	3.333	4.839	1.451	6.676
A32.txt (4 +;8 *;)	3.960	5.479	-0.437	-2.363
Average	**3.417**	**6.358**	**0.616**	**1.417**

Benchmark: AR (28 nodes)

Table 9: PAFUB&PARB - AR

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
A2.txt (1 +;2 *;)	2.500	8.333	2.500	8.333
A3.txt (1 +;3 *;)	4.348	11.111	4.348	11.111
A4.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
A5.txt (1 +;5 *;)	3.509	6.897	3.509	6.897
A6.txt (1 +;6 *;)	3.226	5.882	3.226	5.882
A7.txt (1 +;7 *;)	1.563	2.778	1.563	2.778
A8.txt (1 +;8 *;)	0.000	0.000	0.000	0.000
A9.txt (2 +;1 *;)	18.000	34.615	12.000	23.077
A10.txt (2 +;2 *;)	6.383	15.789	6.383	15.789
A11.txt (2 +;3 *;)	2.000	4.545	2.000	4.545
A12.txt (2 +;4 *;)	5.263	10.345	5.263	10.345
A13.txt (2 +;5 *;)	4.545	7.895	4.545	7.895
A14.txt (2 +;6 *;)	2.817	4.651	2.817	4.651
A15.txt (2 +;7 *;)	1.370	2.222	1.370	2.222
A16.txt (2 +;8 *;)	0.000	0.000	0.000	0.000
A17.txt (3 +;1 *;)	21.053	36.364	14.035	24.242

A18.txt (3 +;2 *;)	7.843	17.391	9.804	21.739
A19.txt (3 +;3 *;)	6.250	8.333	6.250	8.333
A20.txt (3 +;4 *;)	7.353	12.500	7.353	12.500
A21.txt (3 +;5 *;)	5.882	10.000	5.882	10.000
A22.txt (3 +;6 *;)	3.846	6.000	3.846	6.000
A23.txt (3 +;7 *;)	2.410	3.636	2.410	3.636
A24.txt (3 +;8 *;)	6.667	9.677	6.667	9.677
A25.txt (4 +;1 *;)	23.529	36.364	23.529	36.364
A26.txt (4 +;2 *;)	6.780	12.903	5.085	9.677
A27.txt (4 +;3 *;)	9.859	13.953	9.859	13.953
A28.txt (4 +;4 *;)	4.054	6.522	5.405	8.696
A29.txt (4 +;5 *;)	5.333	8.511	5.333	8.511
A30.txt (4 +;6 *;)	1.205	1.818	1.205	1.818
A31.txt (4 +;7 *;)	3.333	4.839	3.333	4.839
A32.txt (4 +;8 *;)	4.950	6.849	4.950	6.849
Average	**5.673**	**10.023**	**5.306**	**9.367**

Benchmark: DCT1 (35 nodes)

Table 10: PAFUB - DCT1

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
D2.txt (1 +;2 *;)	2.128	8.333	-0.115	-0.201
D3.txt (1 +;4 *;)	7.692	23.529	-3.879	-9.839
D4.txt (1 +;6 *;)	12.281	31.818	1.370	4.075
D5.txt (1 +;8 *;)	7.018	18.182	-2.457	-9.882
D6.txt (1 +;10 *;)	8.621	21.739	0.000	0.000
D7.txt (1 +;12 *;)	8.475	20.833	-0.607	-3.663
D8.txt (1 +;14 *;)	12.698	28.571	0.527	3.401
D9.txt (1 +;16 *;)	7.576	16.129	0.476	3.663
D10.txt (2 +;2 *;)	9.259	26.316	-0.031	-0.058
D11.txt (2 +;4 *;)	7.273	20.000	-1.676	-3.770
D12.txt (2 +;6 *;)	5.455	15.000	0.575	1.754
D13.txt (2 +;8 *;)	0.000	0.000	0.000	0.000
D14.txt (2 +;10 *;)	1.493	3.125	0.331	1.391
D15.txt (2 +;12 *;)	13.235	27.273	0.935	5.101
D16.txt (2 +;14 *;)	13.514	25.641	-0.262	-1.644
D17.txt (2 +;16 *;)	0.000	0.000	0.000	0.000
D18.txt (4 +;2 *;)	12.329	23.684	4.508	8.183
D19.txt (4 +;4 *;)	11.940	25.000	3.160	7.391
D20.txt (4 +;6 *;)	17.143	34.286	1.243	3.728

D21.txt (4 +;8 *;)	16.667	30.233	-2.036	-7.576
D22.txt (4 +;10 *;)	17.105	31.707	0.901	4.055
D23.txt (4 +;12 *;)	25.882	44.000	1.268	7.967
D24.txt (4 +;14 *;)	18.182	33.333	-0.298	-2.079
D25.txt (4 +;16 *;)	10.959	21.053	-1.511	-12.000
D26.txt (8 +;2 *;)	4.348	6.250	1.786	3.462
D27.txt (8 +;4 *;)	7.955	13.208	3.256	8.062
D28.txt (8 +;6 *;)	15.730	25.926	0.947	3.009
D29.txt (8 +;8 *;)	11.765	20.000	-3.204	-13.452
D30.txt (8 +;10 *;)	10.870	17.544	0.995	4.415
D31.txt (8 +;12 *;)	22.549	34.328	0.612	3.708
D32.txt (8 +;14 *;)	27.928	40.789	1.564	10.288
D33.txt (8 +;16 *;)	20.183	29.730	-0.523	-4.327
Average	**11.508**	**22.424**	**0.245**	**0.474**

Benchmark: DCT1 (35 nodes)

Table 11: PARB - DCT1

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
D2.txt (1 +;2 *;)	2.128	8.333	2.789	4.882
D3.txt (1 +;4 *;)	3.846	11.765	1.396	3.540
D4.txt (1 +;6 *;)	1.754	4.545	0.000	0.000
D5.txt (1 +;8 *;)	5.263	13.636	0.410	1.647
D6.txt (1 +;10 *;)	1.724	4.348	1.362	6.196
D7.txt (1 +;12 *;)	8.475	20.833	-0.294	-1.774
D8.txt (1 +;14 *;)	6.349	14.286	0.857	5.526
D9.txt (1 +;16 *;)	3.030	6.452	0.000	0.000
D10.txt (2 +;2 *;)	1.852	5.263	-1.016	-1.912
D11.txt (2 +;4 *;)	9.091	25.000	0.745	1.676
D12.txt (2 +;6 *;)	0.000	0.000	0.000	0.000
D13.txt (2 +;8 *;)	4.839	11.111	0.802	3.077
D14.txt (2 +;10 *;)	7.463	15.625	0.000	0.000
D15.txt (2 +;12 *;)	13.235	27.273	0.305	1.667
D16.txt (2 +;14 *;)	13.514	25.641	-0.270	-1.696
D17.txt (2 +;16 *;)	0.000	0.000	0.000	0.000
D18.txt (4 +;2 *;)	1.370	2.632	-0.170	-0.309
D19.txt (4 +;4 *;)	2.985	6.250	5.661	13.242
D20.txt (4 +;6 *;)	1.429	2.857	-0.488	-1.465
D21.txt (4 +;8 *;)	7.692	13.953	2.367	8.809
D22.txt (4 +;10 *;)	1.316	2.439	-0.775	-3.489

D23.txt (4 +;12 *;)	15.294	26.000	-1.505	-9.453
D24.txt (4 +;14 *;)	5.195	9.524	-1.855	-12.933
D25.txt (4 +;16 *;)	0.000	0.000	0.000	0.000
D26.txt (8 +;2 *;)	18.261	26.250	-2.012	-3.901
D27.txt (8 +;4 *;)	4.545	7.547	3.124	7.735
D28.txt (8 +;6 *;)	1.124	1.852	-0.977	-3.103
D29.txt (8 +;8 *;)	8.235	14.000	1.990	8.355
D30.txt (8 +;10 *;)	2.174	3.509	0.667	2.959
D31.txt (8 +;12 *;)	7.843	11.940	-0.853	-5.169
D32.txt (8 +;14 *;)	14.414	21.053	-0.259	-1.706
D33.txt (8 +;16 *;)	10.092	14.865	0.000	0.000
Average	**5.767**	**11.212**	**0.375**	**0.700**

Benchmark: DCT1 (35 nodes)

Table 12: PAFUB&PARB - DCT1

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
D2.txt (1 +;2 *;)	4.255	16.667	4.255	16.667
D3.txt (1 +;4 *;)	7.692	23.529	9.615	29.412
D4.txt (1 +;6 *;)	14.035	36.364	12.281	31.818
D5.txt (1 +;8 *;)	10.526	27.273	8.772	22.727
D6.txt (1 +;10 *;)	10.345	26.087	12.069	30.435
D7.txt (1 +;12 *;)	13.559	33.333	11.864	29.167
D8.txt (1 +;14 *;)	12.698	28.571	12.698	28.571
D9.txt (1 +;16 *;)	7.576	16.129	7.576	16.129
D10.txt (2 +;2 *;)	7.407	21.053	9.259	26.316
D11.txt (2 +;4 *;)	12.727	35.000	7.273	20.000
D12.txt (2 +;6 *;)	1.818	5.000	1.818	5.000
D13.txt (2 +;8 *;)	4.839	11.111	6.452	14.815
D14.txt (2 +;10 *;)	13.433	28.125	11.940	25.000
D15.txt (2 +;12 *;)	13.235	27.273	16.176	33.333
D16.txt (2 +;14 *;)	16.216	30.769	16.216	30.769
D17.txt (2 +;16 *;)	0.000	0.000	0.000	0.000
D18.txt (4 +;2 *;)	13.699	26.316	16.438	31.579
D19.txt (4 +;4 *;)	8.955	18.750	11.940	25.000
D20.txt (4 +;6 *;)	17.143	34.286	17.143	34.286
D21.txt (4 +;8 *;)	21.795	39.535	19.231	34.884
D22.txt (4 +;10 *;)	17.105	31.707	17.105	31.707
D23.txt (4 +;12 *;)	24.706	42.000	25.882	44.000
D24.txt (4 +;14 *;)	16.883	30.952	18.182	33.333
D25.txt (4 +;16 *;)	10.959	21.053	10.959	21.053

D26.txt (8 +;2 *;)	28.696	41.250	20.000	28.750
D27.txt (8 +;4 *;)	12.500	20.755	11.364	18.868
D28.txt (8 +;6 *;)	13.483	22.222	15.730	25.926
D29.txt (8 +;8 *;)	15.294	26.000	12.941	22.000
D30.txt (8 +;10 *;)	13.043	21.053	11.957	19.298
D31.txt (8 +;12 *;)	17.647	26.866	25.490	38.806
D32.txt (8 +;14 *;)	27.928	40.789	28.829	42.105
D33.txt (8 +;16 *;)	20.183	29.730	20.183	29.730
Average	**13.449**	**26.236**	**13.489**	**26.296**

Benchmark: DIFFEQ2 (11 nodes)

Table 13: PAFUB - DIFFEQ2

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
DI2.txt (1 +;2 *;1 -;1 >;)	4.000	0.000	2.918	10.000
DI3.txt (1 +;3 *;1 -;1 >;)	0.000	0.000	0.000	0.000
DI4.txt (1 +;4 *;1 -;1 >;)	3.846	6.667	0.916	6.667
DI5.txt (2 +;1 *;1 -;1 >;)	0.000	0.000	14.531	39.959
DI6.txt (2 +;2 *;1 -;1 >;)	14.286	17.647	0.000	0.000
DI7.txt (2 +;3 *;1 -;1 >;)	11.111	17.647	-2.314	-14.159
DI8.txt (2 +;4 *;1 -;1 >;)	19.355	30.000	0.925	7.692
DI9.txt (1 +;1 *;2 -;1 >;)	0.000	0.000	16.521	43.654
DI10.txt (1 +;2 *;2 -;1 >;)	20.000	26.316	4.488	15.528
DI11.txt (1 +;3 *;2 -;1 >;)	14.286	22.222	-2.314	-14.159
DI12.txt (1 +;4 *;2 -;1 >;)	16.667	26.316	1.816	13.333
DI13.txt (2 +;1 *;2 -;1 >;)	0.000	0.000	12.155	35.746
DI14.txt (2 +;2 *;2 -;1 >;)	27.273	36.364	1.640	6.207
DI15.txt (2 +;3 *;2 -;1 >;)	22.581	33.333	-4.683	-32.990
DI16.txt (2 +;4 *;2 -;1 >;)	28.571	41.667	0.917	7.692
Average	**12.132**	**17.212**	**3.168**	**8.345**

Benchmark: DIFFEQ2 (11 nodes)

Table 14: PARB - DIFFEQ2

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
DI2.txt (1 +;2 *;1 -;1 >;)	12.000	21.429	1.459	5.000
DI3.txt (1 +;3 *;1 -;1 >;)	12.500	21.429	-2.287	-12.403
DI4.txt (1 +;4 *;1 -;1 >;)	7.692	13.333	0.000	0.000
DI5.txt (2 +;1 *;1 -;1 >;)	0.000	0.000	0.000	0.000

DI6.txt (2 +;2 *;1 -;1 >;)	10.714	17.647	1.481	5.556
DI7.txt (2 +;3 *;1 -;1 >;)	11.111	17.647	-1.157	-7.080
DI8.txt (2 +;4 *;1 -;1 >;)	16.129	25.000	0.000	0.000
DI9.txt (1 +;1 *;2 -;1 >;)	0.000	0.000	0.000	0.000
DI10.txt (1 +;2 *;2 -;1 >;)	10.000	15.789	1.436	4.969
DI11.txt (1 +;3 *;2 -;1 >;)	10.714	16.667	1.302	7.965
DI12.txt (1 +;4 *;2 -;1 >;)	6.667	10.526	0.000	0.000
DI13.txt (2 +;1 *;2 -;1 >;)	0.000	0.000	0.000	0.000
DI14.txt (2 +;2 *;2 -;1 >;)	9.091	13.636	1.458	5.517
DI15.txt (2 +;3 *;2 -;1 >;)	9.677	14.286	0.146	1.031
DI16.txt (2 +;4 *;2 -;1 >;)	14.286	20.833	0.000	0.000
Average	**8.705**	**13.881**	**0.256**	**0.704**

Benchmark: DIFFEQ2 (11 nodes)

Table 15: PAFUB&PARB - DIFFEQ2

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
DI2.txt (1 +;2 *;1 -;1 >;)	16.000	21.429	16.000	21.429
DI3.txt (1 +;3 *;1 -;1 >;)	12.500	21.429	12.500	21.429
DI4.txt (1 +;4 *;1 -;1 >;)	23.077	40.000	7.692	13.333
DI5.txt (2 +;1 *;1 -;1 >;)	0.000	0.000	0.000	0.000
DI6.txt (2 +;2 *;1 -;1 >;)	25.000	35.294	25.000	35.294
DI7.txt (2 +;3 *;1 -;1 >;)	22.222	35.294	22.222	35.294
DI8.txt (2 +;4 *;1 -;1 >;)	35.484	55.000	22.581	35.000
DI9.txt (1 +;1 *;2 -;1 >;)	0.000	0.000	0.000	0.000
DI10.txt (1 +;2 *;2 -;1 >;)	30.000	42.105	30.000	42.105
DI11.txt (1 +;3 *;2 -;1 >;)	25.000	38.889	25.000	38.889
DI12.txt (1 +;4 *;2 -;1 >;)	33.333	52.632	20.000	31.579
DI13.txt (2 +;1 *;2 -;1 >;)	0.000	0.000	0.000	0.000
DI14.txt (2 +;2 *;2 -;1 >;)	36.364	50.000	36.364	50.000
DI15.txt (2 +;3 *;2 -;1 >;)	32.258	47.619	32.258	47.619
DI16.txt (2 +;4 *;2 -;1 >;)	42.857	62.500	31.429	45.833
Average	**22.273**	**33.479**	**18.736**	**27.854**

Benchmark: DIFFEQ4 (17 nodes)

Table 16: PAFUB - DIFFEQ4

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
DI2.txt (1 +;2 *;1 -;)	3.448	0.000	1.547	3.759
DI3.txt (1 +;3 *;1 -;)	3.333	0.000	1.993	6.667
DI4.txt (1 +;4 *;1 -;)	12.500	18.750	2.455	10.345
DI5.txt (1 +;5 *;1 -;)	6.897	8.333	1.389	7.143
DI6.txt (1 +;6 *;1 -;)	5.556	10.526	0.599	3.571
DI7.txt (2 +;1 *;1 -;)	0.000	0.000	21.678	43.704
DI8.txt (2 +;2 *;1 -;)	6.667	7.143	4.074	10.400
DI9.txt (2 +;3 *;1 -;)	3.125	0.000	2.990	10.345
DI10.txt (2 +;4 *;1 -;)	17.143	26.316	2.474	11.060
DI11.txt (2 +;5 *;1 -;)	13.158	19.048	3.473	18.519
DI12.txt (2 +;6 *;1 -;)	10.811	20.000	0.600	3.704
DI13.txt (3 +;1 *;1 -;)	0.000	0.000	19.730	41.388
DI14.txt (3 +;2 *;1 -;)	9.677	13.333	0.322	0.885
DI15.txt (3 +;3 *;1 -;)	8.824	11.765	0.000	0.000
DI16.txt (3 +;4 *;1 -;)	23.684	36.364	0.832	3.980
DI17.txt (3 +;5 *;1 -;)	12.195	16.667	3.474	19.231
DI18.txt (3 +;6 *;1 -;)	0.000	0.000	0.000	0.000
DI1a.txt (1 +;1 *;2 -;)	13.514	19.048	23.183	45.843
DI2a.txt (1 +;2 *;2 -;)	10.811	15.000	1.528	3.759
DI3a.txt (1 +;3 *;2 -;)	10.000	17.391	1.952	6.426
DI4a.txt (1 +;4 *;2 -;)	0.000	0.000	-0.832	-3.846
DI5a.txt (1 +;5 *;2 -;)	9.091	11.765	2.722	13.333
DI6a.txt (1 +;6 *;2 -;)	8.333	10.526	0.670	4.000
DI7a.txt (2 +;1 *;2 -;)	20.000	29.167	22.759	45.843
DI8a.txt (2 +;2 *;2 -;)	10.000	13.043	6.422	16.279
DI9a.txt (2 +;3 *;2 -;)	2.439	4.167	2.961	10.345
DI10a.txt (2 +;4 *;2 -;)	0.000	0.000	2.497	12.000
DI11a.txt (2 +;5 *;2 -;)	3.704	0.000	1.399	8.000
DI12a.txt (2 +;6 *;2 -;)	3.125	0.000	-0.607	-4.145
DI13a.txt (3 +;1 *;2 -;)	9.091	11.765	20.885	43.704
DI14a.txt (3 +;2 *;2 -;)	15.625	26.667	2.822	7.692
DI15a.txt (3 +;3 *;2 -;)	5.556	10.526	0.000	0.000
DI16a.txt (3 +;4 *;2 -;)	0.000	0.000	0.825	3.980
DI17a.txt (3 +;5 *;2 -;)	6.250	6.250	2.780	16.000
DI18a.txt (3 +;6 *;2 -;)	2.941	0.000	0.000	0.000
Average	**7.643**	**10.387**	**4.560**	**12.112**

Benchmark: DIFFEQ4 (17 nodes)

Table 17: PARB - DIFFEQ4

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
DI2.txt (1 +;2 *;1 -;)	20.690	46.154	2.475	6.015
DI3.txt (1 +;3 *;1 -;)	3.333	7.692	0.000	0.000
DI4.txt (1 +;4 *;1 -;)	3.125	6.250	0.000	0.000
DI5.txt (1 +;5 *;1 -;)	3.448	8.333	-1.389	-7.143
DI6.txt (1 +;6 *;1 -;)	11.111	21.053	1.199	7.143
DI7.txt (2 +;1 *;1 -;)	0.000	0.000	0.000	0.000
DI8.txt (2 +;2 *;1 -;)	13.333	28.571	-1.253	-3.200
DI9.txt (2 +;3 *;1 -;)	3.125	6.667	-0.997	-3.448
DI10.txt (2 +;4 *;1 -;)	11.429	21.053	0.928	4.147
DI11.txt (2 +;5 *;1 -;)	13.158	23.810	0.695	3.704
DI12.txt (2 +;6 *;1 -;)	8.108	15.000	2.998	18.519
DI13.txt (3 +;1 *;1 -;)	0.000	0.000	0.000	0.000
DI14.txt (3 +;2 *;1 -;)	12.903	26.667	-1.286	-3.540
DI15.txt (3 +;3 *;1 -;)	2.941	5.882	0.000	0.000
DI16.txt (3 +;4 *;1 -;)	7.895	13.636	0.000	0.000
DI17.txt (3 +;5 *;1 -;)	7.317	12.500	0.000	0.000
DI18.txt (3 +;6 *;1 -;)	2.632	4.762	-0.614	-4.545
DI1a.txt (1 +;1 *;2 -;)	3.125	5.882	-1.922	-3.800
DI2a.txt (1 +;2 *;2 -;)	0.000	0.000	0.000	0.000
DI3a.txt (1 +;3 *;2 -;)	3.125	6.667	-0.976	-3.213
DI4a.txt (1 +;4 *;2 -;)	6.061	11.765	-1.664	-7.692
DI5a.txt (1 +;5 *;2 -;)	6.250	13.333	0.680	3.333
DI6a.txt (1 +;6 *;2 -;)	5.556	10.526	0.000	0.000
DI7a.txt (2 +;1 *;2 -;)	2.703	4.545	-5.660	-11.401
DI8a.txt (2 +;2 *;2 -;)	9.375	18.750	1.223	3.101
DI9a.txt (2 +;3 *;2 -;)	2.941	5.882	-0.987	-3.448
DI10a.txt (2 +;4 *;2 -;)	8.108	14.286	-1.769	-8.500
DI11a.txt (2 +;5 *;2 -;)	13.514	25.000	0.000	0.000
DI12a.txt (2 +;6 *;2 -;)	10.000	17.391	-0.607	-4.145
DI13a.txt (3 +;1 *;2 -;)	2.564	4.167	-1.829	-3.827
DI14a.txt (3 +;2 *;2 -;)	9.091	17.647	2.509	6.838
DI15a.txt (3 +;3 *;2 -;)	2.778	5.263	-0.997	-3.704
DI16a.txt (3 +;4 *;2 -;)	10.000	16.667	2.578	12.438
DI17a.txt (3 +;5 *;2 -;)	7.500	13.043	0.000	0.000
DI18a.txt (3 +;6 *;2 -;)	7.317	12.500	-0.607	-4.348
Average	**6.702**	**12.896**	**-0.208**	**-0.306**

Benchmark: DIFFEQ4 (17 nodes)

Table 18: PAFUB&PARB - DIFFEQ4

	PARB &PAFUB		PAFUB&PARB	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
DI2.txt (1 +;2 *;1 -;)	24.138	46.154	17.241	30.769
DI3.txt (1 +;3 *;1 -;)	6.667	7.692	6.667	7.692
DI4.txt (1 +;4 *;1 -;)	12.500	18.750	12.500	18.750
DI5.txt (1 +;5 *;1 -;)	6.897	8.333	6.897	8.333
DI6.txt (1 +;6 *;1 -;)	11.111	21.053	16.667	31.579
DI7.txt (2 +;1 *;1 -;)	0.000	0.000	0.000	0.000
DI8.txt (2 +;2 *;1 -;)	20.000	35.714	20.000	35.714
DI9.txt (2 +;3 *;1 -;)	6.250	6.667	6.250	6.667
DI10.txt (2 +;4 *;1 -;)	17.143	26.316	17.143	26.316
DI11.txt (2 +;5 *;1 -;)	15.789	23.810	15.789	23.810
DI12.txt (2 +;6 *;1 -;)	18.919	35.000	18.919	35.000
DI13.txt (3 +;1 *;1 -;)	0.000	0.000	0.000	0.000
DI14.txt (3 +;2 *;1 -;)	22.581	40.000	22.581	40.000
DI15.txt (3 +;3 *;1 -;)	11.765	17.647	11.765	17.647
DI16.txt (3 +;4 *;1 -;)	23.684	36.364	23.684	36.364
DI17.txt (3 +;5 *;1 -;)	9.756	12.500	14.634	20.833
DI18.txt (3 +;6 *;1 -;)	2.632	4.762	2.632	4.762
DI1a.txt (1 +;1 *;2 -;)	0.000	0.000	0.000	0.000
DI2a.txt (1 +;2 *;2 -;)	3.704	0.000	3.704	0.000
DI3a.txt (1 +;3 *;2 -;)	6.250	6.667	6.250	6.667
DI4a.txt (1 +;4 *;2 -;)	15.152	23.529	15.152	23.529
DI5a.txt (1 +;5 *;2 -;)	18.750	33.333	18.750	33.333
DI6a.txt (1 +;6 *;2 -;)	5.556	10.526	11.111	21.053
DI7a.txt (2 +;1 *;2 -;)	0.000	0.000	0.000	0.000
DI8a.txt (2 +;2 *;2 -;)	12.500	18.750	15.625	25.000
DI9a.txt (2 +;3 *;2 -;)	5.882	5.882	5.882	5.882
DI10a.txt (2 +;4 *;2 -;)	16.216	23.810	16.216	23.810
DI11a.txt (2 +;5 *;2 -;)	16.216	25.000	16.216	25.000
DI12a.txt (2 +;6 *;2 -;)	17.500	30.435	15.000	26.087
DI13a.txt (3 +;1 *;2 -;)	0.000	0.000	0.000	0.000
DI14a.txt (3 +;2 *;2 -;)	18.182	29.412	18.182	29.412
DI15a.txt (3 +;3 *;2 -;)	11.111	15.789	11.111	15.789
DI16a.txt (3 +;4 *;2 -;)	22.500	33.333	22.500	33.333
DI17a.txt (3 +;5 *;2 -;)	10.000	13.043	15.000	21.739
DI18a.txt (3 +;6 *;2 -;)	7.317	12.500	7.317	12.500
Average	**11.333**	**17.793**	**11.754**	**18.496**

Benchmark: EWF (34 nodes)

Table 19: PAFUB - EWF

PAFUB				
Resource Bag	%SA	**%SSA**	**%MArea**	**%TArea**
E2.txt (2 +;1 *;)	1.923	5.263	1.437	2.194
E3.txt (3 +;1 *;)	1.852	4.762	2.623	3.844
E4.txt (4 +;1 *;)	9.677	20.690	-3.564	-5.169
E5.txt (1 +;2 *;)	6.383	23.077	-1.352	-3.074
E6.txt (2 +;2 *;)	3.774	10.526	-0.122	-0.227
E7.txt (3 +;2 *;)	1.754	4.348	0.853	1.528
E8.txt (4 +;2 *;)	5.000	11.538	2.215	4.070
E9.txt (1 +;3 *;)	11.765	35.294	-0.943	-2.733
E10.txt (2 +;3 *;)	8.621	20.833	1.569	3.501
E11.txt (3 +;3 *;)	3.125	6.667	1.486	3.280
E12.txt (4 +;3 *;)	7.692	16.129	0.849	1.871
E13.txt (1 +;4 *;)	5.660	15.789	1.541	5.320
E14.txt (2 +;4 *;)	6.557	14.815	0.665	1.751
E15.txt (3 +;4 *;)	2.817	5.405	-0.020	-0.053
E16.txt (4 +;4 *;)	4.348	8.571	2.755	7.106
Average	**5.397**	**13.581**	**0.666**	**1.547**

Benchmark: EWF (34 nodes)

Table 20: PARB - EWF

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
E2.txt (2 +;1 *;)	0.000	0.000	0.000	0.000
E3.txt (3 +;1 *;)	0.000	0.000	0.000	0.000
E4.txt (4 +;1 *;)	0.000	0.000	0.000	0.000
E5.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
E6.txt (2 +;2 *;)	0.000	0.000	0.000	0.000
E7.txt (3 +;2 *;)	0.000	0.000	0.000	0.000
E8.txt (4 +;2 *;)	1.667	3.846	0.000	0.000
E9.txt (1 +;3 *;)	0.000	0.000	0.000	0.000
E10.txt (2 +;3 *;)	0.000	0.000	0.000	0.000
E11.txt (3 +;3 *;)	0.000	0.000	0.000	0.000
E12.txt (4 +;3 *;)	1.538	3.226	0.000	0.000
E13.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
E14.txt (2 +;4 *;)	0.000	0.000	0.000	0.000
E15.txt (3 +;4 *;)	0.000	0.000	0.000	0.000

E16.txt (4 +;4 *;)	1.449	2.857	0.000	0.000
Average	**0.310**	**0.662**	**0.000**	**0.000**

Benchmark: EWF (34 nodes)

Table 21: PAFUB&PARB - EWF

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
E2.txt (2 +;1 *;)	3.846	10.526	1.923	5.263
E3.txt (3 +;1 *;)	1.852	4.762	1.852	4.762
E4.txt (4 +;1 *;)	6.452	13.793	8.065	17.241
E5.txt (1 +;2 *;)	6.383	23.077	6.383	23.077
E6.txt (2 +;2 *;)	3.774	10.526	3.774	10.526
E7.txt (3 +;2 *;)	1.754	4.348	1.754	4.348
E8.txt (4 +;2 *;)	5.000	11.538	5.000	11.538
E9.txt (1 +;3 *;)	11.765	35.294	11.765	35.294
E10.txt (2 +;3 *;)	8.621	20.833	8.621	20.833
E11.txt (3 +;3 *;)	3.125	6.667	3.125	6.667
E12.txt (4 +;3 *;)	7.692	16.129	7.692	16.129
E13.txt (1 +;4 *;)	5.660	15.789	5.660	15.789
E14.txt (2 +;4 *;)	6.557	14.815	6.557	14.815
E15.txt (3 +;4 *;)	2.817	5.405	2.817	5.405
E16.txt (4 +;4 *;)	4.348	8.571	5.797	11.429
Average	**5.310**	**13.472**	**5.386**	**13.541**

Benchmark: FIR (13 nodes)

Table 22: PAFUB - FIR

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
FIR2.txt (2 *;1 +;)	0.000	0.000	0.000	0.000
FIR3.txt (3 *;1 +;)	0.000	0.000	0.000	0.000
FIR4.txt (4 *;1 +;)	0.000	0.000	0.000	0.000
FIR5.txt (5 *;1 +;)	4.762	12.500	-0.768	-6.612
FIR6.txt (6 *;1 +;)	21.429	40.000	-0.654	-6.598
FIR7.txt (7 *;1 +;)	0.000	0.000	0.000	0.000
FIR8.txt (1 *;2 +;)	0.000	0.000	13.527	30.806
FIR9.txt (2 *;2 +;)	0.000	0.000	0.000	0.000
FIR10.txt (3 *;2 +;)	0.000	0.000	0.000	0.000
FIR11.txt (4 *;2 +;)	0.000	0.000	0.000	0.000

FIR12.txt (5 *;2 +;)	4.000	8.333	-1.515	-12.500
FIR13.txt (6 *;2 +;)	19.355	33.333	-0.650	-6.667
FIR14.txt (7 *;2 +;)	0.000	0.000	0.000	0.000
Average	**3.811**	**7.244**	**0.765**	**-0.121**

Benchmark: FIR (13 nodes)

Table 23: PARB - FIR

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
FIR2.txt (2 *;1 +;)	0.000	0.000	0.000	0.000
FIR3.txt (3 *;1 +;)	0.000	0.000	0.000	0.000
FIR4.txt (4 *;1 +;)	0.000	0.000	0.000	0.000
FIR5.txt (5 *;1 +;)	4.762	12.500	-1.537	-13.223
FIR6.txt (6 *;1 +;)	21.429	40.000	0.000	0.000
FIR7.txt (7 *;1 +;)	0.000	0.000	0.000	0.000
FIR8.txt (1 *;2 +;)	16.000	33.333	-13.598	-30.968
FIR9.txt (2 *;2 +;)	4.545	11.111	2.719	7.692
FIR10.txt (3 *;2 +;)	0.000	0.000	0.000	0.000
FIR11.txt (4 *;2 +;)	0.000	0.000	0.000	0.000
FIR12.txt (5 *;2 +;)	4.000	8.333	-0.758	-6.250
FIR13.txt (6 *;2 +;)	19.355	33.333	0.000	0.000
FIR14.txt (7 *;2 +;)	0.000	0.000	0.000	0.000
Average	**5.392**	**10.662**	**-1.013**	**-3.288**

Benchmark: FIR (13 nodes)

Table 24: PAFUB&PARB - FIR

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
FIR2.txt (2 *;1 +;)	0.000	0.000	0.000	0.000
FIR3.txt (3 *;1 +;)	0.000	0.000	0.000	0.000
FIR4.txt (4 *;1 +;)	0.000	0.000	0.000	0.000
FIR5.txt (5 *;1 +;)	4.762	12.500	4.762	12.500
FIR6.txt (6 *;1 +;)	21.429	40.000	21.429	40.000
FIR7.txt (7 *;1 +;)	0.000	0.000	0.000	0.000
FIR8.txt (1 *;2 +;)	0.000	0.000	0.000	0.000
FIR9.txt (2 *;2 +;)	4.545	11.111	4.545	11.111
FIR10.txt (3 *;2 +;)	0.000	0.000	0.000	0.000
FIR11.txt (4 *;2 +;)	0.000	0.000	0.000	0.000

FIR12.txt (5 *;2 +;)	4.000	8.333	4.000	8.333
FIR13.txt (6 *;2 +;)	19.355	33.333	19.355	33.333
FIR14.txt (7 *;2 +;)	0.000	0.000	0.000	0.000
Average	**4.161**	**8.098**	**4.161**	**8.098**

Benchmark: IIR (14 nodes)

Table 25: PAFUB - IIR

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
I2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
I3.txt (1 +;3 *;)	13.889	22.727	-1.081	-4.324
I4.txt (1 +;4 *;)	13.333	25.000	0.882	4.734
I5.txt (1 +;5 *;)	30.769	48.000	0.000	0.000
I6.txt (1 +;6 *;)	26.316	41.667	0.637	5.229
I7.txt (2 +;1 *;)	0.000	0.000	21.973	43.457
I8.txt (2 +;2 *;)	24.242	42.105	1.141	3.004
I9.txt (2 +;3 *;)	22.727	33.333	-1.069	-3.947
I10.txt (2 +;4 *;)	28.947	45.833	2.734	14.793
I11.txt (2 +;5 *;)	27.500	42.308	0.735	4.969
I12.txt (2 +;6 *;)	36.364	53.333	3.204	24.260
I13.txt (3 +;1 *;)	0.000	0.000	21.747	43.735
I14.txt (3 +;2 *;)	31.579	50.000	3.953	10.766
I15.txt (3 +;3 *;)	29.412	40.541	3.046	11.047
I16.txt (3 +;4 *;)	25.532	36.364	2.606	13.681
I17.txt (3 +;5 *;)	26.531	37.143	0.023	0.163
I18.txt (3 +;6 *;)	30.769	42.105	1.919	16.695
Average	**21.642**	**32.968**	**3.674**	**11.074**

Benchmark: IIR (14 nodes)

Table 26: PARB - IIR

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
I2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
I3.txt (1 +;3 *;)	27.778	45.455	-3.242	-12.973
I4.txt (1 +;4 *;)	3.333	6.250	-0.882	-4.734
I5.txt (1 +;5 *;)	23.077	36.000	-0.797	-4.937
I6.txt (1 +;6 *;)	15.789	25.000	-0.737	-6.046
I7.txt (2 +;1 *;)	9.091	15.789	-1.935	-3.827

I8.txt (2 +;2 *;)	3.030	5.263	1.304	3.433
I9.txt (2 +;3 *;)	27.273	40.000	-1.036	-3.828
I10.txt (2 +;4 *;)	10.526	16.667	-4.429	-23.964
I11.txt (2 +;5 *;)	7.500	11.538	-0.735	-4.969
I12.txt (2 +;6 *;)	15.909	23.333	0.000	0.000
I13.txt (3 +;1 *;)	7.143	10.714	-0.183	-0.369
I14.txt (3 +;2 *;)	2.632	4.167	2.608	7.103
I15.txt (3 +;3 *;)	23.529	32.432	-1.079	-3.913
I16.txt (3 +;4 *;)	10.638	15.152	-3.493	-18.336
I17.txt (3 +;5 *;)	8.163	11.429	-4.454	-31.648
I18.txt (3 +;6 *;)	13.462	18.421	-0.020	-0.172
Average	**12.287**	**18.683**	**-1.124**	**-6.422**

Benchmark: IIR (14 nodes)

Table 27: PAFUB&PARB - IIR

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
I2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
I3.txt (1 +;3 *;)	33.333	54.545	25.000	40.909
I4.txt (1 +;4 *;)	16.667	31.250	16.667	31.250
I5.txt (1 +;5 *;)	30.769	48.000	33.333	52.000
I6.txt (1 +;6 *;)	26.316	41.667	26.316	41.667
I7.txt (2 +;1 *;)	0.000	0.000	0.000	0.000
I8.txt (2 +;2 *;)	24.242	42.105	30.303	52.632
I9.txt (2 +;3 *;)	45.455	66.667	31.818	46.667
I10.txt (2 +;4 *;)	34.211	54.167	34.211	54.167
I11.txt (2 +;5 *;)	30.000	46.154	30.000	46.154
I12.txt (2 +;6 *;)	36.364	53.333	36.364	53.333
I13.txt (3 +;1 *;)	0.000	0.000	0.000	0.000
I14.txt (3 +;2 *;)	34.211	54.167	34.211	54.167
I15.txt (3 +;3 *;)	33.333	45.946	35.294	48.649
I16.txt (3 +;4 *;)	27.660	39.394	29.787	42.424
I17.txt (3 +;5 *;)	26.531	37.143	26.531	37.143
I18.txt (3 +;6 *;)	30.769	42.105	30.769	42.105
Average	**25.286**	**38.626**	**24.741**	**37.839**

Benchmark: LMS (17 nodes)

Table 28: PAFUB - LMS

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
LM2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
LM3.txt (1 +;3 *;)	0.000	0.000	0.000	0.000
LM4.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
LM5.txt (2 +;1 *;)	0.000	0.000	27.903	51.438
LM6.txt (2 +;2 *;)	0.000	0.000	0.000	0.000
LM7.txt (2 +;3 *;)	0.000	0.000	0.000	0.000
LM8.txt (2 +;4 *;)	6.250	13.333	1.535	6.466
LM9.txt (3 +;1 *;)	0.000	0.000	27.412	51.438
LM10.txt (3 +;2 *;)	15.789	28.571	2.480	6.226
LM11.txt (3 +;3 *;)	11.111	17.857	2.015	6.227
LM12.txt (3 +;4 *;)	13.158	23.810	2.416	10.000
LM13.txt (4 +;1 *;)	0.000	0.000	27.101	51.644
LM14.txt (4 +;2 *;)	22.222	35.714	3.675	9.339
LM15.txt (4 +;3 *;)	32.075	47.222	4.907	15.953
LM16.txt (4 +;4 *;)	12.766	20.000	0.898	3.734
Average	**7.558**	**12.434**	**6.689**	**14.164**

Benchmark: LMS (17 nodes)

Table 29: PAFUB&PARB - LMS

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
LM2.txt (1 +;2 *;)	0.000	0.000	0.000	0.000
LM3.txt (1 +;3 *;)	0.000	0.000	0.000	0.000
LM4.txt (1 +;4 *;)	0.000	0.000	0.000	0.000
LM5.txt (2 +;1 *;)	0.000	0.000	0.000	0.000
LM6.txt (2 +;2 *;)	0.000	0.000	0.000	0.000
LM7.txt (2 +;3 *;)	0.000	0.000	0.000	0.000
LM8.txt (2 +;4 *;)	6.250	13.333	6.250	13.333
LM9.txt (3 +;1 *;)	0.000	0.000	0.000	0.000
LM10.txt (3 +;2 *;)	15.789	28.571	15.789	28.571
LM11.txt (3 +;3 *;)	11.111	17.857	11.111	17.857
LM12.txt (3 +;4 *;)	13.158	23.810	13.158	23.810
LM13.txt (4 +;1 *;)	0.000	0.000	0.000	0.000
LM14.txt (4 +;2 *;)	22.222	35.714	22.222	35.714
LM15.txt (4 +;3 *;)	32.075	47.222	32.075	47.222

LM16.txt (4 +;4 *;)	12.766	20.000	12.766	20.000
Average	**7.558**	**12.434**	**7.558**	**12.434**

Benchmark: nestor2 (39 nodes)

Table 30: PAFUB - nestor2

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
nest2.txt (1 +;1 -;2 *;)	7.273	25.000	0.000	0.000
nest3.txt (1 +;1 -;4 *;)	10.000	28.571	0.000	0.000
nest4.txt (1 +;1 -;8 *;)	10.448	25.000	0.808	3.173
nest5.txt (2 +;1 -;1 *;)	6.780	16.667	4.714	6.596
nest6.txt (2 +;1 -;2 *;)	5.357	11.765	2.609	4.329
nest7.txt (2 +;1 -;4 *;)	5.000	14.286	2.339	5.310
nest8.txt (2 +;1 -;6 *;)	1.587	4.167	2.890	8.489
nest9.txt (2 +;1 -;8 *;)	4.110	8.824	1.169	4.216
nest10.txt (4 +;1 -;1 *;)	15.385	33.333	5.224	7.485
nest11.txt (4 +;1 -;2 *;)	16.667	38.095	5.205	9.078
nest12.txt (4 +;1 -;4 *;)	5.556	12.121	3.643	8.743
nest13.txt (4 +;1 -;6 *;)	4.819	9.091	0.450	1.408
nest14.txt (4 +;1 -;8 *;)	7.292	12.281	3.138	11.830
nest15.txt (1 +;2 -;1 *;)	0.000	0.000	0.000	0.000
nest16.txt (1 +;2 -;2 *;)	8.929	29.412	1.602	2.706
nest17.txt (1 +;2 -;4 *;)	4.918	13.636	1.233	2.938
nest18.txt (1 +;2 -;6 *;)	5.970	14.286	0.425	1.325
nest19.txt (1 +;2 -;8 *;)	14.667	30.556	0.050	0.195
nest20.txt (1 +;4 -;1 *;)	6.349	14.286	0.000	0.000
nest21.txt (1 +;4 -;2 *;)	7.018	22.222	1.840	3.389
nest22.txt (1 +;4 -;4 *;)	17.143	38.710	0.547	1.369
nest23.txt (1 +;4 -;8 *;)	26.136	46.939	1.665	7.359
nest24.txt (2 +;2 -;2 *;)	3.448	10.526	2.546	4.390
nest25.txt (2 +;2 -;4 *;)	12.281	38.889	-1.268	-3.192
nest26.txt (2 +;2 -;8 *;)	9.524	25.000	-1.645	-6.873
nest27.txt (4 +;4 -;8 *;)	6.452	11.111	1.602	6.565
Average	**8.581**	**20.568**	**1.569**	**3.493**

Benchmark: nestor2 (39 nodes)

Table 31: PARB - nestor2

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
nest2.txt (1 +;1 -;2 *;)	1.818	6.250	0.931	1.630
nest3.txt (1 +;1 -;4 *;)	0.000	0.000	0.000	0.000
nest4.txt (1 +;1 -;8 *;)	0.000	0.000	0.000	0.000
nest5.txt (2 +;1 -;1 *;)	1.695	4.167	3.397	4.753
nest6.txt (2 +;1 -;2 *;)	0.000	0.000	0.000	0.000
nest7.txt (2 +;1 -;4 *;)	0.000	0.000	0.000	0.000
nest8.txt (2 +;1 -;6 *;)	0.000	0.000	0.000	0.000
nest9.txt (2 +;1 -;8 *;)	0.000	0.000	0.000	0.000
nest10.txt (4 +;1 -;1 *;)	3.077	6.667	-1.024	-1.468
nest11.txt (4 +;1 -;2 *;)	0.000	0.000	0.000	0.000
nest12.txt (4 +;1 -;4 *;)	0.000	0.000	0.000	0.000
nest13.txt (4 +;1 -;6 *;)	0.000	0.000	0.000	0.000
nest14.txt (4 +;1 -;8 *;)	0.000	0.000	0.000	0.000
nest15.txt (1 +;2 -;1 *;)	2.000	6.667	2.521	3.704
nest16.txt (1 +;2 -;2 *;)	1.786	5.882	2.916	4.925
nest17.txt (1 +;2 -;4 *;)	0.000	0.000	0.000	0.000
nest18.txt (1 +;2 -;6 *;)	0.000	0.000	0.000	0.000
nest19.txt (1 +;2 -;8 *;)	0.000	0.000	0.000	0.000
nest20.txt (1 +;4 -;1 *;)	0.000	0.000	0.000	0.000
nest21.txt (1 +;4 -;2 *;)	0.000	0.000	0.000	0.000
nest22.txt (1 +;4 -;4 *;)	0.000	0.000	0.000	0.000
nest23.txt (1 +;4 -;8 *;)	1.136	2.041	-0.387	-1.711
nest24.txt (2 +;2 -;2 *;)	1.724	5.263	-0.053	-0.091
nest25.txt (2 +;2 -;4 *;)	0.000	0.000	0.000	0.000
nest26.txt (2 +;2 -;8 *;)	0.000	0.000	0.000	0.000
nest27.txt (4 +;4 -;8 *;)	4.301	7.407	1.192	4.885
Average	**0.675**	**1.706**	**0.365**	**0.639**

Benchmark: nestor2 (39 nodes)

Table 32: PAFUB&PARB - nestor2

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
nest2.txt (1 +;1 -;2 *;)	9.091	31.250	9.091	31.250
nest3.txt (1 +;1 -;4 *;)	10.000	28.571	10.000	28.571
nest4.txt (1 +;1 -;8 *;)	10.448	25.000	10.448	25.000
nest5.txt (2 +;1 -;1 *;)	8.475	20.833	8.475	20.833

nest6.txt (2 +;1 -;2 *;)	5.357	11.765	5.357	11.765
nest7.txt (2 +;1 -;4 *;)	5.000	14.286	5.000	14.286
nest8.txt (2 +;1 -;6 *;)	1.587	4.167	1.587	4.167
nest9.txt (2 +;1 -;8 *;)	4.110	8.824	4.110	8.824
nest10.txt (4 +;1 -;1 *;)	18.462	40.000	18.462	40.000
nest11.txt (4 +;1 -;2 *;)	18.333	42.857	18.333	42.857
nest12.txt (4 +;1 -;4 *;)	6.944	15.152	6.944	15.152
nest13.txt (4 +;1 -;6 *;)	4.819	9.091	4.819	9.091
nest14.txt (4 +;1 -;8 *;)	8.333	14.035	8.333	14.035
nest15.txt (1 +;2 -;1 *;)	2.000	6.667	2.000	6.667
nest16.txt (1 +;2 -;2 *;)	10.714	35.294	10.714	35.294
nest17.txt (1 +;2 -;4 *;)	4.918	13.636	4.918	13.636
nest18.txt (1 +;2 -;6 *;)	8.955	21.429	8.955	21.429
nest19.txt (1 +;2 -;8 *;)	13.333	27.778	12.000	25.000
nest20.txt (1 +;4 -;1 *;)	6.349	14.286	6.349	14.286
nest21.txt (1 +;4 -;2 *;)	7.018	22.222	7.018	22.222
nest22.txt (1 +;4 -;4 *;)	17.143	38.710	17.143	38.710
nest23.txt (1 +;4 -;8 *;)	22.727	40.816	22.727	40.816
nest24.txt (2 +;2 -;2 *;)	3.448	10.526	3.448	10.526
nest25.txt (2 +;2 -;4 *;)	10.526	33.333	10.526	33.333
nest26.txt (2 +;2 -;8 *;)	6.349	16.667	6.349	16.667
nest27.txt (4 +;4 -;8 *;)	10.753	18.519	10.753	18.519
Average	**9.046**	**21.758**	**8.995**	**21.651**

Benchmark: wavelet (16 nodes)

Table 33: PAFUB - wavelet

PAFUB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
w1.txt (1 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w2.txt (1 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w3.txt (1 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w4.txt (1 +;1 -;4 *;)	24.242	47.059	0.000	0.000
w5.txt (1 +;1 -;5 *;)	13.514	23.810	0.000	0.000
w7.txt (2 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w8.txt (2 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w9.txt (2 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w10.txt (2 +;1 -;4 *;)	13.158	22.727	0.000	0.000
w11.txt (2 +;1 -;5 *;)	14.634	24.000	0.000	0.000
Average	**6.555**	**11.760**	**0.000**	**0.000**

Benchmark: wavelet (16 nodes)

Table 34: PARB - wavelet

PARB				
Resource Bag	**%SA**	**%SSA**	**%MArea**	**%TArea**
w1.txt (1 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w2.txt (1 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w3.txt (1 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w4.txt (1 +;1 -;4 *;)	24.242	47.059	0.000	0.000
w5.txt (1 +;1 -;5 *;)	13.514	23.810	0.000	0.000
w7.txt (2 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w8.txt (2 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w9.txt (2 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w10.txt (2 +;1 -;4 *;)	13.158	22.727	0.000	0.000
w11.txt (2 +;1 -;5 *;)	14.634	24.000	0.000	0.000
Average	**6.555**	**11.760**	**0.000**	**0.000**

Benchmark: wavelet (16 nodes)

Table 35: PAFUB&PARB - wavelet

	PARB &PAFUB		**PAFUB&PARB**	
Resource Bag	**%SA**	**%SSA**	**%SA**	**%SSA**
w1.txt (1 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w2.txt (1 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w3.txt (1 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w4.txt (1 +;1 -;4 *;)	24.242	47.059	24.242	47.059
w5.txt (1 +;1 -;5 *;)	13.514	23.810	13.514	23.810
w7.txt (2 +;1 -;1 *;)	0.000	0.000	0.000	0.000
w8.txt (2 +;1 -;2 *;)	0.000	0.000	0.000	0.000
w9.txt (2 +;1 -;3 *;)	0.000	0.000	0.000	0.000
w10.txt (2 +;1 -;4 *;)	13.158	22.727	13.158	22.727
w11.txt (2 +;1 -;5 *;)	14.634	24.000	14.634	24.000
Average	**6.555**	**11.760**	**6.555**	**11.760**

The below table combines all the results for the different benchmarks:

Table 36: All Techniques - All Benchmarks

Benchmark	PAFUB		PARB		PARB &PAFUB		PAFUB&PARB	
	%SA	%SSA	%SA	%SSA	%SA	%SSA	%SA	%SSA
4pDCT (15 nodes)	7.136	14.925	19.328	40.412	22.186	46.262	22.025	46.183
AR (28 nodes)	3.693	6.177	3.417	6.358	5.673	10.023	5.306	9.367
DCT1 (35 nodes)	11.508	22.424	5.767	11.212	13.449	26.236	13.489	26.296
DIFFEQ2 (11 nodes)	12.132	17.212	8.705	13.881	22.273	33.479	18.736	27.854
DIFFEQ4 (17 nodes)	7.643	10.387	6.702	12.896	11.333	17.793	11.754	18.496
EWF (34 nodes)	5.397	13.581	0.310	0.662	5.310	13.472	5.386	13.541
FIR (13 nodes)	3.811	7.244	5.392	10.662	4.161	8.098	4.161	8.098
IIR (14 nodes)	21.642	32.968	12.287	18.683	25.286	38.626	24.741	37.839
LMS (17 nodes)	7.558	12.434	0.000	0.000	7.558	12.434	7.558	12.434
nestor2 (39 nodes)	8.581	20.568	0.675	1.706	9.046	21.758	8.995	21.651
wavelet (16 nodes)	6.555	11.760	6.555	11.760	6.555	11.760	6.555	11.760
Average	**8.696**	**15.4254**	**6.28527**	**11.6574**	**12.0754**	**21.8128**	**11.7005**	**21.229**

Table 37 shows a collection of all the results for the 11 benchmarks, showing the maximum, minimum, and average improvement for each benchmark. It also shows the maximum improvement of all, the minimum of all, and the average of all:

Table 37: Maximum, minimum, and average improvements

Benchmark	Maximum	Average	Minimum
4pDCT (15 nodes)	21.053	11.641	0.000
AR (28 nodes)	8.333	4.285	0.000
DCT1 (35 nodes)	43.902	25.939	**12.766**
DCT2 (70 nodes)	33.766	21.044	13.158
DIFFEQ2 (11 nodes)	**41.667**	24.416	0.000
DIFFEQ3 (16 nodes)	14.286	2.796	0.000
DIFFEQ4 (17 nodes)	35.714	16.989	0.000
EWF (34 nodes)	16.129	7.837	0.000
FDCT (42 nodes)	27.027	7.926	0.000
FIR (13 ndoes)	74.000	32.112	0.000
IIR (14 nodes)	53.333	**45.743**	36.364

LMS (17 nodes)	47.222	24.365	0.000
wavelet (16 nodes)	50.000	34.670	0.000
nestor2	21.429	16.056	10.526
Maximum =	**74.000**	**45.743**	**36.364**
Average =	**34.847**	**19.701**	**5.201**
Minimum =	**8.333**	**2.796**	**0.000**

5.2 Temperature Results

Benchmark: 4pDCT

Table 38: Temperature - 4pDCT

Resource Bag	Initial T	Final T	%
4D1.txt (1 *;1 +;1 -;)	18	18	0.000
4D10.txt (2 *;1 +;2 -;)	18	17	5.556
4D11.txt (3 *;1 +;2 -;)	21	19	9.524
4D12.txt (4 *;1 +;2 -;)	24	21	12.500
4D13.txt (1 *;2 +;2 -;)	28	21	25.000
4D14.txt (2 *;2 +;2 -;)	22	22	0.000
4D15.txt (3 *;2 +;2 -;)	25	22	12.000
4D16.txt (4 *;2 +;2 -;)	26	22	15.385
4D2.txt (2 *;1 +;1 -;)	21	21	0.000
4D3.txt (3 *;1 +;1 -;)	24	24	0.000
4D4.txt (4 *;1 +;1 -;)	27	27	0.000
4D5.txt (1 *;2 +;1 -;)	20	17	15.000
4D6.txt (2 *;2 +;1 -;)	18	17	5.556
4D7.txt (3 *;2 +;1 -;)	24	21	12.500
4D8.txt (4 *;2 +;1 -;)	26	24	7.692
4D9.txt (1 *;1 +;2 -;)	25	22	12.000
Average			**8.294**

Benchmark: AR

Table 39: Temperature - AR

Resource Bag	Initial T	Final T	%
A1.txt (1 +;1 *;)	35	35	0.000
A10.txt (2 +;2 *;)	43	42	2.326
A11.txt (2 +;3 *;)	45	44	2.222
A12.txt (2 +;4 *;)	51	51	0.000
A13.txt (2 +;5 *;)	59	57	3.390

A14.txt (2 +;6 *;)	63	62	1.587
A15.txt (2 +;7 *;)	64	64	0.000
A16.txt (2 +;8 *;)	68	68	0.000
A17.txt (3 +;1 *;)	53	46	13.208
A18.txt (3 +;2 *;)	46	44	4.348
A19.txt (3 +;3 *;)	58	54	6.897
A2.txt (1 +;2 *;)	37	37	0.000
A20.txt (3 +;4 *;)	61	59	3.279
A21.txt (3 +;5 *;)	60	57	5.000
A22.txt (3 +;6 *;)	69	68	1.449
A23.txt (3 +;7 *;)	73	72	1.370
A24.txt (3 +;8 *;)	79	79	0.000
A25.txt (4 +;1 *;)	63	50	20.635
A26.txt (4 +;2 *;)	53	51	3.774
A27.txt (4 +;3 *;)	64	60	6.250
A28.txt (4 +;4 *;)	66	63	4.545
A29.txt (4 +;5 *;)	66	65	1.515
A3.txt (1 +;3 *;)	42	41	2.381
A30.txt (4 +;6 *;)	73	73	0.000
A31.txt (4 +;7 *;)	79	78	1.266
A32.txt (4 +;8 *;)	89	85	4.494
A4.txt (1 +;4 *;)	46	46	0.000
A5.txt (1 +;5 *;)	51	51	0.000
A6.txt (1 +;6 *;)	55	55	0.000
A7.txt (1 +;7 *;)	56	56	0.000
A8.txt (1 +;8 *;)	59	59	0.000
A9.txt (2 +;1 *;)	47	41	12.766
Average			**3.209**

Benchmark: DCT1

Table 40: Temperature - DCT1

Resource Bag	Initial T	Final T	%
D10.txt (2 +;2 *;)	50	46	8.000
D11.txt (2 +;4 *;)	49	45	8.163
D12.txt (2 +;6 *;)	47	47	0.000
D13.txt (2 +;8 *;)	52	52	0.000
D14.txt (2 +;10 *;)	55	54	1.818
D15.txt (2 +;12 *;)	54	47	12.963
D16.txt (2 +;14 *;)	58	50	13.793

D17.txt (2 +;16 *;)	50	50	0.000
D18.txt (4 +;2 *;)	67	58	13.433
D19.txt (4 +;4 *;)	59	51	13.559
D2.txt (1 +;2 *;)	44	43	2.273
D20.txt (4 +;6 *;)	60	51	15.000
D21.txt (4 +;8 *;)	66	61	7.576
D22.txt (4 +;10 *;)	62	57	8.065
D23.txt (4 +;12 *;)	69	56	18.841
D24.txt (4 +;14 *;)	59	57	3.390
D25.txt (4 +;16 *;)	53	51	3.774
D26.txt (8 +;2 *;)	105	100	4.762
D27.txt (8 +;4 *;)	76	69	9.211
D28.txt (8 +;6 *;)	75	61	18.667
D29.txt (8 +;8 *;)	69	61	11.594
D3.txt (1 +;4 *;)	47	43	8.511
D30.txt (8 +;10 *;)	74	65	12.162
D31.txt (8 +;12 *;)	82	73	10.976
D32.txt (8 +;14 *;)	89	71	20.225
D33.txt (8 +;16 *;)	85	79	7.059
D4.txt (1 +;6 *;)	50	43	14.000
D5.txt (1 +;8 *;)	48	44	8.333
D6.txt (1 +;10 *;)	47	44	6.383
D7.txt (1 +;12 *;)	46	40	13.043
D8.txt (1 +;14 *;)	48	42	12.500
D9.txt (1 +;16 *;)	49	44	10.204
Average			**9.321**

Benchmark: DIFFEQ2

Table 41: Temperature - DIFFEQ2

Resource Bag	Initial T	Final T	%
DI1.txt (1 +;1 *;1 -;1 >;)	18	18	0.000
DI10.txt (1 +;2 *;2 -;1 >;)	24	19	20.833
DI11.txt (1 +;3 *;2 -;1 >;)	21	18	14.286
DI12.txt (1 +;4 *;2 -;1 >;)	22	18	18.182
DI13.txt (2 +;1 *;2 -;1 >;)	25	25	0.000
DI14.txt (2 +;2 *;2 -;1 >;)	26	19	26.923
DI15.txt (2 +;3 *;2 -;1 >;)	23	18	21.739
DI16.txt (2 +;4 *;2 -;1 >;)	26	18	30.769
DI2.txt (1 +;2 *;1 -;1 >;)	20	19	5.000
DI3.txt (1 +;3 *;1 -;1 >;)	18	18	0.000

DI4.txt (1 +;4 *;1 -;1 >;)	19	18	5.263
DI5.txt (2 +;1 *;1 -;1 >;)	21	21	0.000
DI6.txt (2 +;2 *;1 -;1 >;)	22	19	13.636
DI7.txt (2 +;3 *;1 -;1 >;)	20	18	10.000
DI8.txt (2 +;4 *;1 -;1 >;)	23	18	21.739
DI9.txt (1 +;1 *;2 -;1 >;)	22	22	0.000
Average			**11.773**

Benchmark: DIFFEQ4

Table 42: Temperature - DIFFEQ4

Resource Bag	Initial T	Final T	%
DI1.txt (1 +;1 *;1 -;)	23	23	0.000
DI10.txt (2 +;4 *;1 -;)	28	23	17.857
DI10a.txt (2 +;4 *;2 -;)	29	24	17.241
DI11.txt (2 +;5 *;1 -;)	30	25	16.667
DI11a.txt (2 +;5 *;2 -;)	28	25	10.714
DI12.txt (2 +;6 *;1 -;)	28	24	14.286
DI12a.txt (2 +;6 *;2 -;)	30	30	0.000
DI13.txt (3 +;1 *;1 -;)	28	28	0.000
DI13a.txt (3 +;1 *;2 -;)	33	33	0.000
DI14.txt (3 +;2 *;1 -;)	25	23	8.000
DI14a.txt (3 +;2 *;2 -;)	26	25	3.846
DI15.txt (3 +;3 *;1 -;)	27	25	7.407
DI15a.txt (3 +;3 *;2 -;)	28	26	7.143
DI16.txt (3 +;4 *;1 -;)	30	24	20.000
DI16a.txt (3 +;4 *;2 -;)	31	24	22.581
DI17.txt (3 +;5 *;1 -;)	32	28	12.500
DI17a.txt (3 +;5 *;2 -;)	30	26	13.333
DI18.txt (3 +;6 *;1 -;)	28	28	0.000
DI18a.txt (3 +;6 *;2 -;)	30	29	3.333
DI1a.txt (1 +;1 *;2 -;)	28	28	0.000
DI2.txt (1 +;2 *;1 -;)	25	24	4.000
DI2a.txt (1 +;2 *;2 -;)	22	21	4.545
DI3.txt (1 +;3 *;1 -;)	25	24	4.000
DI3a.txt (1 +;3 *;2 -;)	26	25	3.846
DI4.txt (1 +;4 *;1 -;)	26	23	11.538
DI4a.txt (1 +;4 *;2 -;)	26	23	11.538
DI5.txt (1 +;5 *;1 -;)	22	20	9.091
DI5a.txt (1 +;5 *;2 -;)	24	20	16.667
DI6.txt (1 +;6 *;1 -;)	28	28	0.000

DI6a.txt (1 +;6 *;2 -;)	27	27	0.000
DI7.txt (2 +;1 *;1 -;)	27	27	0.000
DI7a.txt (2 +;1 *;2 -;)	32	32	0.000
DI8.txt (2 +;2 *;1 -;)	25	23	8.000
DI8a.txt (2 +;2 *;2 -;)	26	25	3.846
DI9.txt (2 +;3 *;1 -;)	26	25	3.846
DI9a.txt (2 +;3 *;2 -;)	27	26	3.704
Average			**7.209**

Benchmark: EWF

Table 43: Temperature - EWF

Resource Bag	Initial T	Final T	%
E1.txt (1 +;1 *;)	40	40	0.000
E10.txt (2 +;3 *;)	53	48	9.434
E11.txt (3 +;3 *;)	58	56	3.448
E12.txt (4 +;3 *;)	58	52	10.345
E13.txt (1 +;4 *;)	48	45	6.250
E14.txt (2 +;4 *;)	55	51	7.273
E15.txt (3 +;4 *;)	64	62	3.125
E16.txt (4 +;4 *;)	61	58	4.918
E2.txt (2 +;1 *;)	49	48	2.041
E3.txt (3 +;1 *;)	50	49	2.000
E4.txt (4 +;1 *;)	57	53	7.018
E5.txt (1 +;2 *;)	44	41	6.818
E6.txt (2 +;2 *;)	49	47	4.082
E7.txt (3 +;2 *;)	52	51	1.923
E8.txt (4 +;2 *;)	54	50	7.407
E9.txt (1 +;3 *;)	47	42	10.638
Average			**5.420**

Benchmark: FIR

Table 44: Temperature - FIR

Resource Bag	Initial T	Final T	%
FIR1.txt (1 *;1 +;)	14	14	0.000
FIR10.txt (3 *;2 +;)	18	18	0.000
FIR11.txt (4 *;2 +;)	17	17	0.000
FIR12.txt (5 *;2 +;)	18	18	0.000
FIR13.txt (6 *;2 +;)	23	17	26.087
FIR14.txt (7 *;2 +;)	24	24	0.000

FIR2.txt (2 *;1 +;)	15	15	0.000
FIR3.txt (3 *;1 +;)	15	15	0.000
FIR4.txt (4 *;1 +;)	14	14	0.000
FIR5.txt (5 *;1 +;)	15	15	0.000
FIR6.txt (6 *;1 +;)	21	15	28.571
FIR7.txt (7 *;1 +;)	22	22	0.000
FIR8.txt (1 *;2 +;)	22	22	0.000
FIR9.txt (2 *;2 +;)	18	18	0.000
Average			**3.904**

Benchmark: IIR

Table 45: Temperature - IIR

Resource Bag	Initial T	Final T	%
I1.txt (1 +;1 *;)	23	23	0.000
I10.txt (2 +;4 *;)	32	21	34.375
I11.txt (2 +;5 *;)	33	21	36.364
I12.txt (2 +;6 *;)	36	20	44.444
I13.txt (3 +;1 *;)	38	38	0.000
I14.txt (3 +;2 *;)	33	21	36.364
I15.txt (3 +;3 *;)	45	30	33.333
I16.txt (3 +;4 *;)	40	28	30.000
I17.txt (3 +;5 *;)	41	28	31.707
I18.txt (3 +;6 *;)	43	27	37.209
I2.txt (1 +;2 *;)	19	19	0.000
I3.txt (1 +;3 *;)	32	27	15.625
I4.txt (1 +;4 *;)	25	21	16.000
I5.txt (1 +;5 *;)	33	21	36.364
I6.txt (1 +;6 *;)	31	21	32.258
I7.txt (2 +;1 *;)	30	30	0.000
I8.txt (2 +;2 *;)	29	21	27.586
I9.txt (2 +;3 *;)	39	29	25.641
Average			**24.293**

Benchmark: LMS

Table 46: Temperature - LMS

Resource Bag	Initial T	Final T	%
LM1.txt (1 +;1 *;)	23	23	0.000

LM10.txt (3 +;2 *;)	33	27	18.182
LM11.txt (3 +;3 *;)	39	34	12.821
LM12.txt (3 +;4 *;)	31	26	16.129
LM13.txt (4 +;1 *;)	47	47	0.000
LM14.txt (4 +;2 *;)	39	29	25.641
LM15.txt (4 +;3 *;)	46	29	36.957
LM16.txt (4 +;4 *;)	39	33	15.385
LM2.txt (1 +;2 *;)	20	20	0.000
LM3.txt (1 +;3 *;)	32	32	0.000
LM4.txt (1 +;4 *;)	21	21	0.000
LM5.txt (2 +;1 *;)	28	28	0.000
LM6.txt (2 +;2 *;)	26	26	0.000
LM7.txt (2 +;3 *;)	31	31	0.000
LM8.txt (2 +;4 *;)	26	24	7.692
LM9.txt (3 +;1 *;)	39	39	0.000
Average			**8.300**

Benchmark: nestor2

Table 47: Temperature - nestor2

Resource Bag	Initial T	Final T	%
nest1.txt (1 +;1 -;1 *;)	49	49	0.000
nest10.txt (4 +;1 -;1 *;)	59	49	16.949
nest11.txt (4 +;1 -;2 *;)	53	44	16.981
nest12.txt (4 +;1 -;4 *;)	63	60	4.762
nest13.txt (4 +;1 -;6 *;)	72	69	4.167
nest14.txt (4 +;1 -;8 *;)	83	75	9.639
nest15.txt (1 +;2 -;1 *;)	46	46	0.000
nest16.txt (1 +;2 -;2 *;)	51	46	9.804
nest17.txt (1 +;2 -;4 *;)	54	51	5.556
nest18.txt (1 +;2 -;6 *;)	58	54	6.897
nest19.txt (1 +;2 -;8 *;)	64	55	14.063
nest2.txt (1 +;1 -;2 *;)	51	47	7.843
nest20.txt (1 +;4 -;1 *;)	57	53	7.018
nest21.txt (1 +;4 -;2 *;)	50	46	8.000
nest22.txt (1 +;4 -;4 *;)	61	50	18.033
nest23.txt (1 +;4 -;8 *;)	75	57	24.000
nest24.txt (2 +;2 -;2 *;)	52	50	3.846
nest25.txt (2 +;2 -;4 *;)	49	43	12.245
nest26.txt (2 +;2 -;8 *;)	51	45	11.765
nest27.txt (4 +;4 -;8 *;)	77	74	3.896

nest3.txt (1 +;1 -;4 *;)	54	48	11.111
nest4.txt (1 +;1 -;8 *;)	57	50	12.281
nest5.txt (2 +;1 -;1 *;)	55	51	7.273
nest6.txt (2 +;1 -;2 *;)	51	48	5.882
nest7.txt (2 +;1 -;4 *;)	53	51	3.774
nest8.txt (2 +;1 -;6 *;)	54	53	1.852
nest9.txt (2 +;1 -;8 *;)	62	61	1.613
Average			**8.491**

Benchmark: wavelet

Table 48: Temperature - wavelet

Resource Bag	Initial T	Final T	%
w1.txt (1 +;1 -;1 *;)	27	27	0.000
w10.txt (2 +;1 -;4 *;)	31	26	16.129
w11.txt (2 +;1 -;5 *;)	33	27	18.182
w12.txt (2 +;1 -;6 *;)	31	31	0.000
w13.txt (2 +;2 -;1 *;)	43	34	20.930
w14.txt (2 +;2 -;2 *;)	38	30	21.053
w15.txt (2 +;2 -;3 *;)	31	25	19.355
w16.txt (2 +;2 -;4 *;)	38	26	31.579
w17.txt (2 +;2 -;5 *;)	39	27	30.769
w18.txt (2 +;2 -;6 *;)	36	31	13.889
w2.txt (1 +;1 -;2 *;)	22	22	0.000
w3.txt (1 +;1 -;3 *;)	20	20	0.000
w4.txt (1 +;1 -;4 *;)	27	19	29.630
w5.txt (1 +;1 -;5 *;)	30	25	16.667
w6.txt (1 +;1 -;6 *;)	28	28	0.000
w7.txt (2 +;1 -;1 *;)	34	34	0.000
w8.txt (2 +;1 -;2 *;)	30	30	0.000
w9.txt (2 +;1 -;3 *;)	25	25	0.000
Average			**12.121**

Table 49: Temperature - All Benchmarks

Resource Bag	Maximum	Average
4pDCT	25.000	8.294
AR	20.635	3.209
DCT1	18.841	9.321
DIFFEQ2	30.769	11.773
DIFFEQ4	22.581	7.209
EWF	10.638	5.420
FIR	28.571	3.904
IIR	44.444	24.293
LMS	36.957	8.300
nestor2	16.981	8.491
Wavelet	31.579	12.121
Average	**26.091**	**9.303**
Maximum	**44.444**	**24.293**

5.3 Analysis of the Results

From the results we got for both techniques we realized that no room for improvement if only one FU of a type exists.

The higher the number of FUs, the bigger the room for improvement:

- The value of reduced SSA increases when the resource bag increases because this provides a higher possibility to switch bound operations between available resources.
- SSA increases with the resource bag.
- If the resource bag contains FUs more than the number needed in ASAP, they might be used, but the limit will not decrease anymore because ASAP gives the max parallelism.
- Area of FUs will be bigger, but area of muxes is smaller because less reusability!
- Power can increase if # of FUs increases because the FU might be busy executing spurious operations while it can execute non-spurious operations.
- When increasing the number of resources, the number of SA increases too. This is because the available resources will be used, and not the optimal minimal number, and hence this will result in an increase in the number of SA occurring at the inputs of each FU.
    - I noticed that when increasing the size of the resource bag, both SA & SSA increase, regardless of the % of improvement, the number of SA & SSA is higher than when having a smaller resource bag.
- If operations are spread over all the available FUs, less temperature is dissipated.

CHAPTER SIX
CONCLUSION

Designing for lower power & temperature involves tradeoffs. In order to achieve a power and temperature aware design, designers must carefully weigh performance, ease-of-use, cost, density, and power when conducting FPGA design [32].

In this work, we focused on a tradeoff between power and temperature. For this purpose, we developed two new approaches: the first one to reduce the power consumption in a circuit by reducing the power consumed by the functional units which accounts for a large fraction of the overall data-path power budget. The algorithm developed here focus on reducing the activity of the functional units by minimizing the transitions at their input operands.

The second developed approach was the temperature aware functional unit binding in order to avoid hot spots. The algorithm developed here focus on separating the operations bound to a functional unit in order to give it the time needed to cool down between two successive operations.

To test the two approaches proposed in this work, a new synthesis tool was developed. As explained in chapter 3, the developed environment handles both normal and optimized synthesis flows. For the optimization step, the heuristic simulated annealing process was used which tries to converge to an optimal solution by minimizing the switching and temperature costs.

After applying this new approach to 11 benchmarks, we concluded that some benchmarks have better room for improvement than others. Results showed that, on average, spurious switching activity is reduced by 15.4 % which in turn reduces total

switching by 8.7%. Keeping in mind that no area overhead was induced, instead, a 5.3% average area improvement in the area of multiplexers resulted after switching the way operations were bound to functional units.

Using a combination of both techniques: the power-aware approach proposed in this work, and the work done in [1]: register binding optimization for power reduction, and running them simultaneously; results have shown that on average, the total power savings due to reducing switching activity were **12.07%** while spurious switching activity was reduced by an average of **21.8%**.

The results of the temperature-aware functional unit binging showed that, on average, the temperature was reduced by 9.3 % with a maximum reduction of 44%.

Area optimization work done in [33] aimed at decreasing the size or number of multiplexers by swapping operations over the available functional units. Decreasing the size or number of multiplexers needed means to change the interconnections between the functional units and the registers/muxes, hence introduce new room for switching activities which can be spurious sometimes.

For a lower rate of spurious switching activities at the inputs of functional units, having more registers and multiplexers makes it easier to control the inputs at the left and right sides of the functional units, and hence enable the change of inputs when needed. This will be at the cost of area increase. There should be a tradeoff between area, on one side, and power and temperature on the other side.

A possible future work could focus on a combined area, power & temperature aware design where the weighted cost is given by:

$$Cost = \alpha Cost_{(area)} + \beta Cost_{(power)} + \gamma Cost_{(temp)}$$

Where $\alpha + \beta + \gamma = 1$.

The cost for the power is computed as in our proposed power-aware approach, and the cost for the temperature is computed as in our proposed temperature-aware approach.

The values of α & β & γ will be set according to the need: if area is favored over the power and temperature, then α ↗ and β, γ ↘ and vice versa.

References

[1] E. Elaaraj, “A novel approach to reduce spurious switching activity in high-level synthesis.” M.S. thesis, Lebanese American University, Lebanon, 2009.

[2] H.-Ch. Dahmen and U. Glaser, "The impact of area optimization for the power consumption of controllers," in *24th EUROMICRO Conference*, vol. 1, 1998, pp. 204-207.

[3] C. N. Marimuthu and P. Thangaraj, “Low power high performance multiplier.” *Journal of Programmable Devices, Circuits, and Systems*, vol. 8, pp. 31-38, Dec. 2008.

[4] K-H Chen, K-C Chao, J-S Wang, Y-S Chu, and J-I Guo, “An efficient spurious power suppression technique (SPST) and its applications on MPEG-4 AVC/H.264 transform coding design,” in *Proc. ISLPED*, 2005, pp. 155-160.

[5] C. N.Marimuthu, P. Thangaraj, and R. Aswathy, “Low power shift and add multiplier design.” *International Journal of Computer Science and Information Technology*, vol. 2, pp. 12-22, Jun. 2010.

[6] A. Guyot and S. Abou-Samra, “Low power CMOS digital design,” in *Proc. Int. Conf. Microelectron. (ICM)*, 1998, pp. IP6-IP13.

[7] E. Mussoll and J. Cortadella, “Scheduling and resource binding for low power,” in *Proc. Int. Symp. Syst. Synthesis,* 1995, pp. 104-109.

[8] E. Mussoll and J. Cortadella, “High-level synthesis techniques for reducing the activity of functional units,” in *Proc. ISLPED*, 1995, pp. 99-104.

[9] J. M. Chang and M. Pedram, “Register allocation and binding for low power,” in *Proc. Design Automation*, 1995, pp. 29-35.

[10] D. Chen, J. Cong, Y. Fan, and Z. Zhang, “High-level power estimation and low-power design space exploration for FPGAs,” in *Proc. ASP-DAC*, 2007, pp. 529-534.

[11] L. Shang, R. Dick, and N. Jha, "High-Level Synthesis Algorithms for Power and Temperature Minimization," in *High-level Synthesis: From algorithm to digital circuit*, 1st ed. P. Coussy and A. Morawiec, Ed. New York: Springer-Verlag, 2008, pp. 285-297.

[12] F. N. Najm, "Power estimation techniques for integrated circuits," in *ICCAD*, 1995, pp. 492-499.

[13] D. Dal and N. Mansouri, “A high-level register optimization technique for minimizing leakage and dynamic power,” in *Proc. GLSVLSI*, 2007, pp. 517-520.

[14] A. Sarwar. (1997, June). *CMOS power consumption and C_{pd} calculation* (1st ed.) [Online]. Available: http://focus.ti.com/lit/an/scaa035b/scaa035b.pdf [May 15, 2011].

[15] Y. Lu and V. D. Agrawal, "Leakage and dynamic glitch power minimization using integer linear programming for V^{th} assignment and path balancing," in *Proc. PATMOS*, 2005, pp. 217-226.

[16] K. Skadron, T. Abdelzaher, and M. R. Stan. "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *Proc. HPCA*, 2002, pp. 17-28.

[17] Z. Liu, J. Bian, J. Huang, and Y. Wang, “Fast and efficiently binding of functional units for low power design,” in *Int. Conf. on ASIC*, vol. 1, 2005, pp. 10-13.

[18] R. Mukherjee, S.O. Memik, and G. Memik, “Temperature-aware resource allocation and binding in high-level synthesis," in *Proc. DAC*, 2005, pp.196-201.

[19] D. Chen, J. Cong, and Y. Fan, “Low-power high-level synthesis for FPGA architectures," in *Proc. ISLPED*, 2003, pp.134-139.

[20] Lin Zhong, Jiong Luo, Yunsi Fei and Niraj K. Jha, "Register binding based power management for high-level synthesis of control-flow intensive behaviors," in *Proc. IEEE Int. Conf. on Computer Design (ICCD)*, 2002, pp. 1-2.

[21] T. Yeh and S. Wang, “Thermal safe high level test synthesis for hierarchical testability," in *Proc. Asian Test Symposium*, 2010, pp.337-342.

[22] D. Kannan, A. Shrivastava, S. Bhardwaj, S. Vrudhul, “Power reduction of functional units considering temperature and process variations,” in *21st Int. Conf. on VLSI Design,* 2008, pp. 533–539.

[23] D. Chen and S. Cromar, “An optimal resource binding algorithm with inter-transition switching activities for low power.” *Journal of Low Power Electronics,* vol. 5, no. 4, pp. 454-463, Dec. 2009.

[24] D. Chen, J. Cong, and J. Xu, “Optimal module and voltage assignment for low-power,” in *Proc. of the ASP-DAC*, 2005, pp. 850-855.

[25] M. Pedram, “Low power design methodologies and techniques: An overview.” Internet: http://atrak.usc.edu/~massoud/Papers/LPD-talk.pdf, 1999 [May 30, 2011].

[26] G. Lakshminarayana, A. Raghunathan, N.K. Jha, and S. Dey, "Power management in high-level synthesis." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp.7-15, Mar. 1999.

[27] H. B. Kim, “High-level synthesis and implementation of built-in self-testable data path intensive circuit.” Ph.D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, 1999.

[28] A. Avakian, “High-level synthesis optimization of area.” Senior Project, Lebanese American University, Lebanon, 2005.

[29] A. Hashimoto and J. Stevens, “Wire routing by optimizing channel assignment within large apertures,” in *Proc. 8th Design Automation Workshop*, 1971, pp. 155-163.

[30] C. Tseng and D.P. Siewiorek, “Automated synthesis of data paths in digital systems.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no.3, pp. 379-395, Jul. 1986.

[31] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis*. Boston, MA: Kluwer Academic Publishers, 1992, pp. 200-215.

[32] M. Khan, "Power optimization in FPGA designs," Internet: http://www.altera.com/literature/cp/cp-pwropt.pdf, *SNUG San Jose*, 2006 [Jun. 1, 2011].

[33] L. Bassil, “High-level synthesis: Optimization of area through optimization of functional unit binding and force-directed scheduling,” Senior Project, Lebanese American University, Lebanon, 2005.

[34] R.V. Menon, S. Chennupati, N.K. Samala, D. Radhakrishnan, and B.A. Izadi, "Switching activity minimization in combinational logic design,” in *Proc. ESA/VLSI*, 2004, pp.47-53.

[35] M. C. McFarland, A. C. Parker, and R. Camposano, "Tutorial on high level synthesis," in *Proc. 25th Design Automation*, 1988, pp. 330-336.

[36] V. K. Srikantam, N. Ranganathan, and S. Srinivasan, “CREAM: combined register and module assignment with floorplanning for low power datapath synthesis,” in *Proc. VLSI Design*, 2000, pp. 228-233.