5-2004

Automated Exploration of the ASIC Design Space for Minimum Power-Delay-Area Product at the Register Transfer Level

Fuat Karakaya
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Fuat Karakaya entitled "Automated Exploration of the ASIC Design Space for Minimum Power-Delay-Area Product at the Register Transfer Level." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

Donald W. Bouldin, Major Professor

We have read this dissertation and recommend its acceptance:

Gregory D. Peterson, Chandra Tan, Michael A. Langston

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Fuat Karakaya entitled " Automated Exploration of the ASIC Design Space for Minimum Power-Delay-Area Product at the Register Transfer Level." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

Dr. Donald W. Bouldin

Major Professor

We have read this dissertation
and recommend its acceptance:

Dr. Gregory Peterson

Dr. Chandra Tan

Dr. Michael A. Langston

Accepted for the Council:

Dr. Anne Mayhew

Vice Provost and
Dean of The Graduate Studies

(Original signatures are on file with official student records.)

AUTOMATED EXPLORATION OF THE ASIC DESIGN SPACE FOR MINIMUM POWER-DELAY-AREA PRODUCT AT THE REGISTER TRANSFER LEVEL

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Fuat Karakaya

May, 2004

DEDICATION

Dedicated to

my wife Selda, my daughters Hatice Nur, and Zeynep Hannan for their love and support

ACKNOWLEDGEMENT

First of all, I would like to thank Professor Donald W. Bouldin, my advisor, for his guidance, encouragement, support and valuable criticism during this research. I also thank my other committee members, Dr. Michael A. Langston, Dr. Gregory Peterson and Dr. Chandra Tan for their guidance and valuable feedback. I would also like to acknowledge the Defense Advanced Research Projects Agency for providing partial financial support for this research under grant F30602-01-2-0562.

ABSTRACT

Exploring the integrated circuit design space for minimum power-delay-area (PDA) product can be time-consuming and tedious, especially when the target standard-cell library has hundreds of options. In this dissertation, heuristic algorithms that automate this process have been developed, implemented and validated at the register transfer level. In some cases, the PDA product was 1.9 times better than the initial baseline solution. The parallel search algorithm exhibited 9x speed up when executed on 10 machines simultaneously. These two new methods also characterize the design space for the given RTL code by generating power-delay-area points in addition to the minimum PDA point in case the designer wishes to select a different solution that is a tradeoff among these metrics. As a final step, these two search algorithms are integrated into a fully automated ASIC design flow.

TABLE OF CONTENTS

CHAPTER **PAGE**

1. Introduction **1**

1.1 Motivation 3

1.2 Goals and Expected Contributions 9

1.3 Thesis Overview 11

2. Background **12**

2.1 Power Consumption in CMOS 12

2.1.1 Dynamic Power 14

2.1.2 Static Power 16

2.2 Delay in CMOS 17

2.3 Optimization in CMOS Circuits 20

2.3.1 Power Optimization 20

2.3.2 Delay Optimization 24

2.4 ASIC Design Flow 31

2.5 Related Work 33

3. Design Space Search Algorithm: DSSA **38**

3.1 Implementation of DSSA 38

3.2 Integrating DSSA into an ASIC Flow 44

3.3 Experimental Results for DSSA 46

CHAPTER PAGE

3.4 Parallel Design Space Search Algorithm: PDSSA 54

3.4.1 Implementation of PDSSA 54

3.4.2 Experimental Results for PDSSA 56

3.5 Pareto Points 57

3.6 Chapter Conclusion 59

4. Searching the Design Space Using Simulated Annealing 66

4.1 Simulated Annealing 66

4.2 Implementation 68

4.3 Experimental Results 70

4.4 Chapter Conclusion 75

5. Graphical User Interface (GUI) Design 76

5.1 MacroGen 76

5.2 Initiating Design Space Search 80

5.3 Reporting and Analyzing Run Results 85

5.4 Initiating Layout 91

6. Summary and Future Work 96

BIBLIOGRAPHY 99

VITA 102

LIST OF TABLES

TABLE **PAGE**

1.1 The effect of parameter optimization on GOPS/W and GOPS/μ^2 . . . 9

3.1 Characteristics of the macros 46

3.2 Default values of delay and PDAP for the macros 47

3.3 Optimum values of delay and PDAP for the macros 47

3.4 Number of generated points and run times 54

3.5 Run times for DSSA and PDSSA 57

3.6 Effect of step-size on PDAP (for multiplier) 59

4.1 A general SA algorithm in pseudo code 67

4.2 Characteristics of the macros 70

4.3 Constraints of the macros 71

4.4 PDA products 71

4.5 PDAP improvement ratios 71

4.6 Run times 75

LIST OF FIGURES

FIGURE		PAGE
1.1	Design level, estimation accuracy and estimation speed.	2
1.2	3D design-space	4
1.3	Delay vs. area trade-off curve.	5
1.4	Power vs. area trade-off curve.	6
1.5	Power vs. delay trade-off curve	7
1.6	GOPS/W-PDA/1000 vs. delay curve	10
2.1	Current flow in a CMOS inverter.	13
2.2	CMOS inverter while charging and discharging a load.	14
2.3	Two-stage inverter and its equivalent circuit for delay calculations.	18
2.4	Power optimization methods.	21
2.5	Before and after clock gating.	23
2.6	Before and after operand isolation.	25
2.7	Sample switching activity file.	26
2.8	Inserting buffer chain.	28
2.9	Before and after pipelining	30
2.10	ASIC design flow.	32
2.11	Monte-Carlo design space exploration flow	34
3.1	DSSA in pseudo code - Part I	39
3.2	DSSA in pseudo code - Part II	40

FIGURE **PAGE**

3.3 Default ASIC design flow 41

3.4 ASIC design flow with optimization 43

3.5 DSSA integrated into an ASIC flow. 45

3.6 Searched space for 16-bit adder 48

3.7 Searched space for 16-bit multiplier 49

3.8 Searched space for 16-bit complex-multiplier 50

3.9 Searched space for 12-bit 32 tab FIR 51

3.10 Searched space for 12-bit 4 pole 32 tab poly FIR 52

3.11 Searched space for 12-bit 64-point FFT 53

3.12 PDSSA in pseudo code 55

3.13 A sample shell script generated by PDSSA for parallel search 56

3.14 Perl code to determine the Pareto points. 58

3.15 Pareto curve for 16-bit adder. 60

3.16 Pareto curve for 16-bit multiplier. 61

3.17 Pareto curve for 16-bit complex-multiplier. 62

3.18 Pareto curve for FIR. 63

3.19 Pareto curve for poly-FIR. 64

3.20 Pareto curve for 64 point FFT. 65

4.1 Searched space for adder 72

4.2 Searched space for multiplier 73

4.3 Searched space for complex-multiplier 74

5.1 Main window of the GUI. 77

FIGURE **PAGE**

5.2 Initiating MacroGen from the main window. 78

5.3 Selecting a macro from MacroGen. 79

5.4 Viewing specifications for the selected macro. 81

5.5 Entering values for the generic parameters of the selected macro. . . . 82

5.6 Viewing VHDL file of the selected macro. 83

5.7 Initiating a design space search. 84

5.8 Design space search in progress. 86

5.9 Tabulated design space search results for the selected macro. 87

5.10 Plotting search results for the selected parameters. 88

5.11 Generating Pareto curve. 89

5.12 Viewing the statistics of the standard-cells, number of transistors for the selected design point. 90

5.13 Selecting a design point to send for layout. 92

5.14 Initiating place and route procedure. 93

5.15 Selected design point after placement and routing. 94

5.16 Final layout for the selected design point. 95

CHAPTER 1

Introduction

The need for integrated circuits that consume less power and area yet are faster than unoptimized circuits has accelerated the development of new and powerful synthesis tools. Synthesis tools and the state-of-the-art standard-cell libraries are some of the crucial parts of successful application-specific integrated circuit (ASIC) design. ASIC design can be summarized in 3 major phases [1]:

1. Description of the design in a process-independent hardware description language (HDL) such as VHDL or Verilog at either the behavioral level or at the register transfer level (RTL).

2. Mapping the HDL to a specific process to generate a process-dependent net-list.

3. Physical placement and routing of the design.

It is possible to optimize the design during each of these phases to meet the delay, power, and area specifications. In the first phase, the HDL code can be optimized with the help of HDL power estimation tools [19] (however, the estimates may not be very accurate; see Figure 1.1). A statistical design space exploration at the behavioral level can also be conducted [10]. A successful optimization during the second phase depends on how well the targeted standard-cell library has been characterized, the ca-

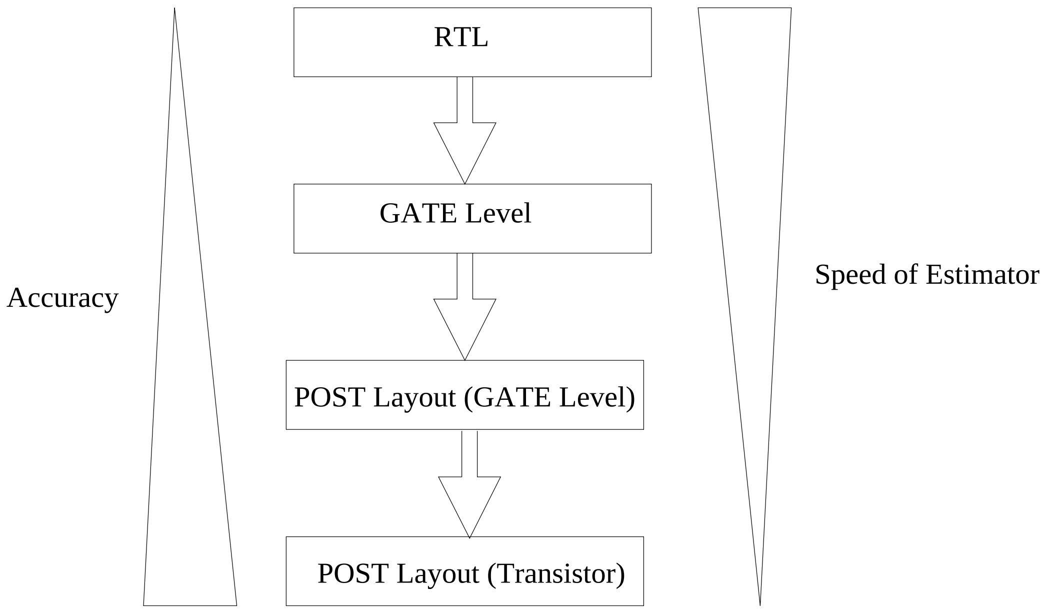


Figure 1.1: Design level, estimation accuracy and estimation speed.

pabilities of the synthesis tool and the optimization procedure used. Today's standard-cell libraries contain several versions of the same cell with different drive strengths [5], which makes it easy to target the same design with different sets of constraints and find a solution. Because of this fact, there are several different power-delay-area solutions for the design. Design space exploration using RTL involves generating a set of power-delay-area combinations between the lower and the upper boundaries of the design space (an example 3-D design space is illustrated in Figure 1.2 which was generated for a 16-bit multiplier). As a result, it is almost impossible or very difficult to explore the design space manually. Even by using semi-automated search techniques, thorough design space exploration can be very time-consuming (taking hours or even days). The main contribution of this work is a fully automated search technique which explores the design space. This technique does not require any initial target values for power, delay, and area. It automatically determines the upper and lower bounds of the design-space for the targeted process and explores the design space to minimize a given cost function.

1.1 Motivation

Designing efficient chips in terms of power, speed, and silicon area is the principal goal of an ASIC designer. But as seen from Figure 1.3, Figure 1.4 and Figure 1.5 (these figures were generated experimentally using a 16 bit multiplier), designing a higher speed circuit means making sacrifices in power consumption and in silicon area, or designing a low-power circuit means making a sacrifice in speed. As a result,

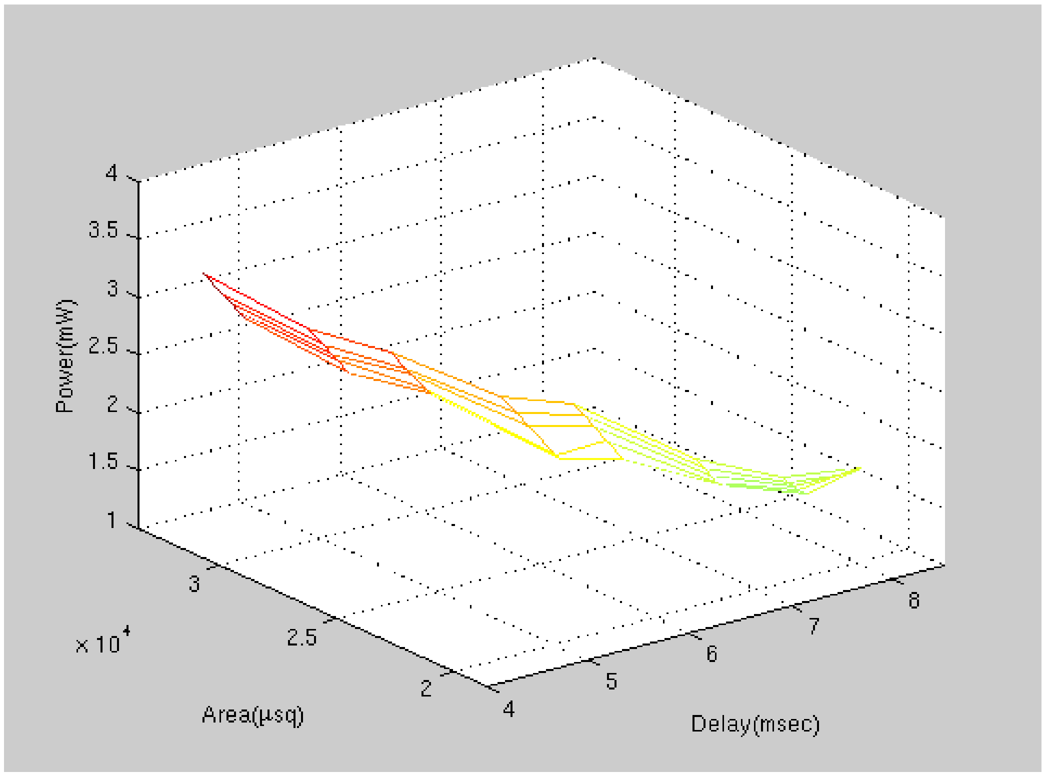


Figure 1.2: 3D design-space

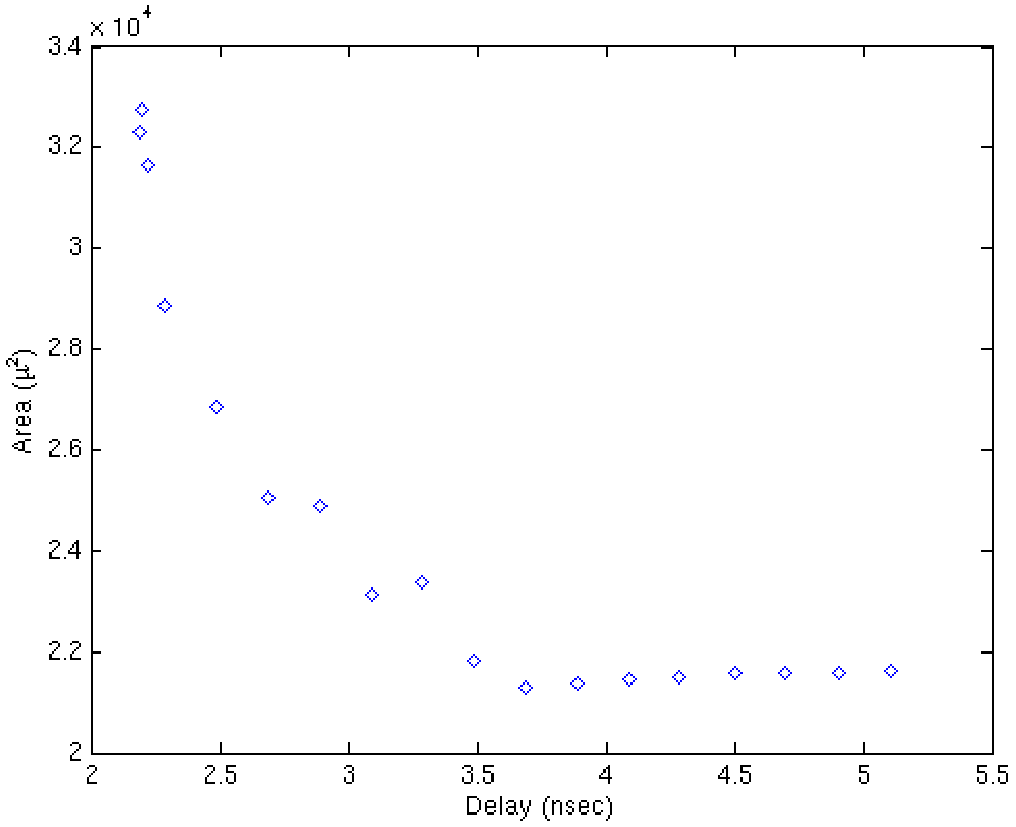


Figure 1.3: Delay vs. area trade-off curve.

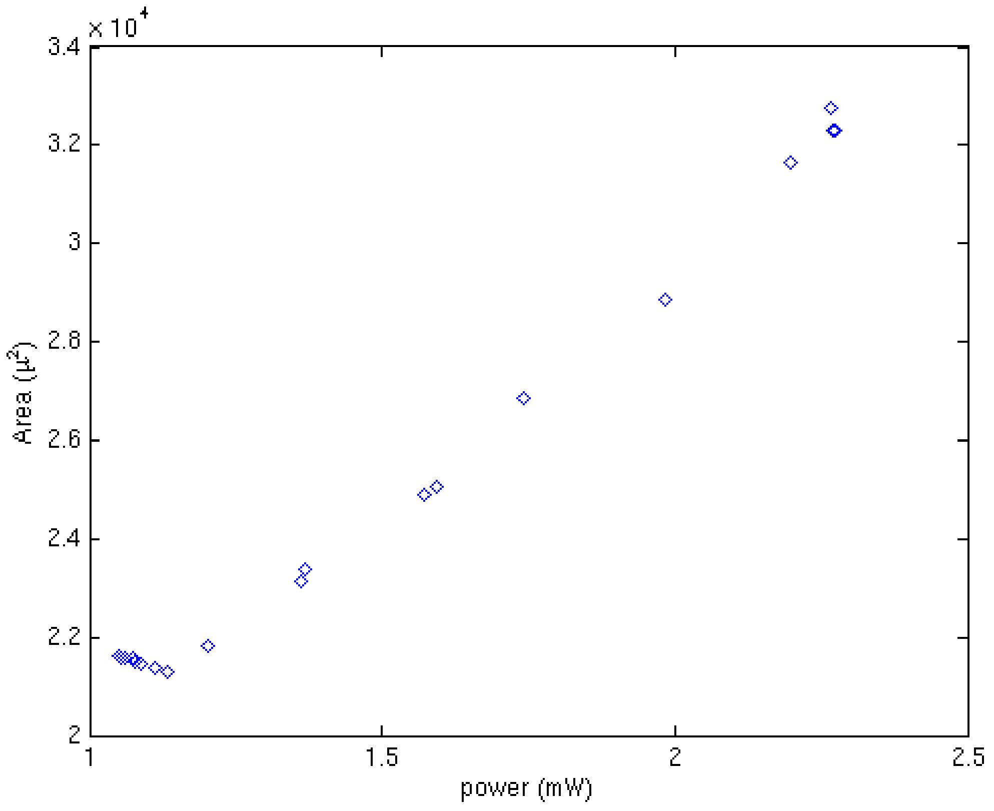


Figure 1.4: Power vs. area trade-off curve.

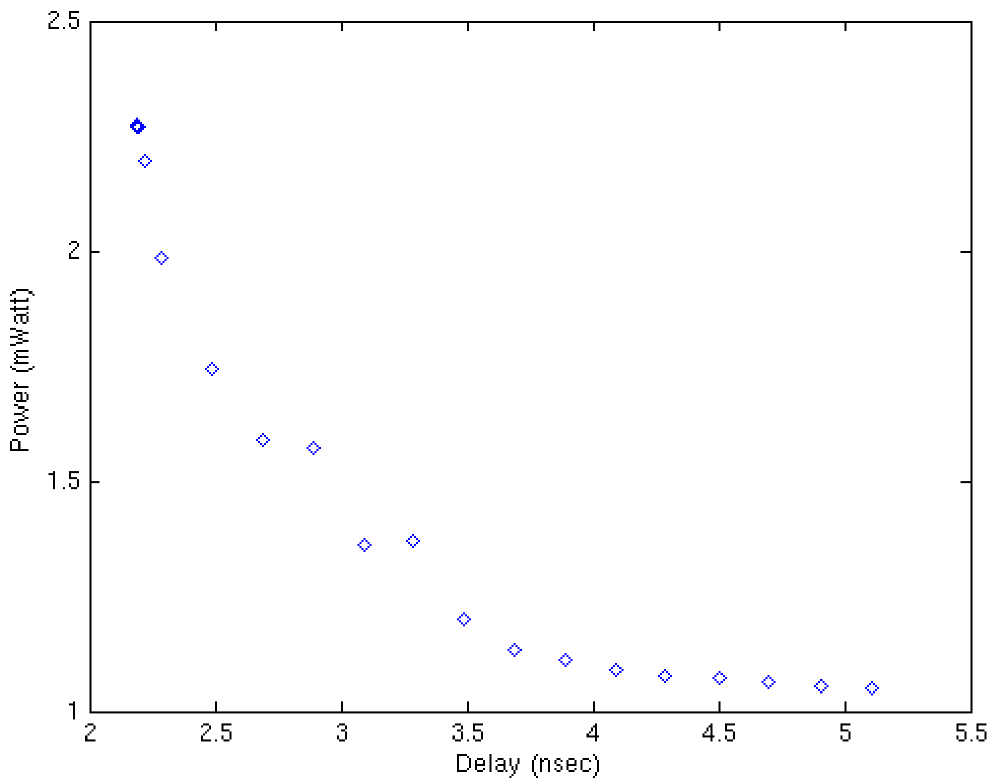


Figure 1.5: Power vs. delay trade-off curve

the designer must constantly make trade-offs between conflicting requirements while looking for a solution that satisfies the design constraints. The designer will make the trade-offs depending on the purpose of the design. If it is designed for wireless applications, power and the portability will be the main focus. On the other hand if it is designed for real-time applications speed may be the main focus. But some cases require attention for power, speed and portability, such as in tactical missile applications. The ASIC must be compact enough to fit into a confined space with limited capability to remove the heat generated during high-speed processing. Thus, low power consumption is of equal concern with processing speed and area so that optimizing the design for minimum PDA will be the best choice. Two important parameters to measure integrated circuit efficiency are operations per second per watt (OPS/W) and operations per second per square micron (OPS/μ^2). The first parameter is an indication of energy efficiency of the chip and the second parameter is an indication of the area efficiency of the chip. Both parameters are closely related to the power, delay and area values of the design. OPS/W is inversely proportional to the product of delay and power (DPP), and OPS/μ^2 is inversely proportional to the product of area and delay. To increase operations per second per watt, DPP has to be minimized but at the same time silicon area also has to be kept in mind. Thus, minimization of the product of power times delay times area (PDA) can be a more efficient means of increasing OPS/W without hurting OPS/μ^2 efficiency.

Table 1.1, generated experimentally using a 16 bit multiplier, shows that optimizing the design for PDAP gives the best GOPS/W with an insignificant effect on GOPS/μ^2.

Table 1.1: The effect of parameter optimization on GOPS/W and GOPS/μ^2

Optimized parameter	GOPS/W	GOPS/μ^2
D	200	14.5
P	190	9.05
A	187	12.5
PDA	240	12.7

Figure 1.6, generated experimentally using a 16 bit multiplier, indicates that maximum GOPS/W occurs when the PDA is minimum.

To implement an efficient chip, the design space has to be searched for minimum PDAP. However, this process is not an easy task and today's very sophisticated commercial tools do not provide an automated technique to meet this need. Hence, there is a need for a technique which can work with commercial tools and guide them to search the design space for minimum PDAP. Not only should the technique provide the means to achieve this goal, it should also be integrated into an automated ASIC design flow to reduce design time.

1.2 Goals and Expected Contributions

The main objective of this dissertation is to develop techniques which make design space search easier or possible for certain parameters of the design. The emphasized parameter in this thesis will be the product of power-delay-area or PDAP. Moreover,

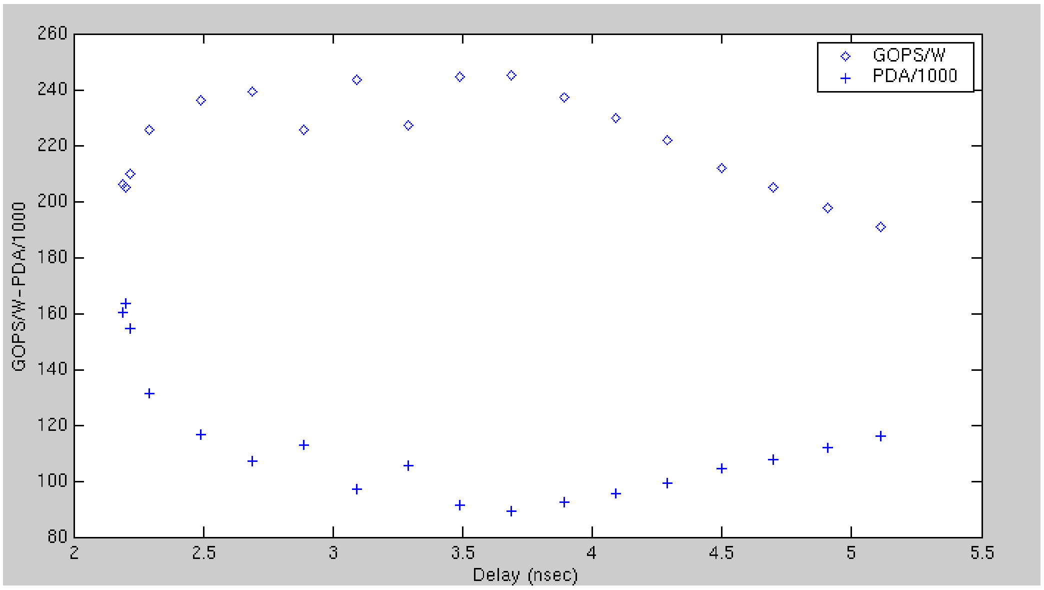


Figure 1.6: GOPS/W-PDA/1000 vs. delay curve

the tools will be capable of handling other parameters like PAP, PDP, DAP etc.. Thus, the goals of this research can be itemized as follows:

1. Develop a technique to assist or to guide a commercial tool to do a design-space search.
2. Perform the design-space search with user guidance in terms of:
    (a) search time,
    (b) number of iterations, and/or
    (c) improvement ratio.
3. Integrate the search technique into an automated ASIC design flow.
4. Design a graphical user interface (GUI) for the automated flow.

1.3 Thesis Overview

In Chapter 2, background about ASIC flow, CMOS circuits (power consumption, delay, etc.), power optimization techniques, delay optimization techniques is given. The related work section is also given in this chapter. Implementation of the proposed design space search algorithms (DSSA, PDSSA) and experimental results are given in Chapter 3. Chapter 4 presents the application of simulated annealing to design space search and some experimental results of the application. Designing a graphical user interface (GUI) for the developed algorithms is given in Chapter 5. Finally Chapter 6 contains the conclusions and future work.

CHAPTER 2

Background

2.1 Power Consumption in CMOS

When the inverter in Figure 2.1 is functioning, it consumes two types of power:

1. Dynamic power
   (a) Power due to charging/discharging of capacitive load and parasitic capacitances (which is also known as switching power).
   (b) Short-circuit power
2. Static power
   (a) Leakage power
   (b) Sub-threshold conductance

Total power consumption can be formulated as following [11]:

$$P_{total} = \frac{1}{2} * \alpha * C_L * V_{DD}^2 * f + I_{sc} * V_{DD} + I_{leakage} * V_{DD} \tag{2.1}$$

The following subsections explain each of these classes of power consumption.

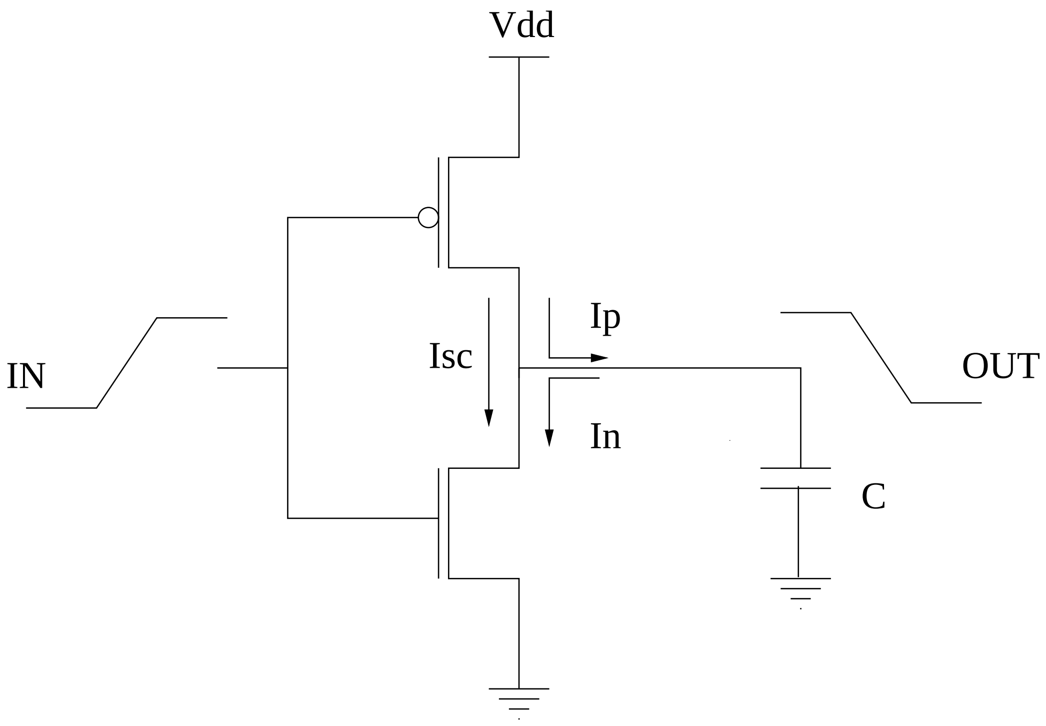


Figure 2.1: Current flow in a CMOS inverter.

2.1.1 Dynamic Power

Switching Power

Dynamic dissipation due to capacitive switching often consumes 80-90% of the total power [17]. Dynamic dissipation is the result of charging and discharging parasitic capacitances in the circuit. Figure 2.2 represents the equivalent circuit of a CMOS inverter while charging and discharging a load capacitance in which the load capacitance represents the total lumped parasitic capacitances. When the input to the inverter goes from 1 to 0 the output will go from 0 to 1. For that phase (Figure 2.2) Equation 2.2 to Equation 2.6 show the calculations to determine the stored energy in the load capacitance [25].

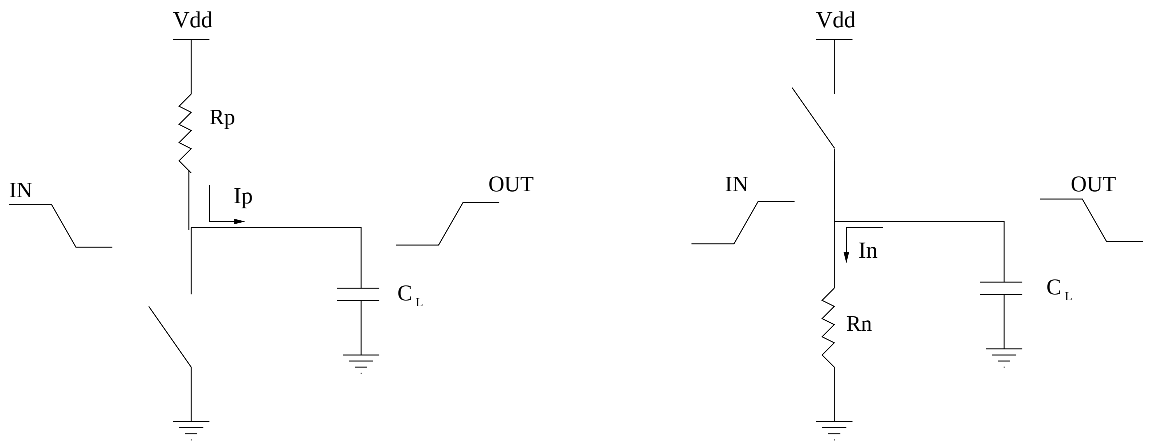


Figure 2.2: CMOS inverter while charging and discharging a load.

$$I_p(t) = \frac{V_{DD}}{R_p} exp(\frac{-t}{R_p C_L}) \tag{2.2}$$

$$V_{C_L}(t) = V_{DD}(1 - exp(\frac{-t}{R_p C_L})) \tag{2.3}$$

$$E_{C_L} = \int_0^\infty I_p(t) V_{C_L}(t) dt \tag{2.4}$$

$$E_{C_L} = C_L V_{DD}^2 (exp(\frac{-t}{R_p C_L}) - \frac{1}{2} exp(\frac{-2t}{R_p C_L}))|_0^\infty \tag{2.5}$$

$$E_{C_L} = \frac{1}{2} C_L V_{DD}^2 \tag{2.6}$$

The total stored energy is given by Equation 2.6. The same amount of energy is dissipated in the PMOS transistor (due to R_p resistor). If the input goes from 0 to 1 then the output will go from 1 to 0 (Figure 2.2). In this case there is a direct path to ground for C_L to discharge. The energy stored (Equation 2.6) in C_L will be dissipated by the NMOS transistor (due to R_n). Fortunately, switching power is only dissipated if a parasitic capacitor is charged or discharged. Thus, the total switching power consumption can be written as:

$$P_{sw} = \frac{1}{2} * \alpha * C_L * V_{DD}^2 * f \tag{2.7}$$

where α is the number of capacitors switched (1 -> 0 or 0 -> 1, switching activity) per clock period, and f is the clock frequency. Equation 2.7 illustrates that the switching power is proportional to switching activity, total lumped parasitic capacitance, and the square of the supply voltage. Note that dynamic switching power consumption is independent of the effective resistances of the transistors.

Short-Circuit Power

At some point during the switching transient, both the NMOS and PMOS transistors are turned on at the same time. This occurs for gate voltages between V_{tn} and V_{dd}+V_{tp}. During this time, there is a short circuit between V_{dd} and ground and a short-circuit current flows through that path (I_{sc}). Short-circuit power (Equation 2.8) generally accounts for 10%-20% of the total power dissipated in CMOS circuits [11].

$$P_{sc} = I_{sc} * V_{DD} \tag{2.8}$$

2.1.2 Static Power

Since CMOS transistors are not ideal switches, there is always a leakage current from V_{dd} to ground through the channel of an off transistor.

2.2 Delay in CMOS

Prior to the advent of sub-micron processes, chip delay was dominated by the rise and fall delays of the CMOS transistors. But in a sub-micron process, interconnect delays become a more significant part of the total delay. Figure 2.3 shows an inverter driving a second inverter as a load. If we know the the effective resistance of the transistor, the delay calculation will be very simple. To calculate a resistor value to represent the transistor over its entire operating range, the transistor's resistance is measured at two operating points (saturation and linear) and averaged [14] as shown in Equation 2.9.

$$R_p = (\frac{V_{sat}}{I_{sat}} + \frac{V_{lin}}{I_{lin}})/2 \tag{2.9}$$

$$I_{sat} = \frac{1}{2} k_p \frac{W}{L} (V_{DD} - V_t)^2 \tag{2.10}$$

$$V_{sat} = V_{DD} \tag{2.11}$$

$$V_{lin} = (V_{DD} - V_t)/2 \tag{2.12}$$

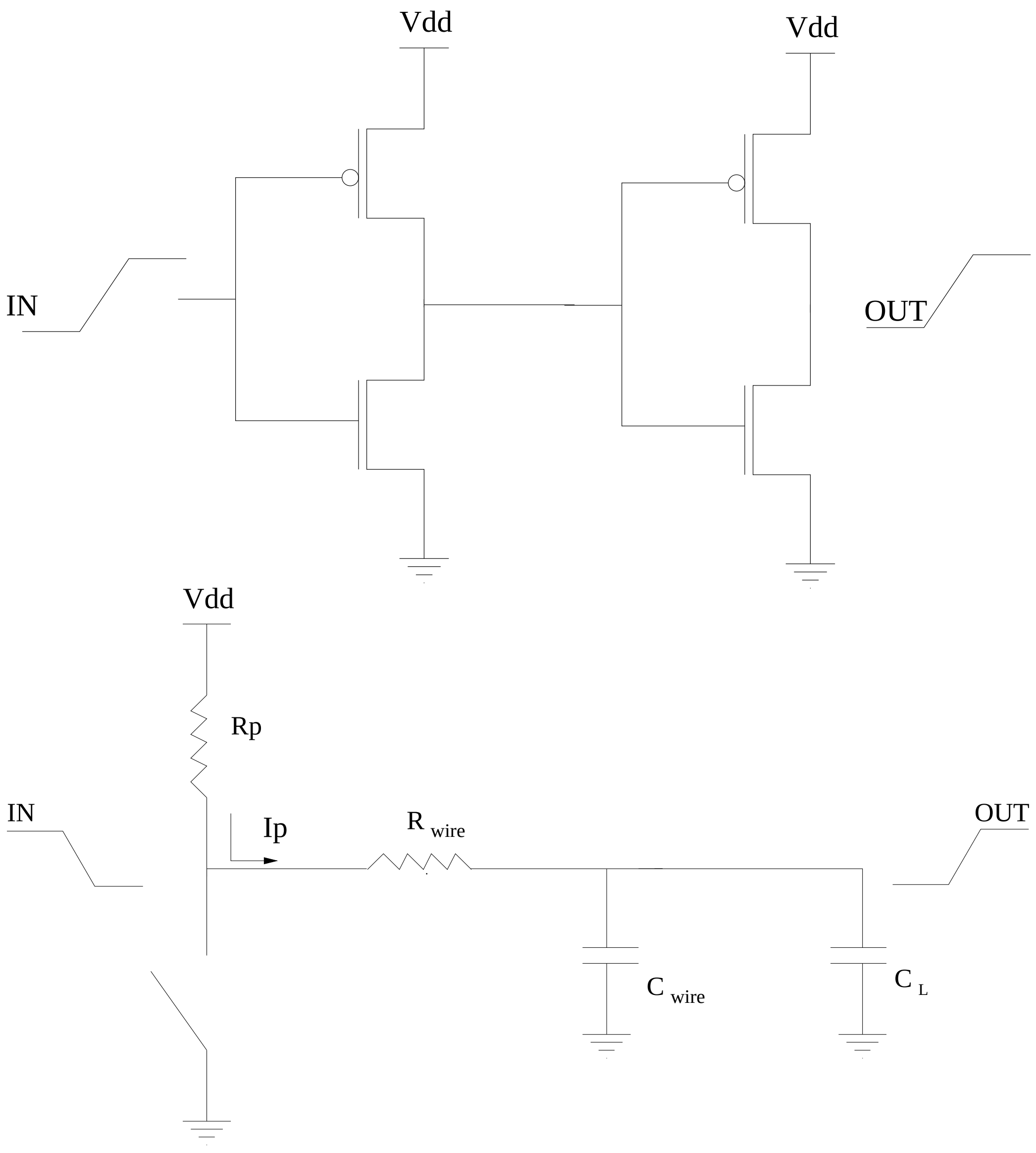


Figure 2.3: Two-stage inverter and its equivalent circuit for delay calculations.

$$I_{lin} = \frac{3}{8} k_p \frac{W}{L} (V_{DD} - V_t)^2 \tag{2.13}$$

After inserting Equation 2.10, Equation 2.11, Equation 2.12, and Equation 2.13, into Equation 2.9 approximately we get;

$$R_p \simeq \frac{1}{k_p} \frac{L}{W} \frac{V_{DD}}{(V_{DD} - V_t)^2} \tag{2.14}$$

$$V_{out}(t) = V_{DD} exp(\frac{-t}{(R_p + R_{wire})(C_{wire} + C_L)}) \tag{2.15}$$

for a rise time corresponding to a change in output voltage from $0.1V_{DD}$ to $0.9V_{DD}$

$$t_{rise} = 2.2(R_p + Rwire)(C_{wire} + C_L) \tag{2.16}$$

$$t_{rise} \simeq 2.2(\frac{1}{k_p} \frac{L}{W} \frac{V_{DD}}{(V_{DD} - V_t)^2} C_{total} + R_{wire} C_{total}) \tag{2.17}$$

As seen from Equation 2.16, delay is inversely proportional to both V_{DD} and transistor size. Increasing the supply voltage or increasing the transistor size results in a faster circuit.

2.3 Optimization in CMOS Circuits

2.3.1 Power Optimization

As the popularity of portable and wireless devices has increased, power consumed by these devices has become a critical issue. They have to be designed to consume low power because of battery life time and reliability issues [18]. As the circuits get hot, their failure rate gets higher. Proper cooling mechanisms have to be installed in the system to prevent failures. But the best way of increasing the battery life and decreasing the failure rate is designing low-power devices. At each design level, a designer can use strategies to lower the power consumption. Figure 2.4 shows the power saving strategies at each design level and possible percentages of power savings by utilizing these strategies. Only the ones applicable at the RTL level will be mentioned in this section, and they are:

1. Clock gating
2. Operand isolation
3. Switching activity back annotation.

Clock Gating

As mentioned earlier, most power consumption occurs when there is switching. If unnecessary switching is prevented, power consumption can be lowered dramatically.

LEVEL	OPTIMIZATION METHODS	POWER REDUCTION
System level	Algorithms, Process, Library Supply Voltage	50-90%
Behavioural Level	Scheduling, Allocation, Resource Sharing & Retiming	40-70%
RTL Level	***Clock-Gating, Operand-Isolation Precomputation, FSM Encoding***	***30-50%***
Gate Lavel	Technology Mapping, Rewiring, Phase Assignment, Lowering Glitching	20-30%
Device Level	Buffering, Transistor Sizing	10-20%
Physical Level	P&R Interconnect Capacity Reduction Clock-Tree Synthesis, Floorplaning	5-10%

Figure 2.4: Power optimization methods [2]

For that purpose, clock-gating [9], [26], [4] can be the answer. The circuit in Figure 2.5(a) will consume switching power even if there is no state or output transition. But after inserting clock-gating circuitry (Figure 2.5(b)), unnecessary switching will be prevented. Clock-gating circuitry consists of a clock enable circuit, a latch, and an AND-gate. The state and input signals are fed into the clock enable circuit which determines whether to enable the clock. After latching the clock enable signal, the system clock and the clock enable signal are passed through an AND-gate. The output of the AND-gate is the GATED-CLOCK signal. Clock gating can reduce the dynamic power up to 40% depending on the design. The latch is in the clock-gating circuitry to prevent glitches in the clock.

Operand Isolation

In a design, data-path operators (multipliers, adders, etc.) are implemented in a way that they are always operational. Even when the output of the operation is not used, these circuits continue to experience switching activity which results in power dissipation. A data-path operator is mainly a combinational circuit. A combinational circuit continues to switch whenever its inputs change value, even if its output is not used. If the data-path operator output is an input to an unselected multiplexer or if it is an input to a register that is currently disabled, its output is not used, even though it continues to switch. This useless switching increases the wasted power consumption. This waste of power can be prevented if the data-path operators are stopped from switching when their output is ignored. This can be done by using the operand-isolation tech-

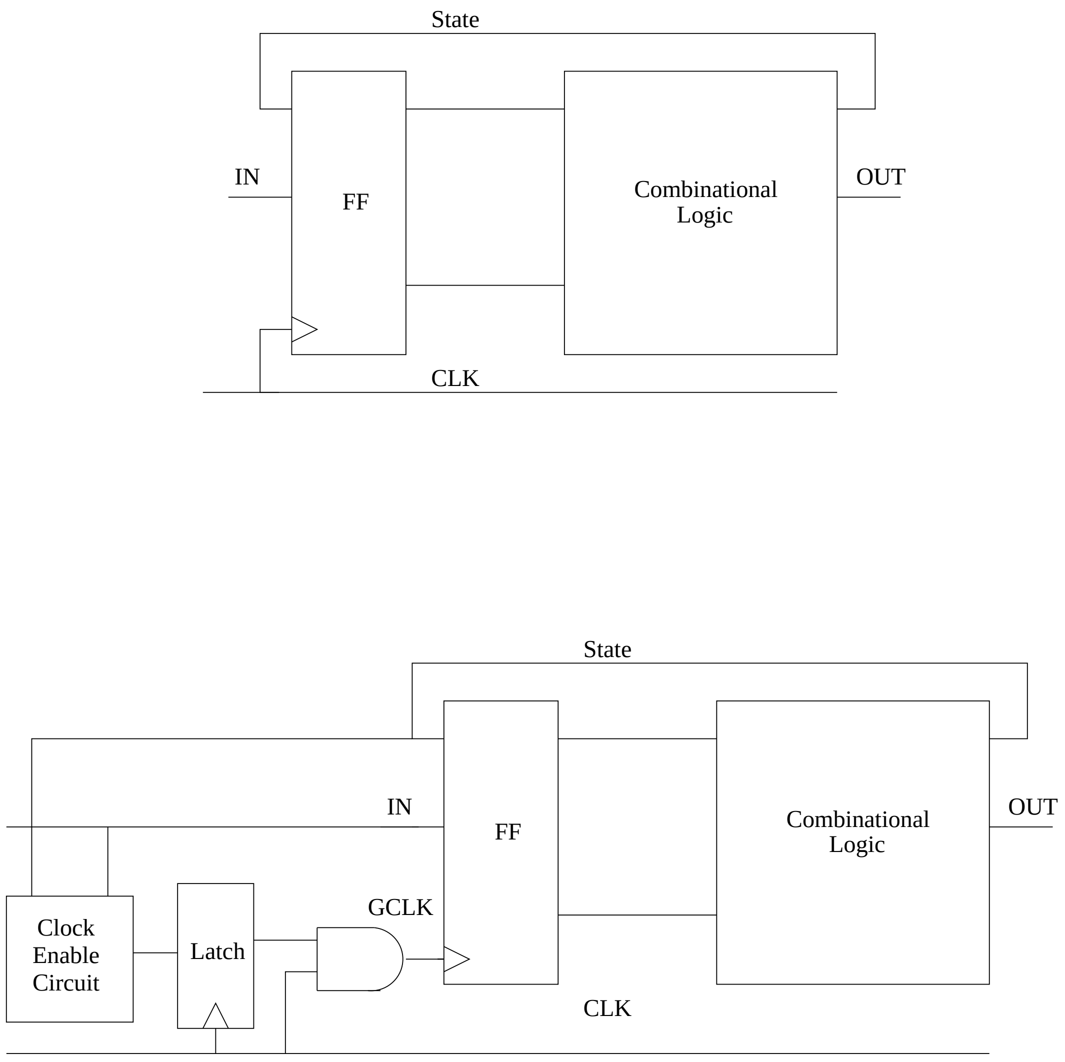


Figure 2.5: Before and after clock gating.

nique [4], [21]. Figure 2.6 shows a design before and after operand-isolation. With the operand-isolation technique, additional logic (AND or OR gates) is inserted along with an activation signal to hold the inputs of the data-path operators stable whenever their output is not used.

Switching Activity Backannotation

It is clear from Equation 2.1 that switching activity has an effect on dynamic power. More switching increases the dynamic power. Switching activity is used by the synthesis tools, and power estimation tools for accurate power optimization and accurate power estimation. A switching activity file can be generated either at the RTL or at the gate-level abstraction. This file contains a list of 0->1 or 1->0 transitions of the internal nodes and the input and output ports for a given time. The switching activity file is utilized by the optimization tools to pinpoint the hot-spots (more power consuming regions) of the design [4]. Power optimization tools put more effort on these hot-spots in order to reduce the power consumption. The switching activity file (Figure 2.7) can be captured using simulation tools. The input vector set, which is used to generate the switching activity file, should characterize a typical operation of the design.

2.3.2 Delay Optimization

Buffer Insertion

Delay through a stage increases as the capacitance driven by that stage increases. Usually each stage drives a load which matches its drive capability. However, there are

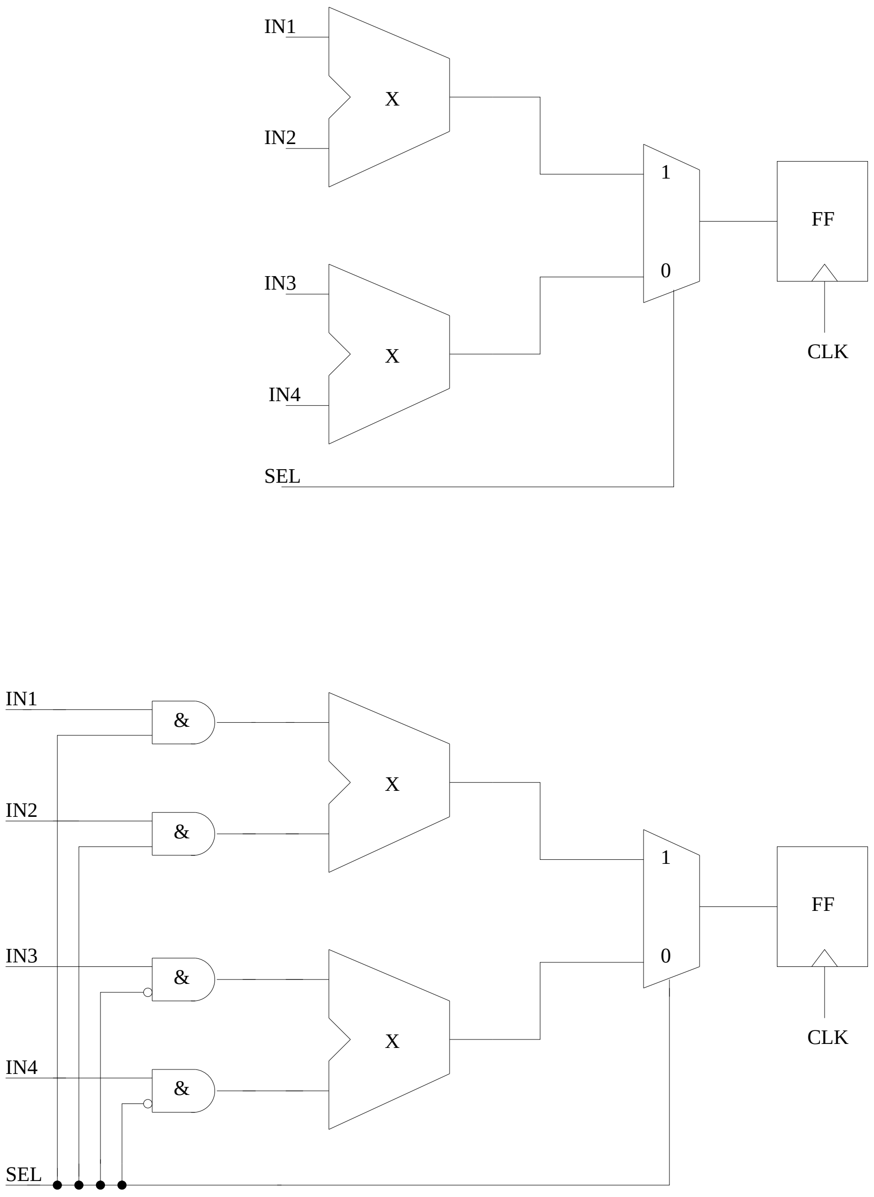
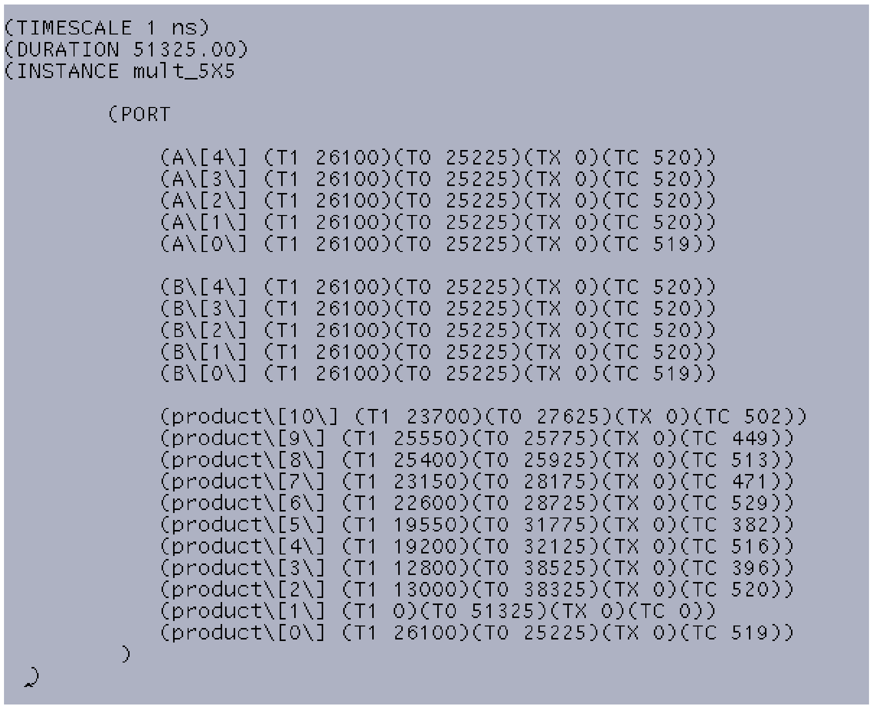


Figure 2.6: Before and after operand isolation.



```
(TIMESCALE 1 ns)
(DURATION 51325.00)
(INSTANCE mult_5X5

        (PORT

            (A\[4\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (A\[3\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (A\[2\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (A\[1\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (A\[0\] (T1 26100)(T0 25225)(TX 0)(TC 519))

            (B\[4\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (B\[3\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (B\[2\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (B\[1\] (T1 26100)(T0 25225)(TX 0)(TC 520))
            (B\[0\] (T1 26100)(T0 25225)(TX 0)(TC 519))

            (product\[10\] (T1 23700)(T0 27625)(TX 0)(TC 502))
            (product\[9\] (T1 25550)(T0 25775)(TX 0)(TC 449))
            (product\[8\] (T1 25400)(T0 25925)(TX 0)(TC 513))
            (product\[7\] (T1 23150)(T0 28175)(TX 0)(TC 471))
            (product\[6\] (T1 22600)(T0 28725)(TX 0)(TC 529))
            (product\[5\] (T1 19550)(T0 31775)(TX 0)(TC 382))
            (product\[4\] (T1 19200)(T0 32125)(TX 0)(TC 516))
            (product\[3\] (T1 12800)(T0 38525)(TX 0)(TC 396))
            (product\[2\] (T1 13000)(T0 38325)(TX 0)(TC 520))
            (product\[1\] (T1 0)(T0 51325)(TX 0)(TC 0))
            (product\[0\] (T1 26100)(T0 25225)(TX 0)(TC 519))
         )
 )
```

Figure 2.7: Sample switching activity file.

several cases in which the load can be much larger [3]:

1. Capacitive load due to long wires.

2. Driving an off-chip component.

3. Driving a global signal (clock, reset, etc.).

The simple answer to this problem is to increase the transistor size of the driver, which will increase the current available for the load. However, this does not solve the problem completely since it just pushes the problem one logic level back. Increasing the transistor size of the driver also increased its gate capacitance. Now it has become a large load for the stage which drives it. It is obvious that eventually we have to use stages with large transistor sizes, but we can minimize delay by using a chain of drivers. Figure 2.8 shows a chain of buffers to drive a large capacitive load. Calculations to find an optimum number of buffers to drive a large capacitive load are given below [25], [23]:

$$t_{total} = 2.2 * R_{min} * \alpha * C_{min} + 2.2 * \frac{R_{min}}{\alpha} * \alpha^2 * C_{min} + ... + 2.2 * \frac{R_{min}}{\alpha^{n-1}} * C_L \quad (2.18)$$

$$t_{total} = 2.2 * R_{min} * C_{min}(\alpha + \alpha + ... + \frac{1}{\alpha^{n-1}} * \frac{C_L}{C_{min}}) \quad (2.19)$$

At each stage, the ratio of the output capacitance to the input capacitance is α. It should be also the same for last stage:

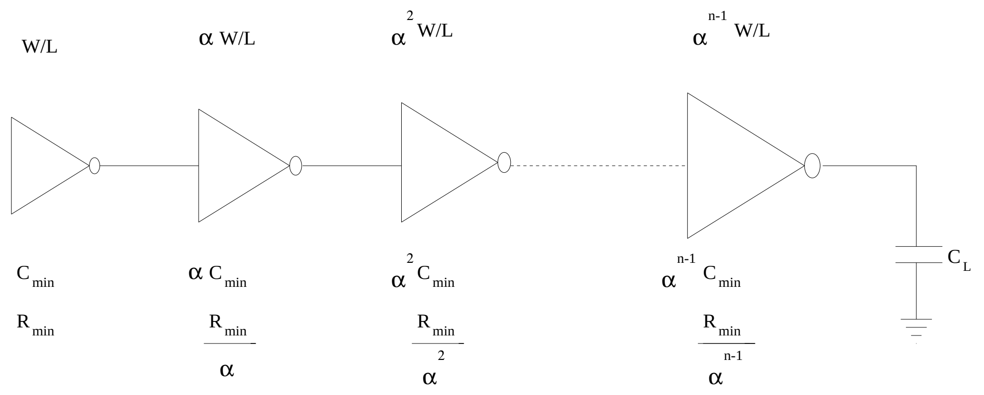


Figure 2.8: Inserting buffer chain.

$$\alpha = \frac{1}{\alpha^{n-1}} * \frac{C_L}{C_{min}}) \tag{2.20}$$

$$\alpha^n = (\frac{C_L}{C_{min}}) \tag{2.21}$$

After inserting Equation 2.21 into equation Equation 2.19;

$$t_{total} = n * \alpha * t_{min} \tag{2.22}$$

$$t_{min} = 2.2 * R_{min} * C_{min} \tag{2.23}$$

$$t_{total} = n * (\frac{C_L}{C_{min}})^{\frac{1}{n}} * t_{min} \tag{2.24}$$

t_{min} is the delay for a minimum size driver (R_{min}) which drives a minimum size load (C_{min}). To find the optimum number of stages which minimizes t_{total}, we have to solve the following equation:

$$\frac{dt_{total}}{dn} = 0 \tag{2.25}$$

which gives:

$$n_{optimum} = ln(\frac{C_L}{C_{min}}) \tag{2.26}$$

Pipelining

The clock period in a design is determined by the delay of the largest conbinational block. The large combinational block in Figure 2.9 has a delay of T_d. This indicates that the clock period for this design has to be at least T_d (if we ignore set-up and hold times of the flip-flops for the sake of simplicity). If the designer wants to run at faster clock, he/she has to reduce the delay of that large combinational block if possible. If it is not possible, pipelining is a technique which offers a solution. Pipelining partitions blocks of combinational logic into stages of equal delays, with the stages separated by banks of pipeline registers. Figure 2.9 also shows the design after pipelining registers

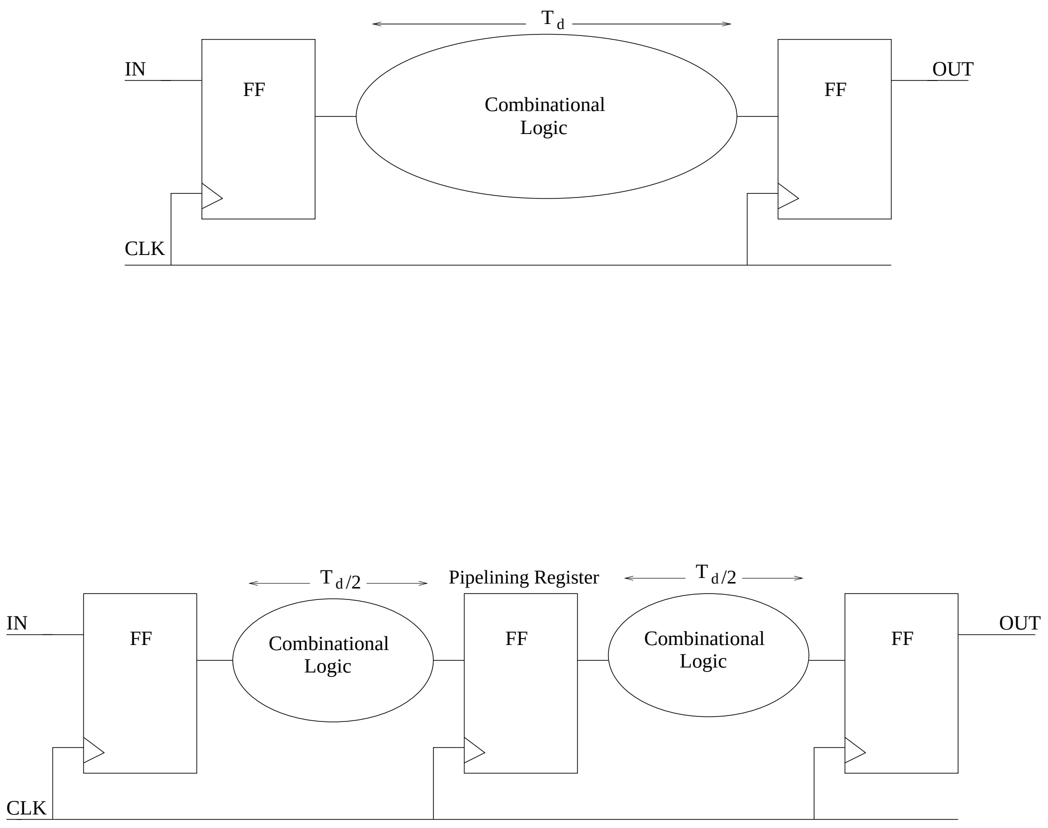


Figure 2.9: Before and after pipelining

have been inserted which has reduced the delay by half. If this reduction is not enough, new pipelining registers can be inserted to partition the combinational blocks further. This process can be carried on until a desired delay obtained.

Pipelining increases the throughput of the design but it introduces latency between the input data and the resulting output. Also pipelining increases the gate count because of the inserted pipelining registers. This technique is most useful for systems with a very high sampling rate.

2.4 ASIC Design Flow

An ASIC design flow is shown in Figure 2.10. The first step in the design flow is the development of a hardware description of the design. This can be done by using any of the hardware description languages (VHDL or Verilog) or by schematic entry. The second step is the functional verification of the HDL. This can be done by using RTL level simulators. A well defined testbench can be very helpful for verification purposes. The third step is the synthesis of the HDL. Synthesis is the process of generating logic-level representation from the HDL. Output of the synthesis step is a net-list of standard cells and interconnects. Standard cells are obtained from the vendors technology library. Technology libraries contains precharacterized standard cells for the target technology. The fourth step is the verification of the gate-level net-list. This step is also called gate-level simulation. The same testbench used for RTL verification can be used at this step. The fifth step is the placement and routing of the standard cells. Standard cells in the gate-level net-list are placed together to generate the layout.

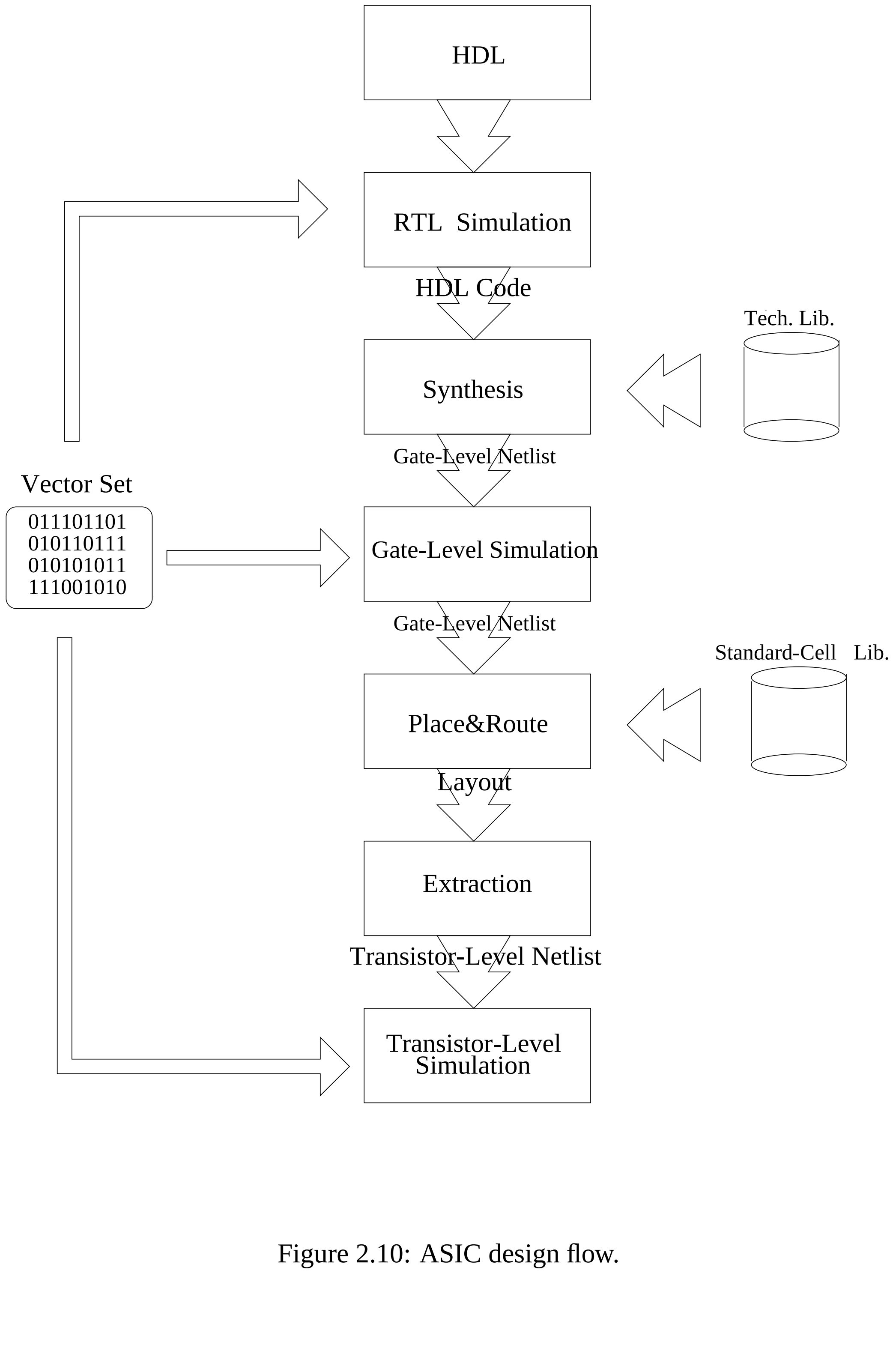


Figure 2.10: ASIC design flow.

Physical characteristics of the standard cells (height, width, I/O locations, power and ground rails, layers, vias, etc.) are already defined in the standard-cell libraries. Then, placed standard cells are routed. Routing can be timing driven, to give the first priority to the timing constraints, or power driven, to give the first priority to power constraints. If there is no violation, a transistor-level net-list is extracted from the layout for final verification of the functionality, timing and power. If the functionality and constraints are met, the design is placed in an I/O frame and sent for fabrication.

2.5 Related Work

In a paper closely related to the research, Bruni [10] developed a flow which starts by synthesizing the behavioral representation of the design with randomly selected constraints. The resulting RTL net-list is then evaluated for area, delay and power. The flow (Figure 2.11) utilizes a Monte-Carlo sampling of design space which is repeated automatically in an unsupervised mode to produce a statistical characterization of the design space. Extreme value theory is applied to extrapolate achievable bounds from the sampling points. Thus, this flow is intended as an off-line precursor to collect statistical information before actually starting a detailed search for an optimal solution. In contrast, the proposed design space search techniques are intended to identify the best solution possible given the user constraints.

In [15], RTL design space search is done through projected AT-curves (Area-Time curves). First the RTL module is synthesized for minimum area (A_a,T_a) and minimum delay (A_t,T_t), and these two delay values are treated as upper and lower bound tim-

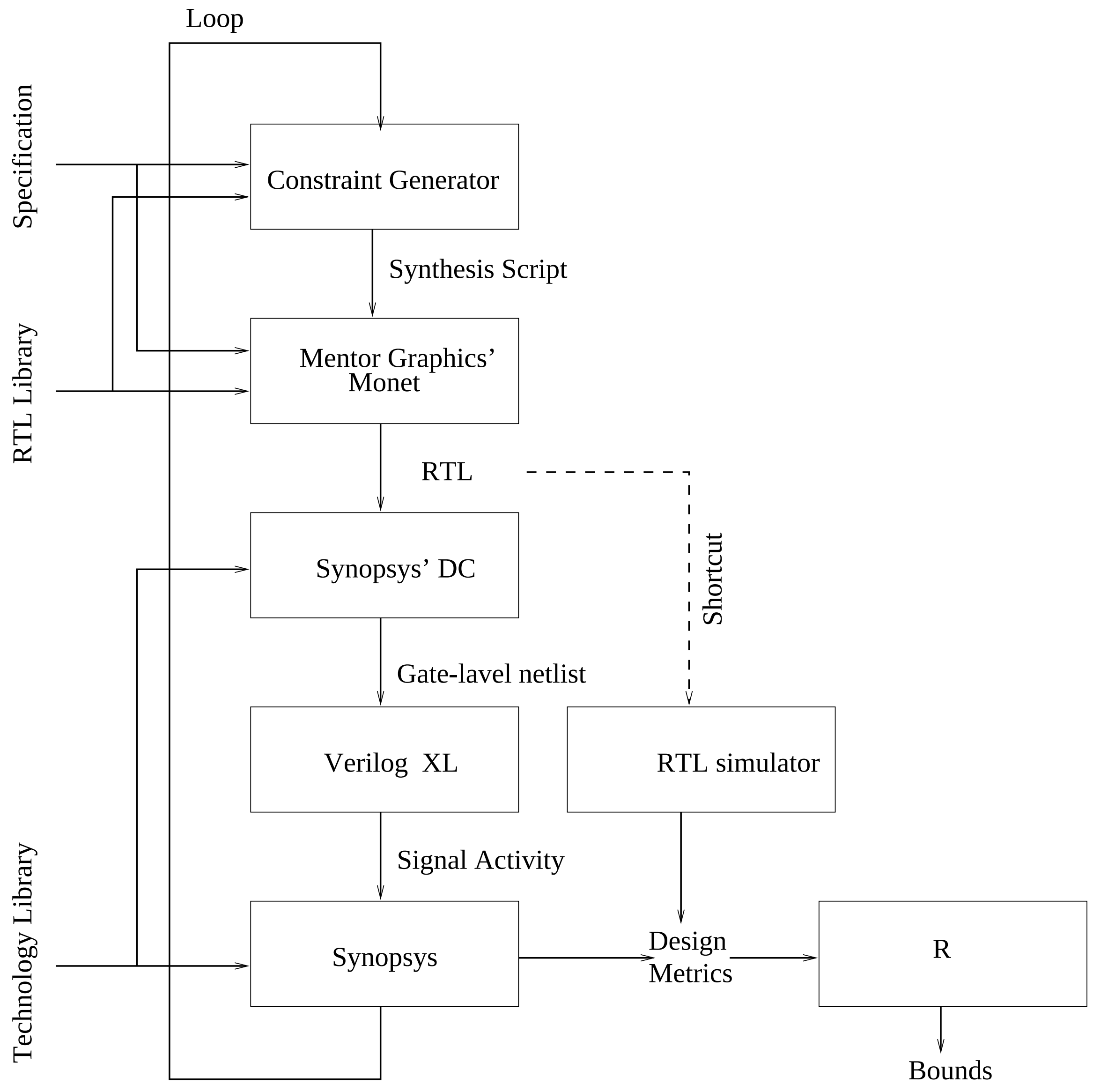


Figure 2.11: Monte-Carlo design space exploration flow

ing for the module. Then, the module is synthesized again for (T_a+T_t)/2 to generate third design point (A_{mid},T_{mid}). Then, (A_a,T_a),(A_t,T_t) points are used to generate a two-point AT projection. For three-point AT projection, points (A_a,T_a), (A_{mid},T_{mid}), (A_t,T_t) are used. Then, a linear function is used to derive the AT-curves. Optimum module selection is done using projected AT-curves. This approach has two main differences from our approach: (1) Optimum point selection is done from a projected AT-curve, in our case optimum design is selected among real data points generated through synthesis. (2) design space is considered 2 dimensional (Area-Time), in our case design space is considered 3 dimensional (Area-Time-Power).

In [8], a tool called GALOPS [7], a transformational based tool that uses a Genetic Algorithm (GA), has been used for design space search. GALOPS utilizes a GA to apply high level transformations to DSP algorithms at the behavioral level. Each high level transformation changes the area, speed, and power characteristics of the design. GA is used to generate a set of designs with the lowest power value for every generated area value. After each generation is created, Pareto-optimal points are selected from the generated set of designs. These steps are repeated and selected Pareto points at each generation are combined to create a global set of Pareto-optimal points. By using those Pareto-optimal points, Pareto-optimal surfaces are generated to illustrate the trade-offs between the conflicting parameters. Our approach combines three conflicting parameters (area, delay, power) in to a single cost function, and also our search technique is at the logic synthesis level.

In [13], authors presented a module selection procedure which uses both a complex operators library and the voltage scaling in order to optimize the design. The procedures goal is to determine the optimal supply voltage and the optimal operators set from a given module library according to a DSP application for design optimization. The module selection procedure uses two entries; the application, and the complex module library. The module library contains several parameters for each operator (module): area, latency time, pipeline stages number, functionality (adder, multiplier, etc.) and the effective capacitance. To explore the area-power space the module selection algorithm explores all the different solutions dealing with different operators with different area-power-time characteristic and different supply voltages. At each supply voltage value, the complex module library is searched through to identify the modules which minimize the given cost function (Cost$_{\alpha}(s) = (1-\alpha)\eta$Area(s)+$\alpha$Power(S)) For the selected set of operators and the supply voltage the cost function is re-evaluated using area and power estimates for the selected operators at the selected supply voltage. This method requires a pre-characterized module library. Whereas our method uses a standard-cell library which is available through a vendor. Also the cost function used in this method only accounts for area and power whereas in our approach the cost function accounts for area, power, and delay.

In [24], authors proposed a method to obtain area and delay estimates from RTL description. They observed that technology-dependent area and delay optimization consumes 85% of the total design time. Therefore, they proposed a method to estimate area and delay on technology- independent design. The estimates are obtained through fast compiler-type optimizations on the RTL description followed by application of best-fit polynomial area and delay models (models are previously generated from a technology library) on the resulting technology-independent code. Surely, this will help the designer to do a design space search at RTL code before mapping it into a technology. However, this approach does not provide any estimate about power.

CHAPTER 3

Design Space Search Algorithm: DSSA

A Design Space Search Algorithm (DSSA) has been developed to explore the design space for the minimum power-delay-area product (PDAP). Since the design-space is discrete, DSSA is implemented as heuristic. This algorithm is integrated into an ASIC design flow to guide the synthesis tool. DSSA has been tested on several macros and results are presented in section 3.3.

3.1 Implementation of DSSA

Pseudo code for the Design Space Search Algorithm (DSSA) is given in Figure 3.1 and Figure 3.2. DSSA is completely automated so that no human interaction is required once all the necessary initial parameters and files are provided. The initial parameters and files required are:

- RTL level description (using either VHDL or Verilog).
- Testbench for RTL-level, gate-level simulations and for the generation of the switching activity file.
- Initial synthesis script for the default run (Figure 3.3) which does not contain any constraints. This run is to provide the initial values of power, delay and area

```
$eps := min.improvement;$
$s := iteration\_step\_size;$
$n := number\_of\_iterations;$
$r := run\_time;$
$i := improvement\_ratio;$
$f := figure\_of\_merit;$
$Label0 : Loop\{$
$if(noi == 0)\{$
$run(defaultflow.scr);\}$
$else\{$
$run(optimizationflow.scr);\}$
$P := extract\_P(power.rpt);$
$D := extract\_D(timing.rpt);$
$A := extract\_A(area.rpt);$
$Path := extract\_max\_path(timing.rpt);$
$VP := extract\_violater\_path(violater.rpt);$
$FOM(noi) := calculate\_fom(f, P, D, A);\}$
$if(noi == 0)\{$
$save\_design\_parameters(0, P, D, A, FOM(0));\}$
$if(VP == 0)\{$
```

Figure 3.1: DSSA in pseudo code - Part I

```
$impr := calculate\_impr(FOM(0), FOM(noi));$
$save\_design\_parameters(noi, P, D, A, FOM(noi), impr); \}$
$const(noi) = generate\_new\_constrain(Path, D, P, A, s);$
$update\_synthesis\_script(const(noi));$
$noi++;$
$if(eps > (impr(noi) - impr(noi-1)))\{terminate;\}$
$if(impr = i)\{case := 1; gotoLabel1;\}$
$if(cput = r)\{case := 2; gotoLabel1;\}$
$if(noi = n)\{case := 3; gotoLabel1;\}\}$
$else\{$
$vconst(noi) = generate\_constrain\_for\_violater(VP, const(noi));$
$update\_synthesis\_script(vconst(noi));\}$
$\}while(noi < n);$
$Label1 : print"Continue?";$
$answer :='yes'/'no';$
$if(answer :='yes')\{$
$if(case == 1)\{i := new\_i;\}$
$if(case == 2)\{r := cput + extra\_r;\}$
$if(case == 3)\{n := noi + extra\_n;\}$
$gotoLabel0;\}$
$else\{terminate;\}$
```

Figure 3.2: DSSA in pseudo code - Part II

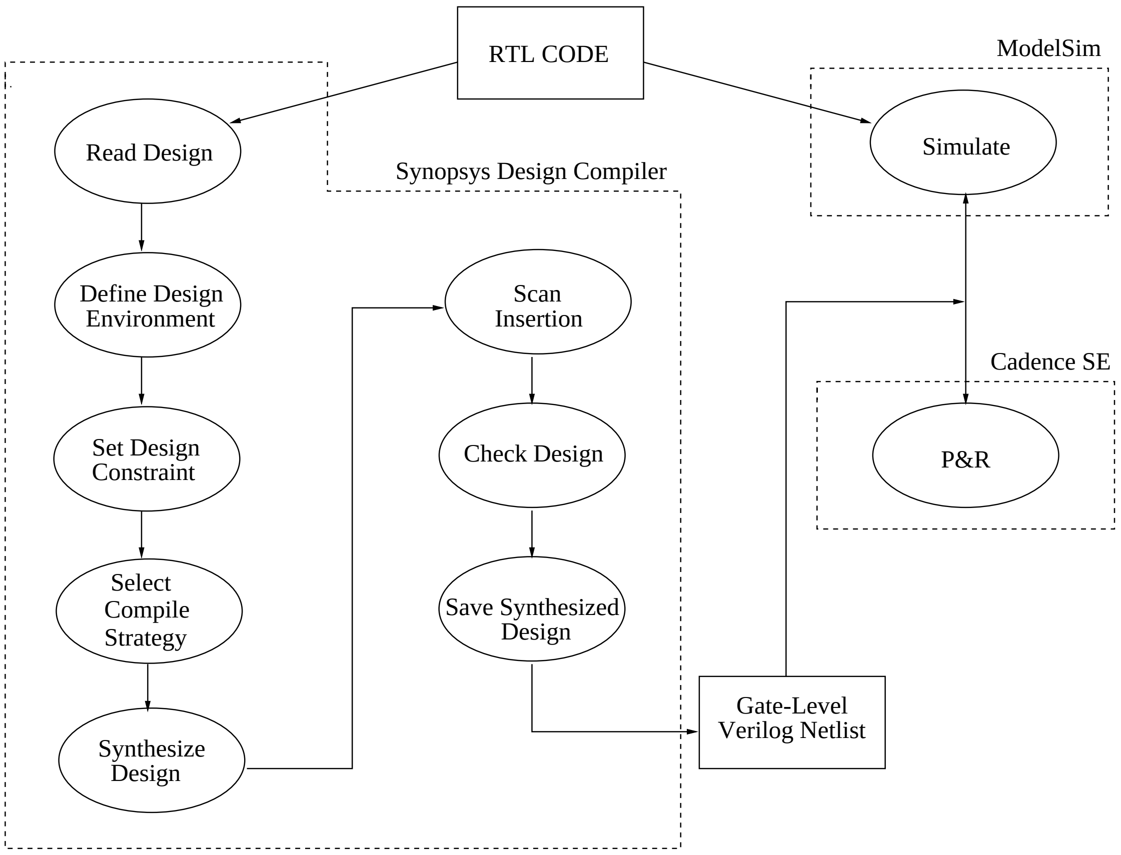


Figure 3.3: Default ASIC design flow

for the DSSA.

- Optimization synthesis script for the optimization runs (Figure 3.4).
- Control parameters that are set by the user: the desired run time, number of iterations and improvement ratio.

DSSA starts with the default run (which doesn't have any kind of optimization) for the given RTL. Then the initial values for delay, power, area and the path where the maximum delay occurs are extracted from the report files generated by the synthesis tool. The maximum delay path is constrained by the value of the maximum delay minus the iteration step size (choosing an iteration step size very small will increase the total run time but it will give a more complete picture of the design space.) and the synthesis script is updated with this new constraint. The DSSA also checks the other paths for any timing violations (which are then added as constraints).

At each iteration step, the CPU time, the number of iterations and the improvement ratio of the figure of merit (e.g. PDAP, DPP, power, delay, area, etc.) are checked. If the value of any of these parameters satisfies the user-defined control parameters (run time, number of iterations, improvement ratio), the iteration stops and waits for a response from the designer. Detailed sorted results are presented on the screen and in a file so the designer can decide whether to continue or stop. If the designer decides to continue, the DSSA will resume from the point where it stopped.

During the run, the improvement ratio of the figure of merit for the previous iteration and current iteration are checked. If the improvement is less than epsilon (a

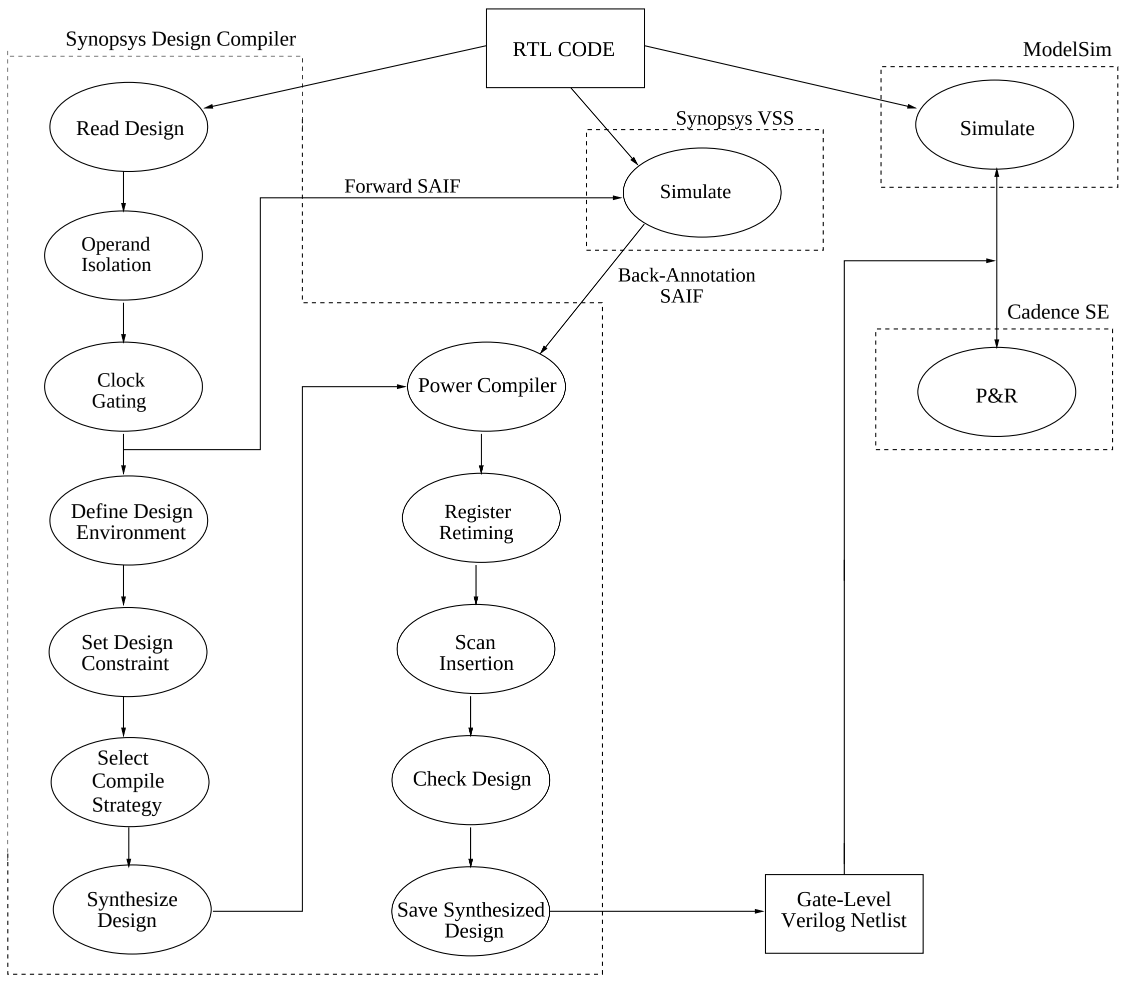


Figure 3.4: ASIC design flow with optimization

user-specified parameter), the iteration will stop even if the other user-defined specifications are not met. No change or little change in the improvement ratio means the iteration process has likely already reached the boundaries of the design space.

3.2 Integrating DSSA into an ASIC Flow

Figure 3.5 shows how DSSA is integrated into an ASIC flow. DSSA integrated ASIC flow contains 7 commercial tools coordinated by set of Perl and Tcl/Tk scripts. DSSA itself is written in Perl. RTL-level and gate-level simulations are performed by ModelSim. ModelSim is also used to generate RTL-level and gate-level switching activity files. Synopsys Design Compiler is used for synthesis. Synopsys PowerCompiler is invoked within the Design Compiler for power optimization and power estimation. DSSA performs design-space search (characterization) until one of the user defined parameters is satisfied. The search process is shown as a loop in Figure 3.5. Once the search is concluded, the design point with minimum FOM is sent for placement and routing. Placement and routing is performed by Silicon Ensemble. After placement and routing is completed a parasitic information file called DSPF (Detailed Specific Parasitic Format) is generated and backannotated to Design Compiler for more accurate power and delay values. Layout generation is performed by Cadence Design Frame Work. After a successful LVS (Layout versus Schematic) check a transistor-level net-list with parasitic RC values is extracted from the Layout. A transistor-level simulation is performed to verify functionality and to obtain more accurate power and delay values. Transistor-level simulation is performed by NanoSim and PathMill.

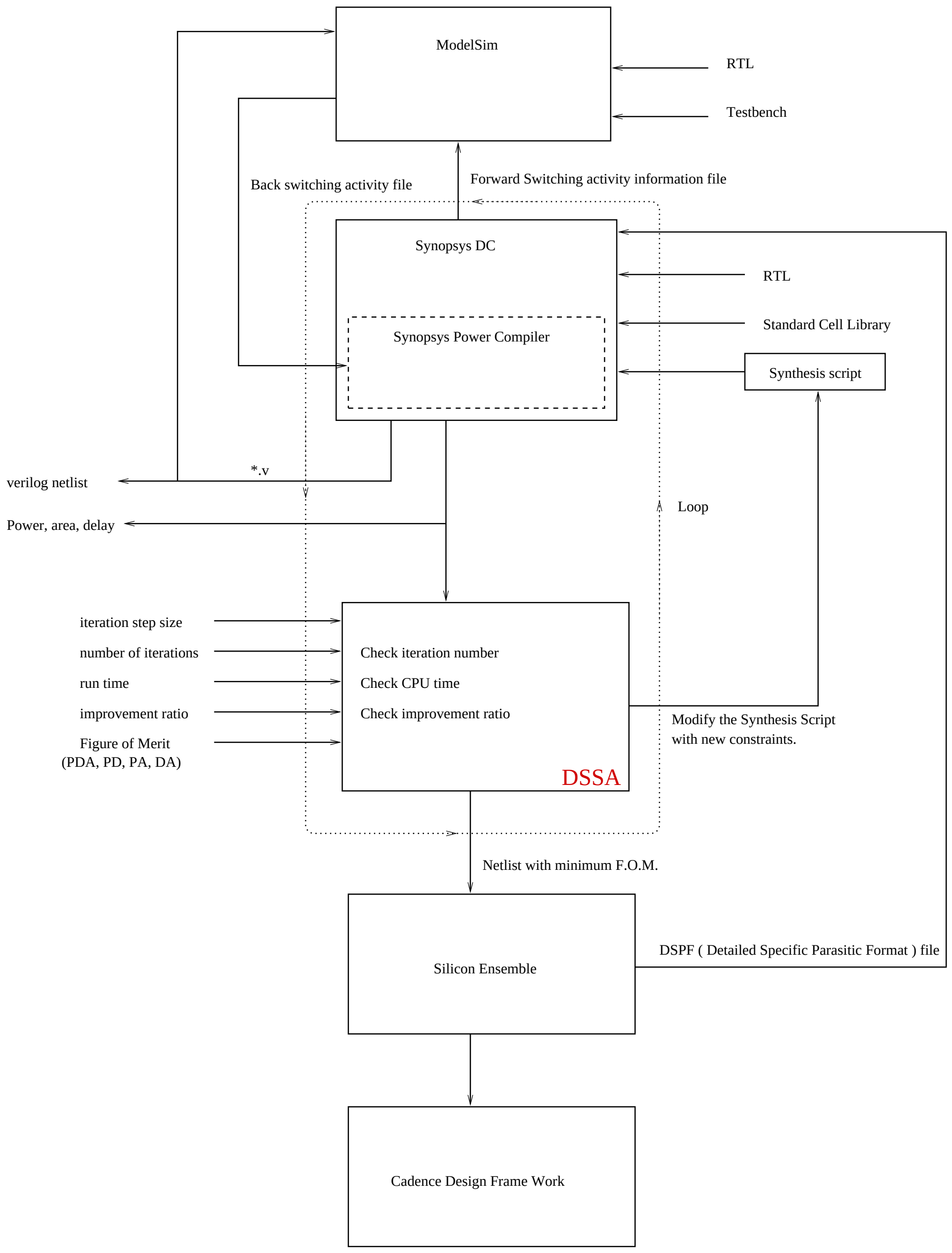


Figure 3.5: DSSA integrated into an ASIC flow.

Table 3.1: Characteristics of the macros

Design	# of I/O's	# of standard cells	# of Transistors
16-bit adder	49	19	492
16-bit multiplier	64	722	8552
16-bit complex-multiplier	130	2991	36218
12-bit 32-tap FIR	41	6445	92008
12-bit 4-pole 32-tap PFIR	111	9170	130036
12-bit 64-point FFT	101	8751	155084

3.3 Experimental Results for DSSA

We have tested the DSSA integrated flow on three combinational macros and three sequential macros. Characteristics of the macros are listed in Table 3.1. Table 3.2 shows the default delay and PDAP for the macros. Default values are obtained by using the default flow shown in Figure 3.3. Table 3.3 contains the optimum delay and PDAP. These are the optimum values for the optimization flow (Figure 3.4). Table 3.3 also tabulates the improvement ratio on the FOM.

Figure 3.6, Figure 3.7, Figure 3.8 Figure 3.9, Figure 3.10, and Figure 3.11 are the plots of data points generated during the design space search. Table 3.4 shows the number of points generated during the design-space search and how long it took to complete the search.

Table 3.2: Default values of delay and PDAP for the macros

Design	$d_{default}$(nsec)	$PDAP_{default}$(nJxμ^2)
16-bit adder	3.35	225.5
16-bit multiplier	4.43	129123
16-bit complex-multiplier	8.67	4543481
12-bit 32-tap FIR	7.06	43659291.6
12-bit 4-pole 32-tap PFIR	6.14	65476076.8
12-bit 64-point FFT	23.29	60758193.4

Table 3.3: Optimum values of delay and PDAP for the macros

Design	d_{opt}(nsec)	$PDAP_{opt}$(nJxμ^2)	$PDAP_{default}/PDAP_{opt}$
adder	3.43	183.2	1.23
multiplier	3.69	89120	1.45
complex-multiplier	5.07	2383333	1.91
FIR	5.16	35350768.4	1.24
PFIR	3.95	41202846.1	1.59
FFT	16.46	44239530.7	1.37

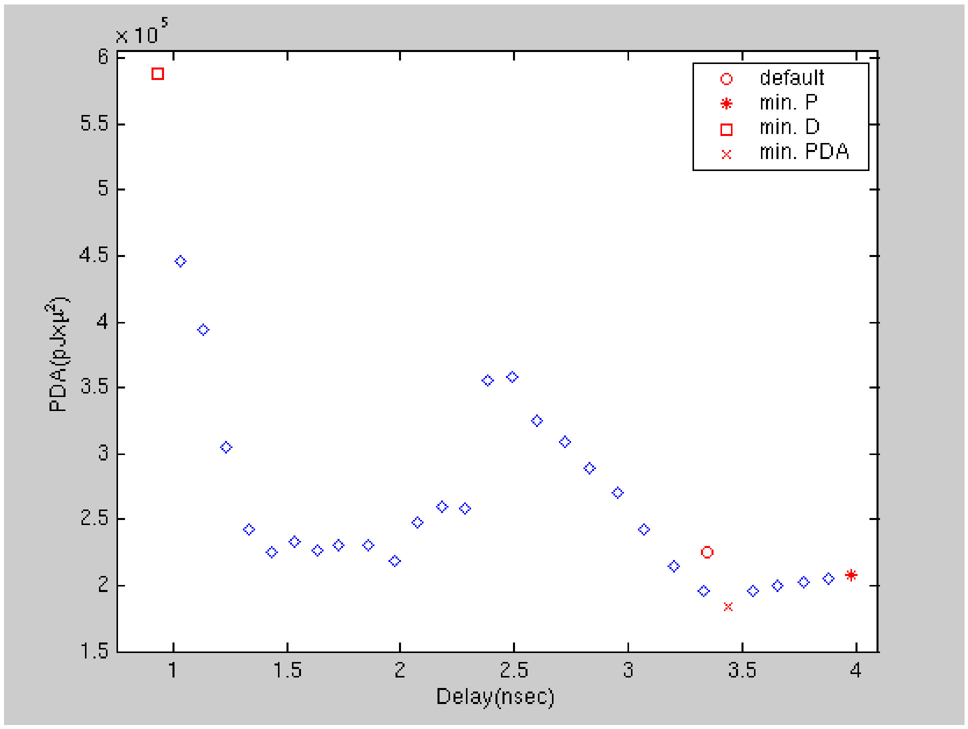


Figure 3.6: Searched space for 16-bit adder

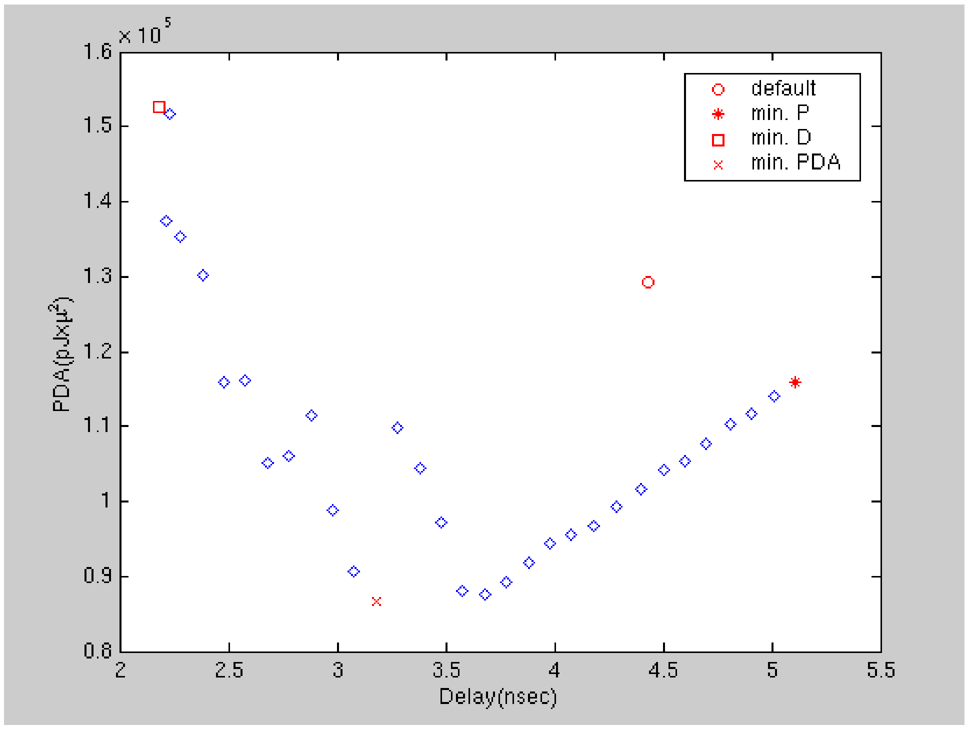


Figure 3.7: Searched space for 16-bit multiplier

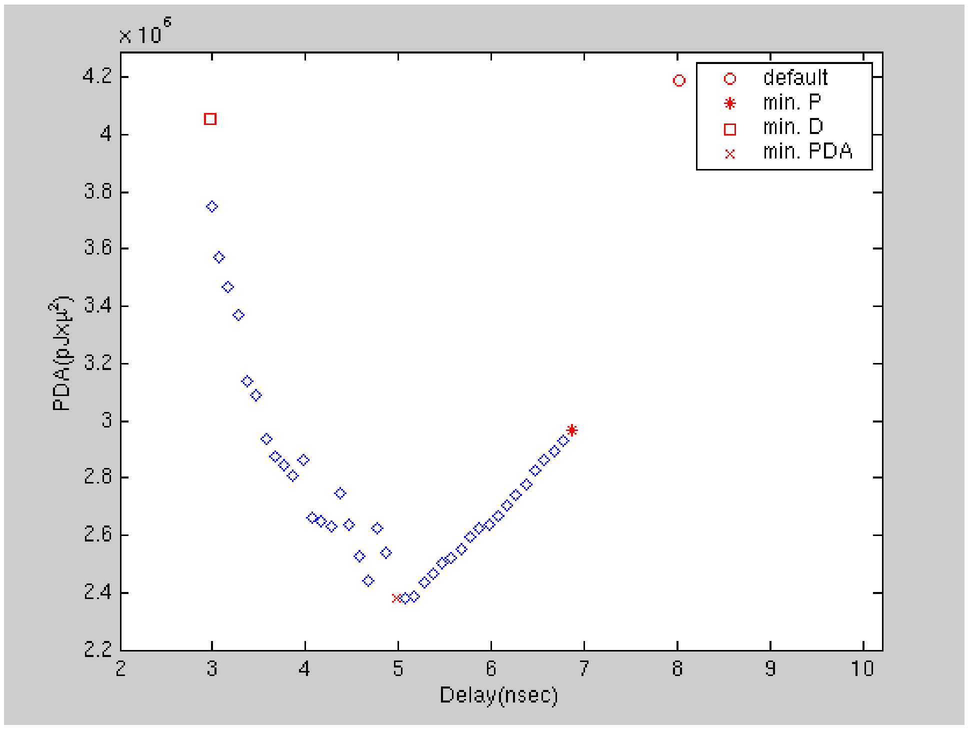


Figure 3.8: Searched space for 16-bit complex-multiplier

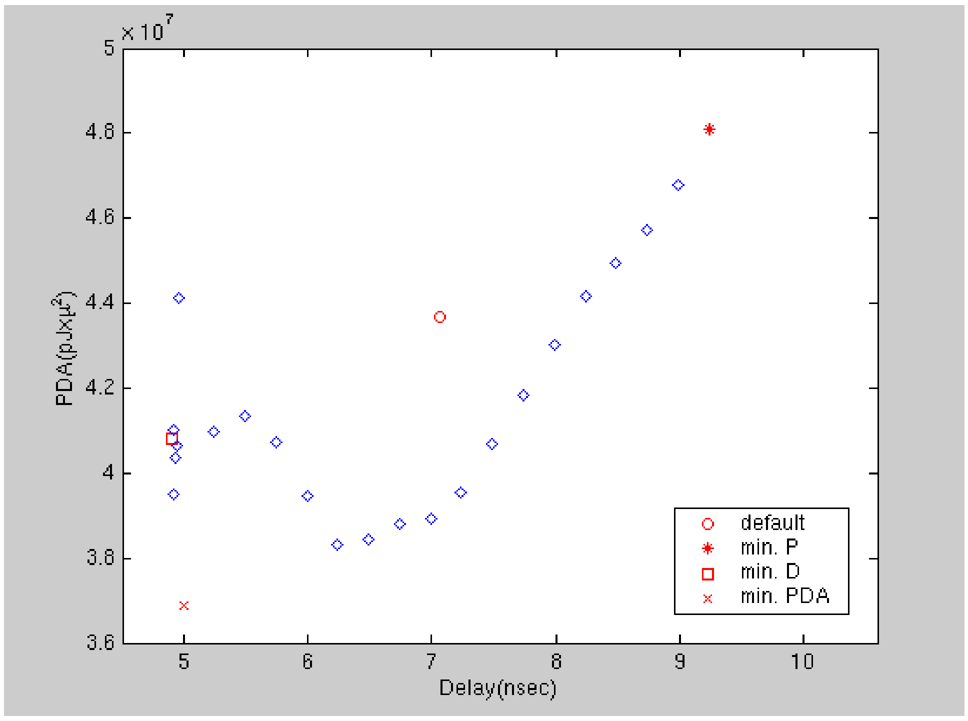


Figure 3.9: Searched space for 12-bit 32 tab FIR

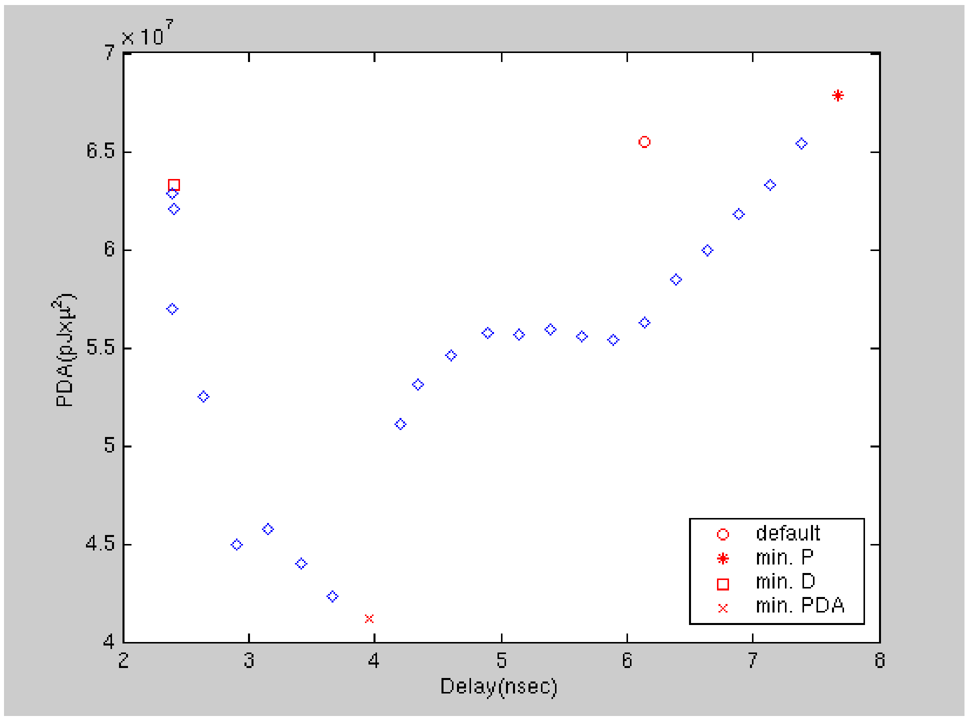


Figure 3.10: Searched space for 12-bit 4 pole 32 tab poly FIR

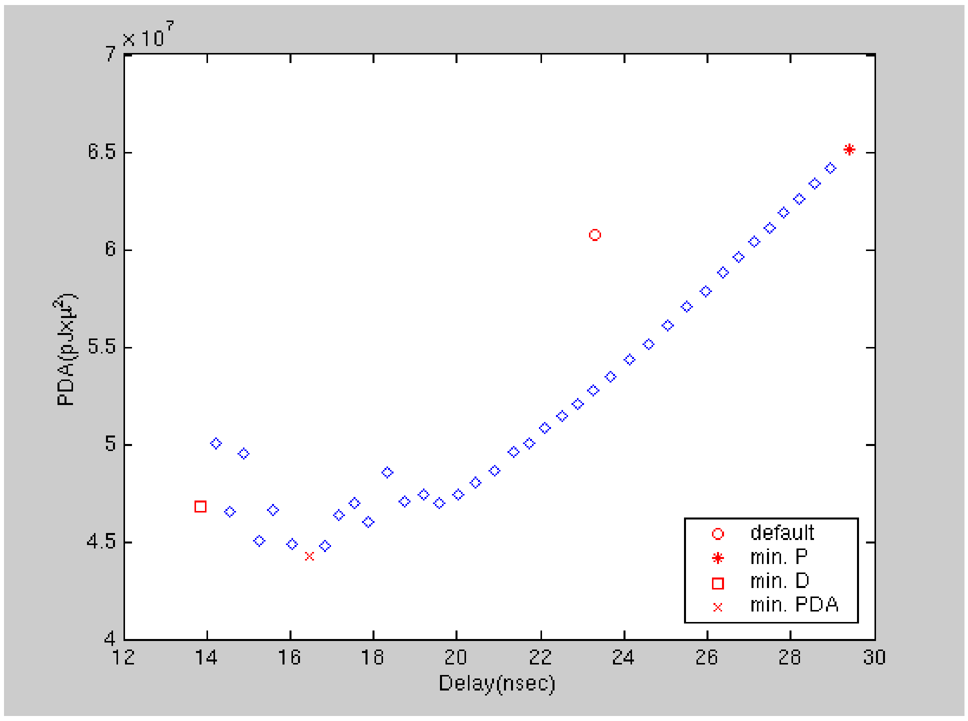


Figure 3.11: Searched space for 12-bit 64-point FFT

Table 3.4: Number of generated points and run times

Design	Number of points	Run time
adder	30	0.8 hrs.
multiplier	32	1.67 hrs.
complex-multiplier	41	8.5 hrs.
FIR	24	41.8 hrs.
PFIR	25	25.6 hrs.
FFT	40	72.35 hrs.

3.4 Parallel Design Space Search Algorithm: PDSSA

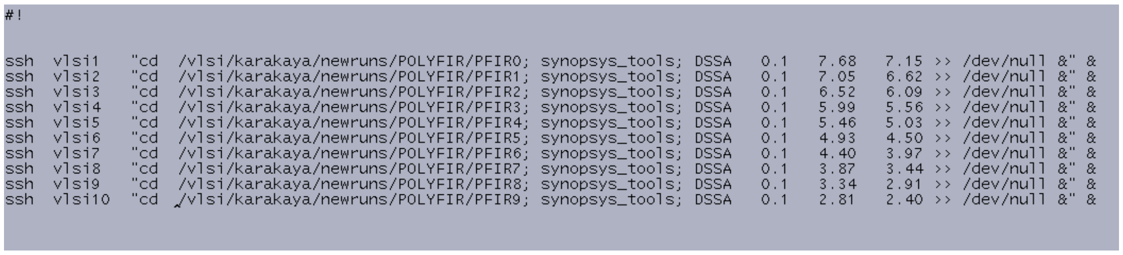
As shown in Table 3.4 design-space search times are very long. One way to reduce the search time is using parallel search algorithms. We have implemented parallel DSSA, called PDSSA, for that purpose. This section exs implementation of PDSSA, and presents the experimental results.

3.4.1 Implementation of PDSSA

Pseudo code for PDSSA is given in Figure 3.12. PDSSA starts with two runs to determine the lower and upper bound of the design-space. The upper limit for the design-space is the point in which power constrain set to minimum (at this point delay is not a constraint). The lower bound of the design-space is the point in which timing

```
$s := iteration\_step\_size;$
$N := number\_of\_parallel\_processes;$
$f := figure\_of\_merit;$
$m := macro\_name;$
$run(min\_power.scr);$
$upb := extract(timing.rpt); \%upperboundfordelay$
$run(min\_delay.scr);$
$lwb := extract(timing.rpt); \%lowerboundfordelay$
$clone(macro, N);$
$generate\_parallel\_run\_scrpt(macro, N, upb, lwb);$
$start\_parallel\_run;$
```

Figure 3.12: PDSSA in pseudo code



```
#!

ssh  vlsi1   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR0; synopsys_tools; DSSA   0.1   7.68   7.15 >> /dev/null &" &
ssh  vlsi2   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR1; synopsys_tools; DSSA   0.1   7.05   6.62 >> /dev/null &" &
ssh  vlsi3   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR2; synopsys_tools; DSSA   0.1   6.52   6.09 >> /dev/null &" &
ssh  vlsi4   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR3; synopsys_tools; DSSA   0.1   5.99   5.56 >> /dev/null &" &
ssh  vlsi5   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR4; synopsys_tools; DSSA   0.1   5.46   5.03 >> /dev/null &" &
ssh  vlsi6   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR5; synopsys_tools; DSSA   0.1   4.93   4.50 >> /dev/null &" &
ssh  vlsi7   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR6; synopsys_tools; DSSA   0.1   4.40   3.97 >> /dev/null &" &
ssh  vlsi8   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR7; synopsys_tools; DSSA   0.1   3.87   3.44 >> /dev/null &" &
ssh  vlsi9   "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR8; synopsys_tools; DSSA   0.1   3.34   2.91 >> /dev/null &" &
ssh  vlsi10  "cd  /vlsi/karakaya/newruns/POLYFIR/PFIR9; synopsys_tools; DSSA   0.1   2.81   2.40 >> /dev/null &" &
```

Figure 3.13: A sample shell script generated by PDSSA for parallel search

constraint and the power constraint are set to minimum. After determining the lower and upper bounds of the design-space, it is divided in to the number of the machines available for the parallel run. Then, PDSSA generates a shell script for the parallel run. A sample shell script, generated by PDSSA, is given in Figure 3.13. The sample script was generated to run 10 machines in parallel. As seen from Figure 3.13, on each machine a DSSA runs in its assigned range. For this purpose DSSA exed in section 1.1 is slightly modified to except lower and upper boundaries for the iteration. Since the design space is discrete, there is no need for the communication of the parallel running machines during the search.

3.4.2 Experimental Results for PDSSA

Design-space search for all macros are repeated using PDSSA and run times for DSSA and PDSSA are tabulated in Table 3.5 for comparison. As seen from the results PDSSA dramatically reduced the run times. For DSSA runs only one machine (Sun Enterprise

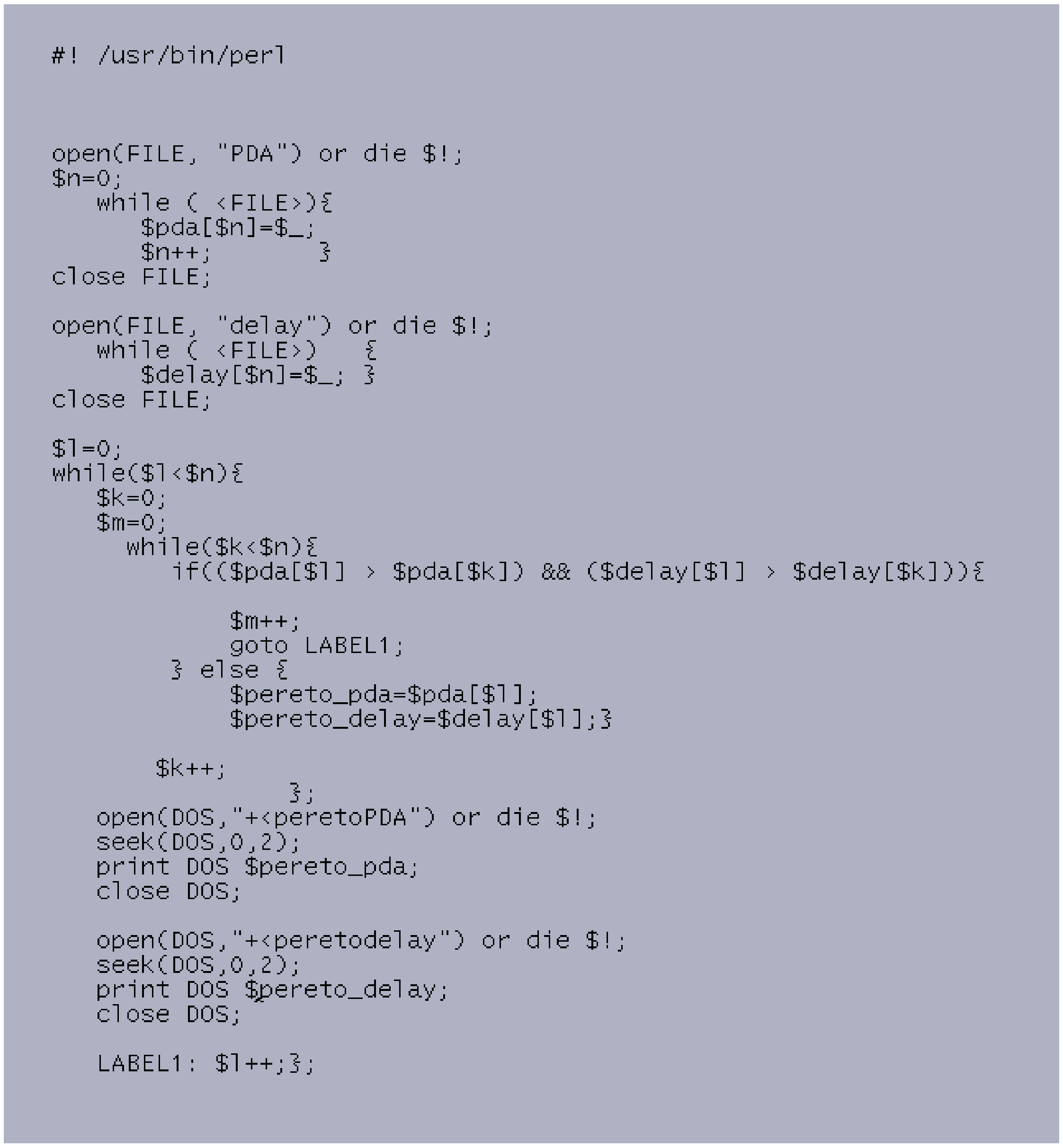
Table 3.5: Run times for DSSA and PDSSA

	Run Time	
Design	DSSA	PDSSA
adder	0.8 hrs.	4 min.
multiplier	1.67 hrs.	12 min.
complex-multiplier	8.5 hrs.	1.12 hrs.
FIR	41.8 hrs.	4.5 hrs.
PFIR	25.6 hrs.	3.75 hrs.
FFT	72.35 hrs.	7.4 hrs.

220R server, 450 MHz. dual CPU) is used. PDSSA runs used 10 similar machines in parallel.

3.5 Pareto Points

It is possible to characterize the design-space by the set of optimal tradeoff design points. For those points there is no design with a smaller PDAP and the same or smaller delay, and no design with a smaller delay and the same or smaller PDAP. Such points are called Pareto points. An algorithm (Figure 3.14) is attached to the DSSA and PDSSA to identify the Pareto points. This algorithm is written in Perl. It goes through the design-points generated during the design-space search and selects Pareto points



```
#! /usr/bin/perl


open(FILE, "PDA") or die $!;
$n=0;
   while ( <FILE>){
      $pda[$n]=$_;
      $n++;       }
close FILE;

open(FILE, "delay") or die $!;
   while ( <FILE>)   {
      $delay[$n]=$_; }
close FILE;

$l=0;
while($l<$n){
   $k=0;
   $m=0;
     while($k<$n){
        if(($pda[$l] > $pda[$k]) && ($delay[$l] > $delay[$k])){

            $m++;
            goto LABEL1;
        } else {
            $pereto_pda=$pda[$l];
            $pereto_delay=$delay[$l];}

       $k++;
                };
   open(DOS,"+<peretoPDA") or die $!;
   seek(DOS,0,2);
   print DOS $pereto_pda;
   close DOS;

   open(DOS,"+<peretodelay") or die $!;
   seek(DOS,0,2);
   print DOS $pereto_delay;
   close DOS;

   LABEL1: $l++;};
```

Figure 3.14: Perl code to determine the Pareto points.

to characterize the design-space. Figure 3.15, Figure 3.16, Figure 3.17, Figure 3.18, Figure 3.19, Figure 3.20 shows the Pareto curves for the DSP macros.

3.6 Chapter Conclusion

Table 3.6 shows that as we increase the step size, the "optimum" PDAP found by this technique gets worse. The reason is, because of the large step-size, what is probably the true optimum point is skipped. To prevent this, we have to use an algorithm which is not using a fixed step-size.

Table 3.6: Effect of step-size on PDAP (for multiplier)

Step-size(nsec)	d_{opt}(nsec)	$PDAP_{opt}$(nJxμ^2)
0.1	3.18	86656
0.2	3.69	89120
0.3	3.61	90577
0.4	3.49	91483

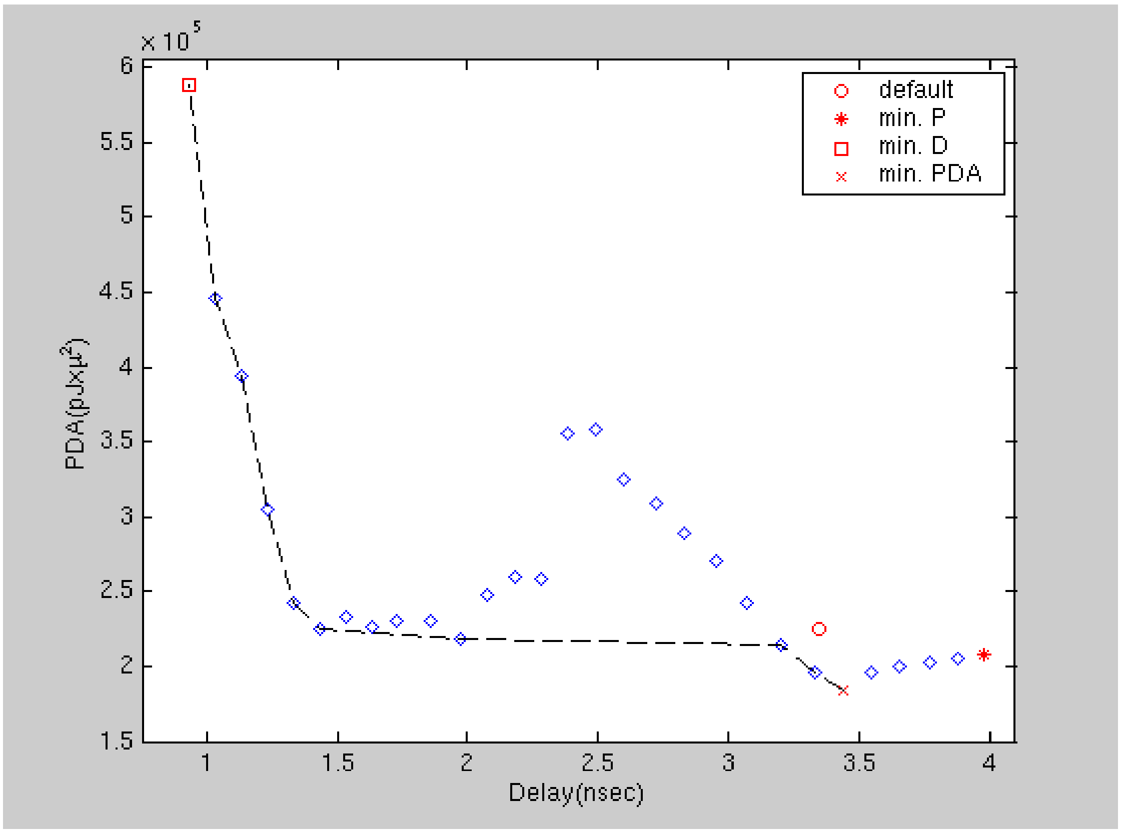


Figure 3.15: Pareto curve for 16-bit adder.

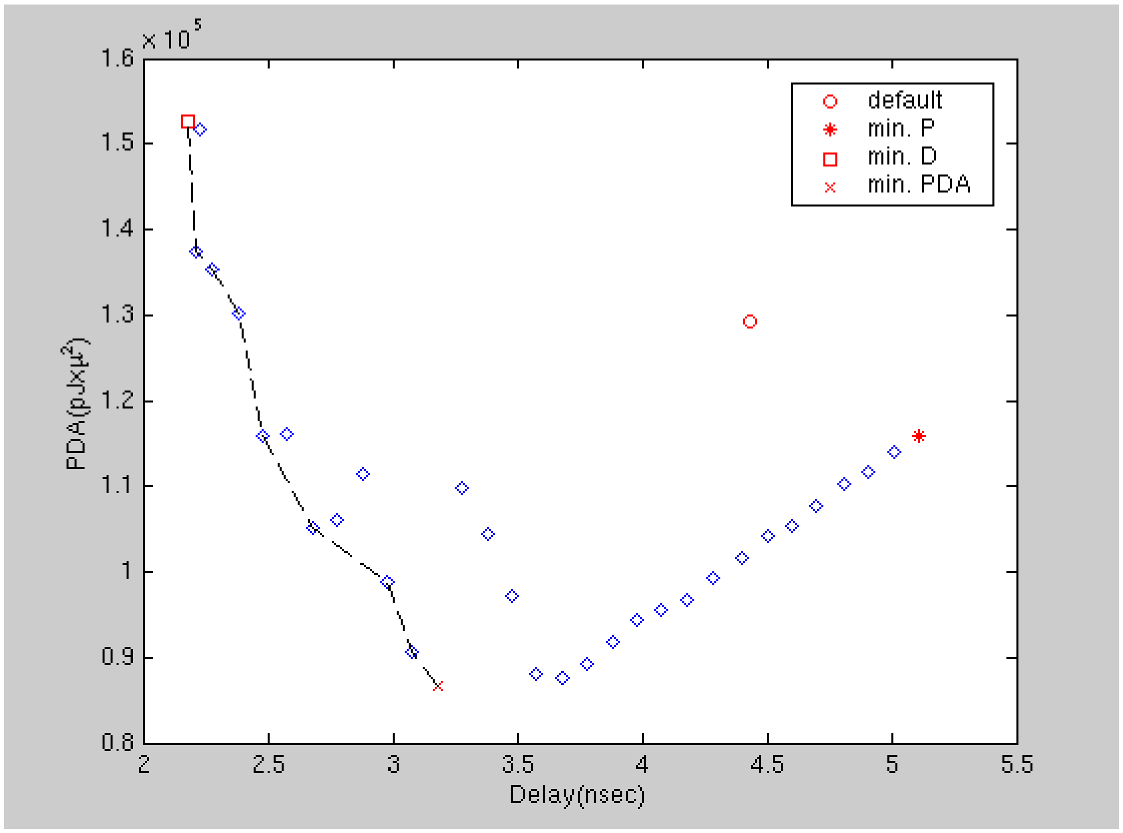


Figure 3.16: Pareto curve for 16-bit multiplier.

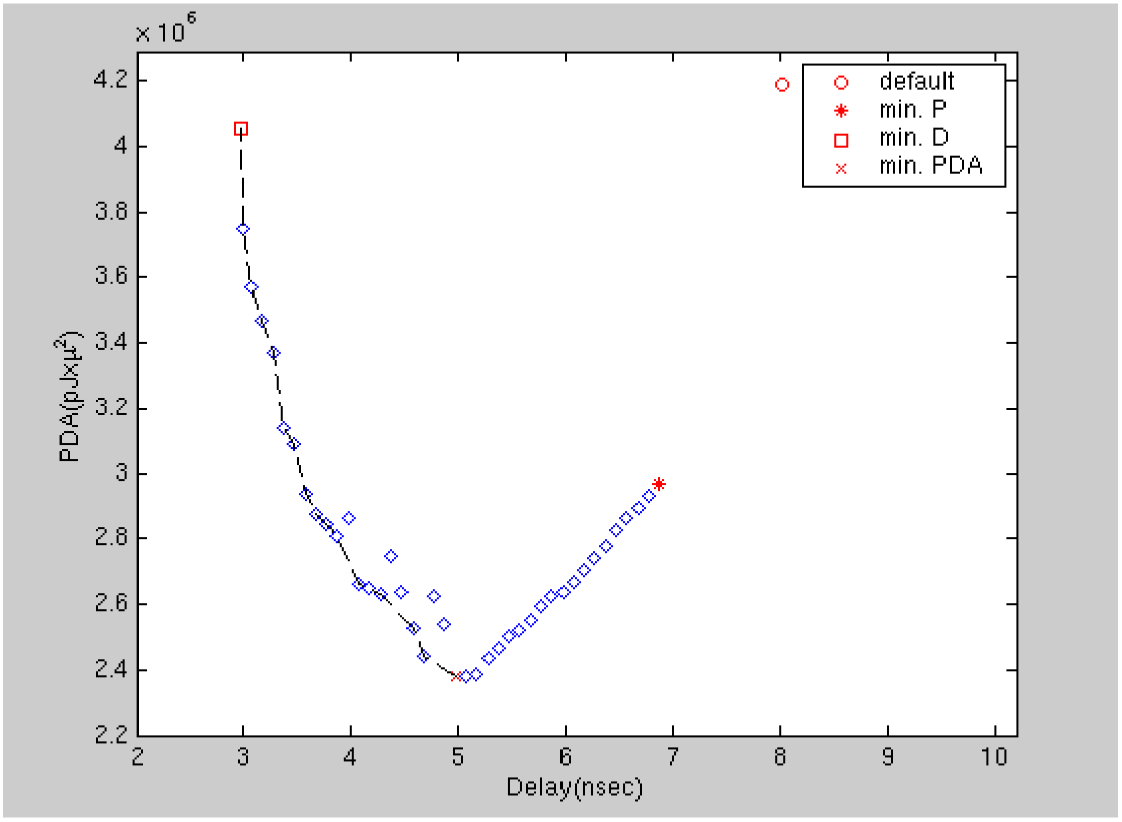


Figure 3.17: Pareto curve for 16-bit complex-multiplier.

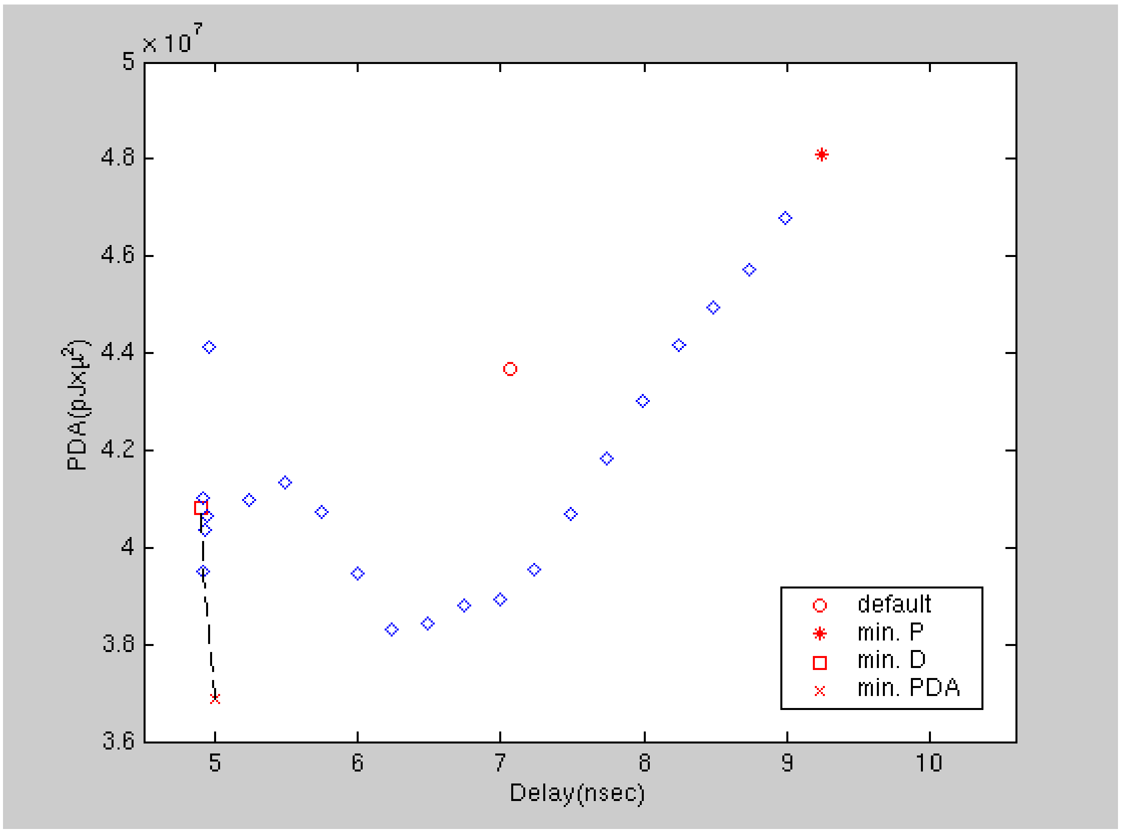


Figure 3.18: Pareto curve for FIR.

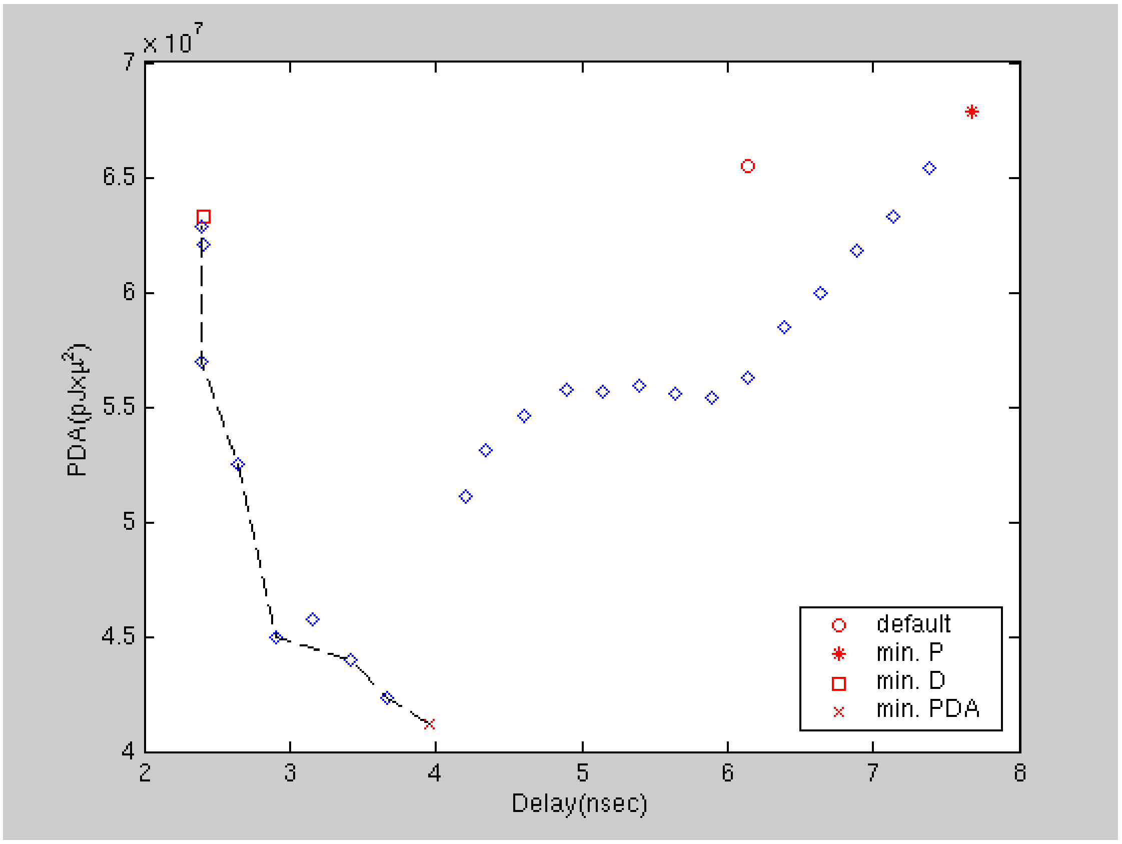


Figure 3.19: Pareto curve for poly-FIR.

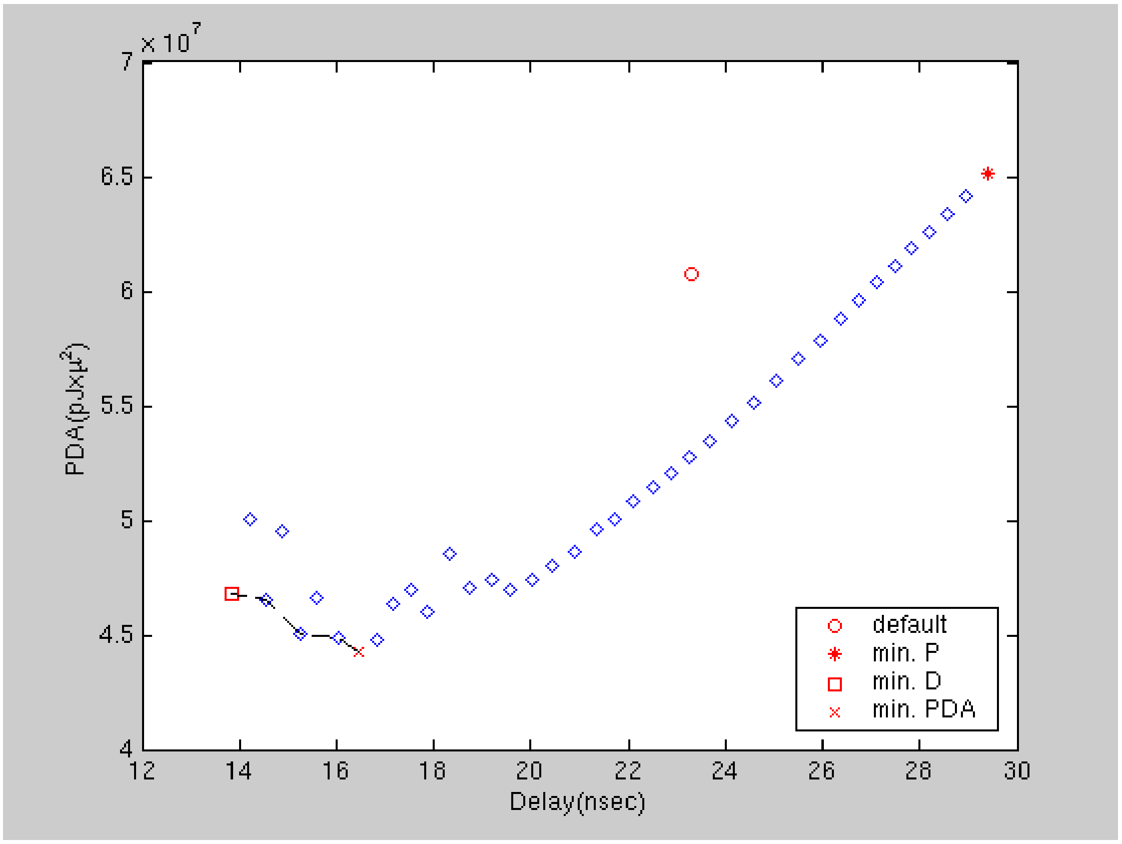


Figure 3.20: Pareto curve for 64 point FFT.

CHAPTER 4

Searching the Design Space Using Simulated Annealing

Due to the step size dependence of the DSSA and PDSSA, it is possible to skip the true optimum PDAP point during the design space search. As a result we decided to investigate an alternate search technique to determine whether it could produce higher quality results.

4.1 Simulated Annealing

Simulated Annealing (SA) is an iterative improvement algorithm (optimization technique) based on the principles of thermodynamics. SA is based on an analogy of the annealing process of solids. It was first introduced in a paper published by Metropolis et al. in 1953 [20]. In [16] it was first used on a large combinatorial problem to find an approximate solution. SA accepts better moves (which reduces the cost function) unconditionally, but unlike some other algorithms it also accepts inferior moves as the new solution with a probability to provide a possibility of escaping a local minimum. A general SA algorithm is given in Table 4.1.

- *Initialize()* is to set initial guess value for the parameter to be optimized.
- *HeatUp()* is to determine the starting temperature.

Table 4.1: A general SA algorithm in pseudo code

```
$S_{current} := Initialize()$
T:=*HeatUp()*
Loop
Loop
$S_{next} := perturb(S_{current})$
$\Delta Cost := Cost(S_{next}) - Cost(S_{current})$
$if \Delta Cost < 0$ or $Accept(\Delta Cost, T)$
$S_{current} = S_{next}$
until *Equilibrium()*
T:=*CoolDown()*
until *Frozen()*
```

- *Perturb()* is to disturb the system with the current input to determine the next input.

- *Accept()* is to accept the inferior moves (which increases the cost function) with the probability of $e^{(-\Delta Cost/T)}$.

- *Equilibrium()* is to check whether the required number of iterations has been made or not.

- *CoolDown()* is to reduce the temperature according to a given schedule.

- *Frozen()* is to evaluate the rate of change of the system response. If it is smaller than a given epsilon, terminate the algorithm.

4.2 Implementation

Simulated Annealing has been implemented in a number of applications in electronic design automation including routing [6], circuit partitioning [22], scheduling [12]. In our implementation we have used SA to search the design space for minimum Power-Delay-Area product. The SA code is written in C. The interface between the SA code and the synthesis tool is provided by a set of scripts written in Perl. Synopsys Design Compiler [3] is used for logic synthesis and Synopsys Power Compiler [4] is used for power optimization during the search. A switching activity file is generated for better power optimization using a random set of input vectors. At each run, Design Compiler writes out power, delay and area reports. Perl scripts are used to extract the power, delay and area values from those report files and to update the current value of the cost function. The SA algorithm makes its decision according to this new value of the cost function. The cost function in our SA based optimization is PDA product which is given in Equation 4.1. However, the cost function can easily be changed to be Power-Delay product, Power-Area product or Area-Delay product depending on the user's preference.

$$\left.\begin{array}{c} Cost = P * D * A \\ d_{min} \leq D \leq d_{max} \end{array}\right\} \tag{4.1}$$

As seen from Equation 4.1, it is a cost function with constraints. d_{min} is the minimum delay for the design. d_{max} is the delay where power consumption of the design is minimum. Before starting the main design space search, an automated procedure finds d_{min} and d_{max}. d_{min} is found simply by setting the timing constraint to zero, and d_{max} is found by setting the power constraint to zero during synthesis. Using a constraint cost function reduces the search time. We have used Cauchy training as given in Equation 4.2 and Equation 4.3 (T is the temperature and α is a constant).

$$\Delta D = \alpha * T * tan(\theta) \tag{4.2}$$

$$\theta = \frac{\pi}{2} * (2 * rand() - 1) \tag{4.3}$$

Equation 4.4 shows how the next delay point is generated. After each new delay point is generated, the timing constraint in the synthesis script is updated to reflect the new delay target. At each point power, delay, and area values are recorded and the technology-dependent net-list is written out.

$$D_{next} = D_{current} + \Delta D \tag{4.4}$$

Table 4.2: Characteristics of the macros

Design	# of Transistors	# of I/O's
16-bit adder	492	49
16-bit multiplier	8552	64
16-bit complex-multiplier	36218	130

The schedule used for cooling is given by Equation 4.5. Where T_0 is the starting temperature, β is a constant, and k is the current iteration step number.

$$T(k) = \frac{T_0}{1 + \beta * k} \tag{4.5}$$

4.3 Experimental Results

We have tested the SA implementation using three combinational macros. The characteristics of the macros are given in Table 4.2. The adder and multiplier were taken directly from Synopsys Design Ware library. The complex-multiplier was created using components from the Synopsys Design Ware library.

Table 4.3 shows the upper (d_{max}) and lower (d_{min}) limits of the search space for the combinational macros used. d_{opt} is the delay point where the minimum PDA product was found. PDA products at d_{min}, d_{max}, d_{def}, and d_{opt} are listed in Table 4.4. Table 4.5 tabulates improvement ratio on PDAP for the three search techniques. Figure 4.1, Figure 4.2 and Figure 4.3 are the plots of data points generated during the search.

Table 4.3: Constraints of the macros

Design	d_{max}(nsec)	d_{min}(nsec)	d_{opt}(nsec)
adder	4.05	0.77	3.30
multiplier	5.11	2.20	3.17
complex-multiplier	8.67	2.99	4.69

Table 4.4: PDA products

	PDA (nJxμ^2)			
Design	d_{max}	d_{min}	d_{def}	d_{opt}
adder	210	1161.3	225.5	179.5
multiplier	115925.5	163371.7	129123	87470.7
comp-mult.	2969216	4186043.6	4543481	2314768.7

Table 4.5: PDAP improvement ratios

	PDAP Improvement Ratio		
Design	DSSA	PDSSA	SA
adder	1.23	1.23	1.26
multiplier	1.45	1.45	1.48
complex-multiplier	1.91	1.91	1.96

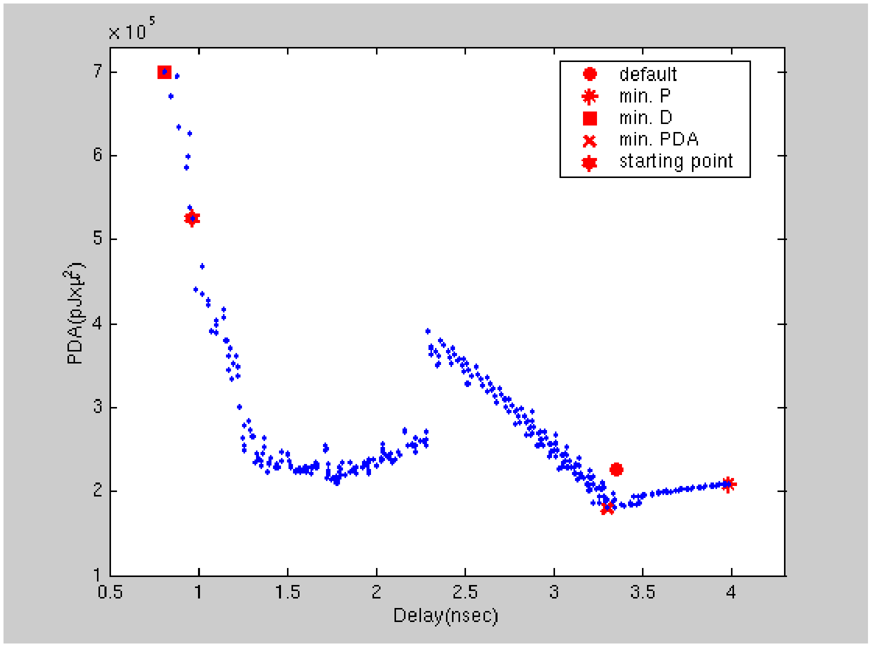


Figure 4.1: Searched space for adder

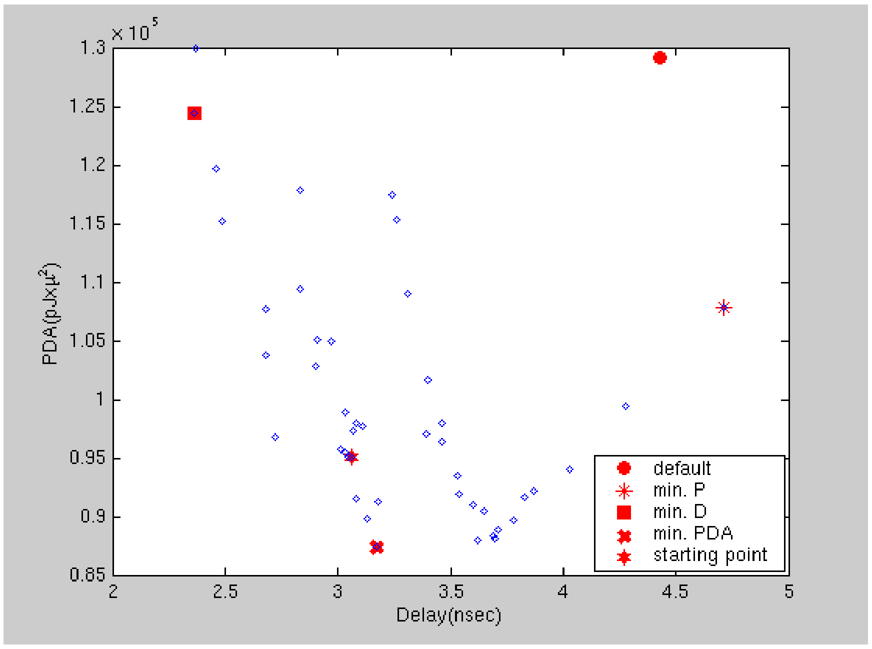


Figure 4.2: Searched space for multiplier

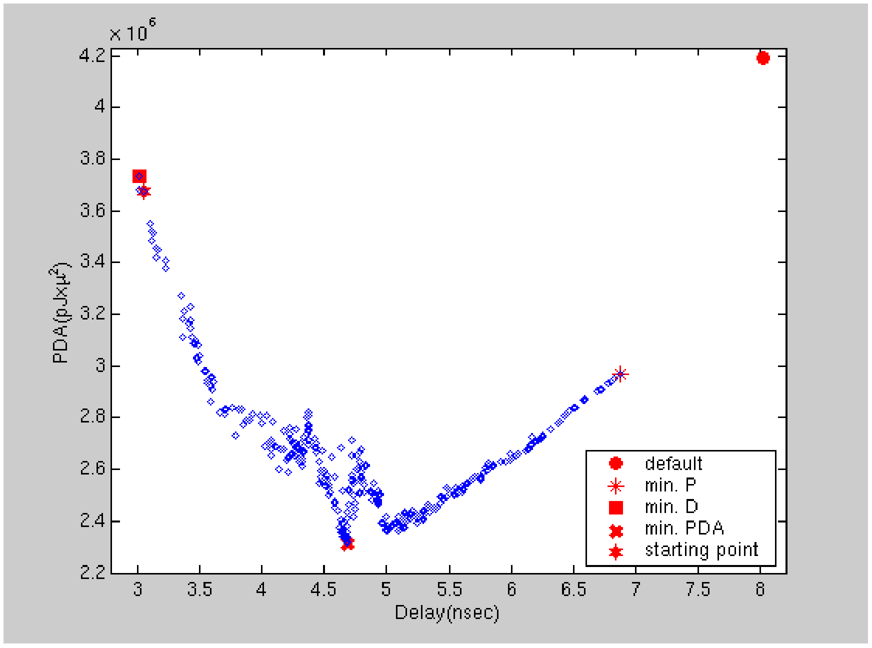


Figure 4.3: Searched space for complex-multiplier

4.4 Chapter Conclusion

Table 4.6 tabulates the number of points generated during each search technique and also the total run time of each technique. It should be noted that run times for the macros using the SA technique are from 2x-9x greater than those for the DSSA. Of course, more design points are generated for the SA technique. As tabulated in Table 4.5, the improvement ratio for the SA is only slightly better than those obtained by DSSA and PDSSA. For example, for the case of the complex-multiplier, the SA produced only a 3% better result than DSSA and PDSSA, but it took SA 9x more than DSSA and 67x more than PDSSA to find it. As a result we have decided not to run the design space search using the SA technique for FIR, PFIR, and FFT since the DSSA and PDSSA achieved comparable results in far less time.

Table 4.6: Run times

	Number of Points Generated			Run Time		
Design	DSSA	PDSSA	SA	DSSA	PDSSA	SA
adder	30	30	324	0.8 hrs.	4 min.	4.93 hrs.
multiplier	32	32	49	1.67 hrs.	12 min.	2.25 hrs.
comp.-mult.	41	41	389	8.5 hrs.	1.12 hrs.	75.62 hrs.

CHAPTER 5

Graphical User Interface (GUI) Design

Team_Liberator1.0 (Figure 5.1) is a Graphical User Interface (GUI) which is designed to combine the optimization techniques developed by three universities (University of Tennessee, University of Washington, and University of California at Santa Cruz) for the Phase I of DARPA project. The GUI is written in Tk. Data transfer between the tools and the GUI is provided by a set of Perl scripts.

5.1 MacroGen

MacroGen is a DSP macro generator tool. It can be initialized within the Team_Liberator1.0 (Figure 5.2) or it can be run as a stand-alone tool. MacroGen contains templates of the generic DSP macros. It generates macros from those templates according to the values for the generic parameters given by the user. MacroGen is initialized within the Team_Libeartor1.0 by simply clicking on the ”MACROLIST” button located in the main window (Figure 5.2) The set of available DSP macros is listed in the MacroGen window (for now the only macros available are adder, multiplier, complex-multiplier, FIR, poly-FIR, and FFT). The desired macro can be selected from the MacroGen window simply by clicking on the macro name. A new window will then pop-up automatically(Figure 5.3). Values for the generic parameters of the

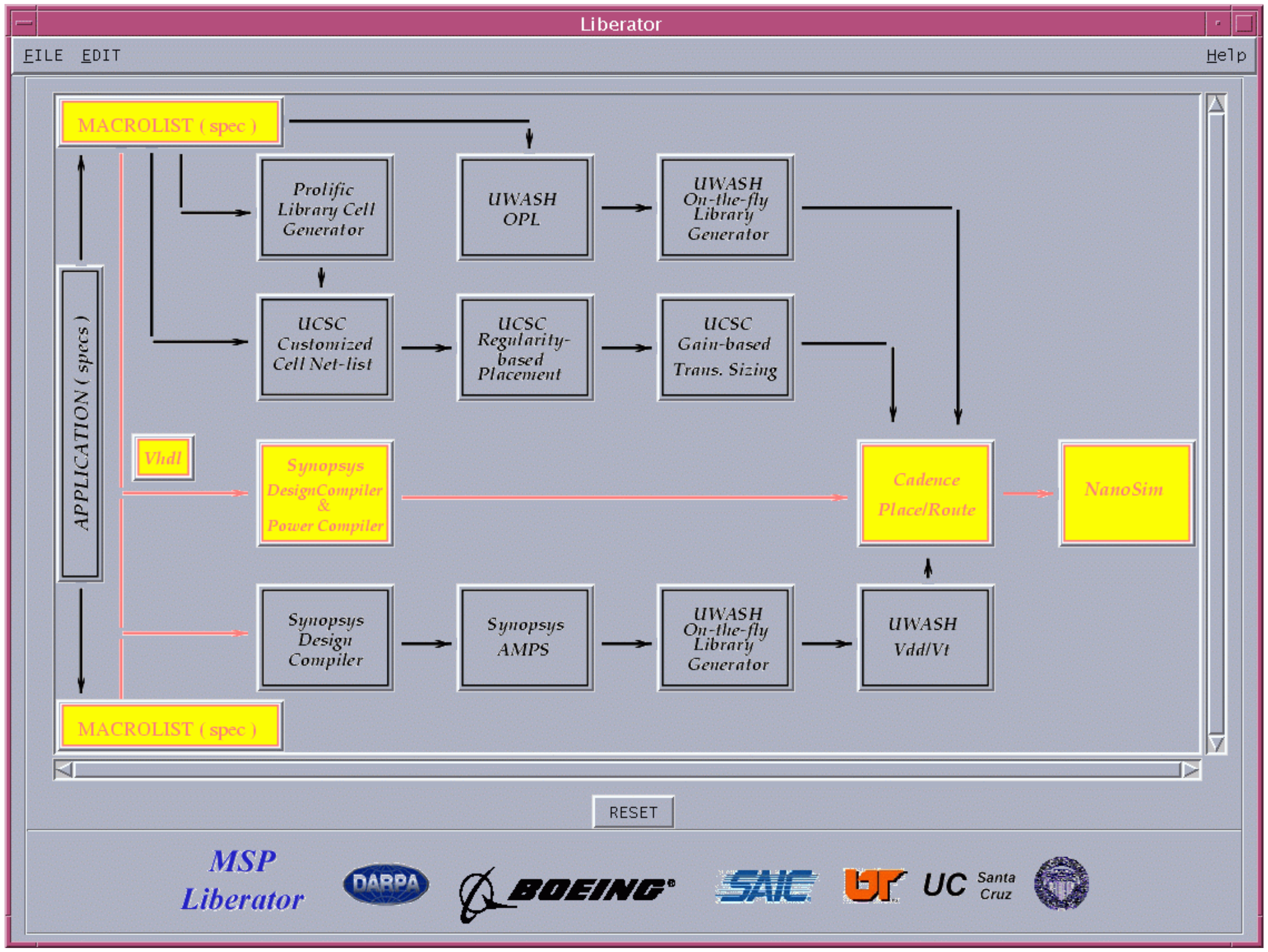

Liberator
FILE EDIT
Help
MACROLIST (spec)
Prolific Library Cell Generator
UWASH OPL
UWASH On-the-fly Library Generator
UCSC Customized Cell Net-list
UCSC Regularity-based Placement
UCSC Gain-based Trans. Sizing
APPLICATION (specs)
Vhdl
Synopsys DesignCompiler & Power Compiler
Cadence Place/Route
NanoSim
Synopsys Design Compiler
Synopsys AMPS
UWASH On-the-fly Library Generator
UWASH Vdd/Vt
MACROLIST (spec)
RESET
MSP Liberator
DARPA
BOEING
SAIC
UT
UC Santa Cruz

Figure 5.1: Main window of the GUI.

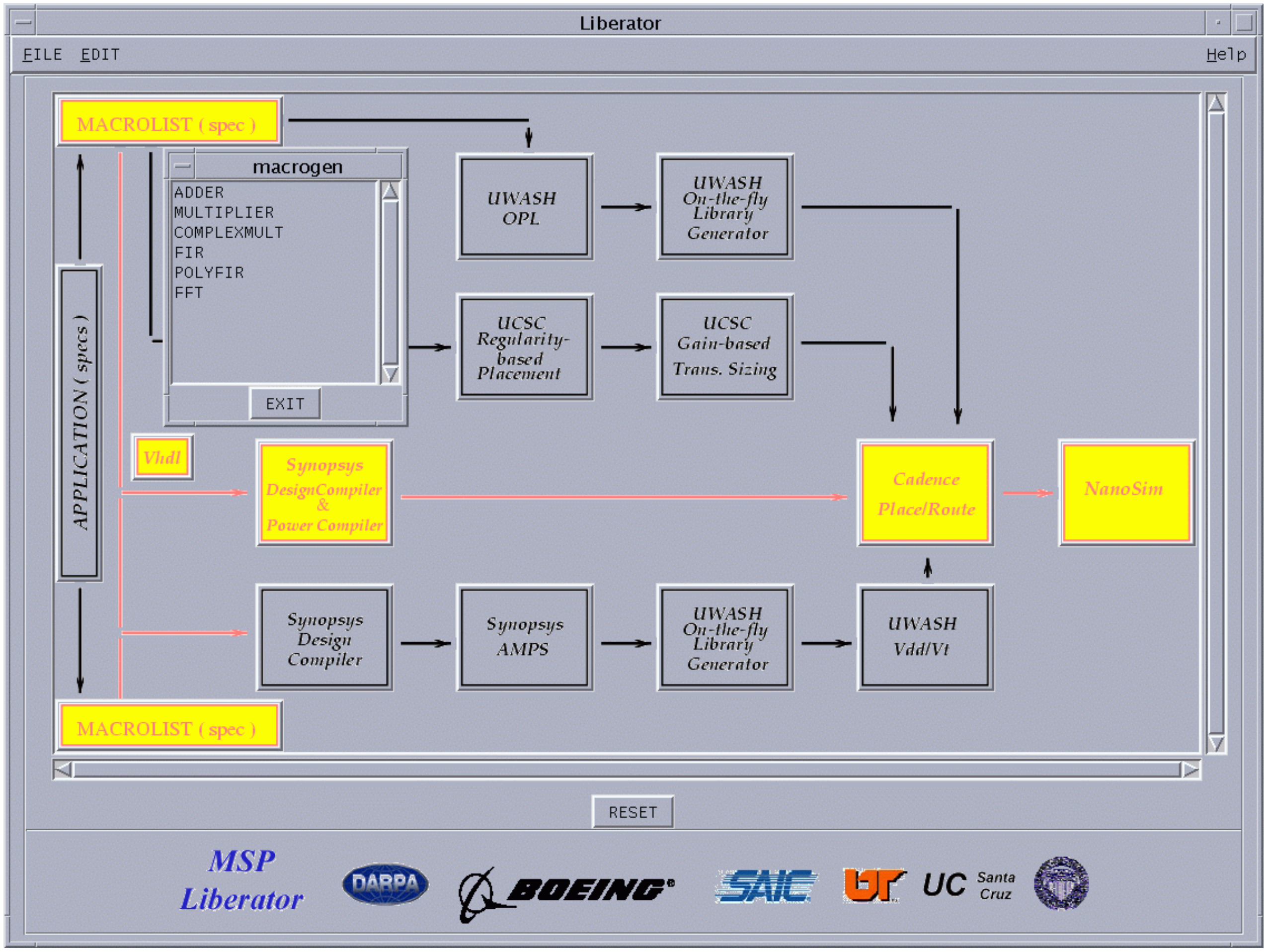


Figure 5.2: Initiating MacroGen from the main window.

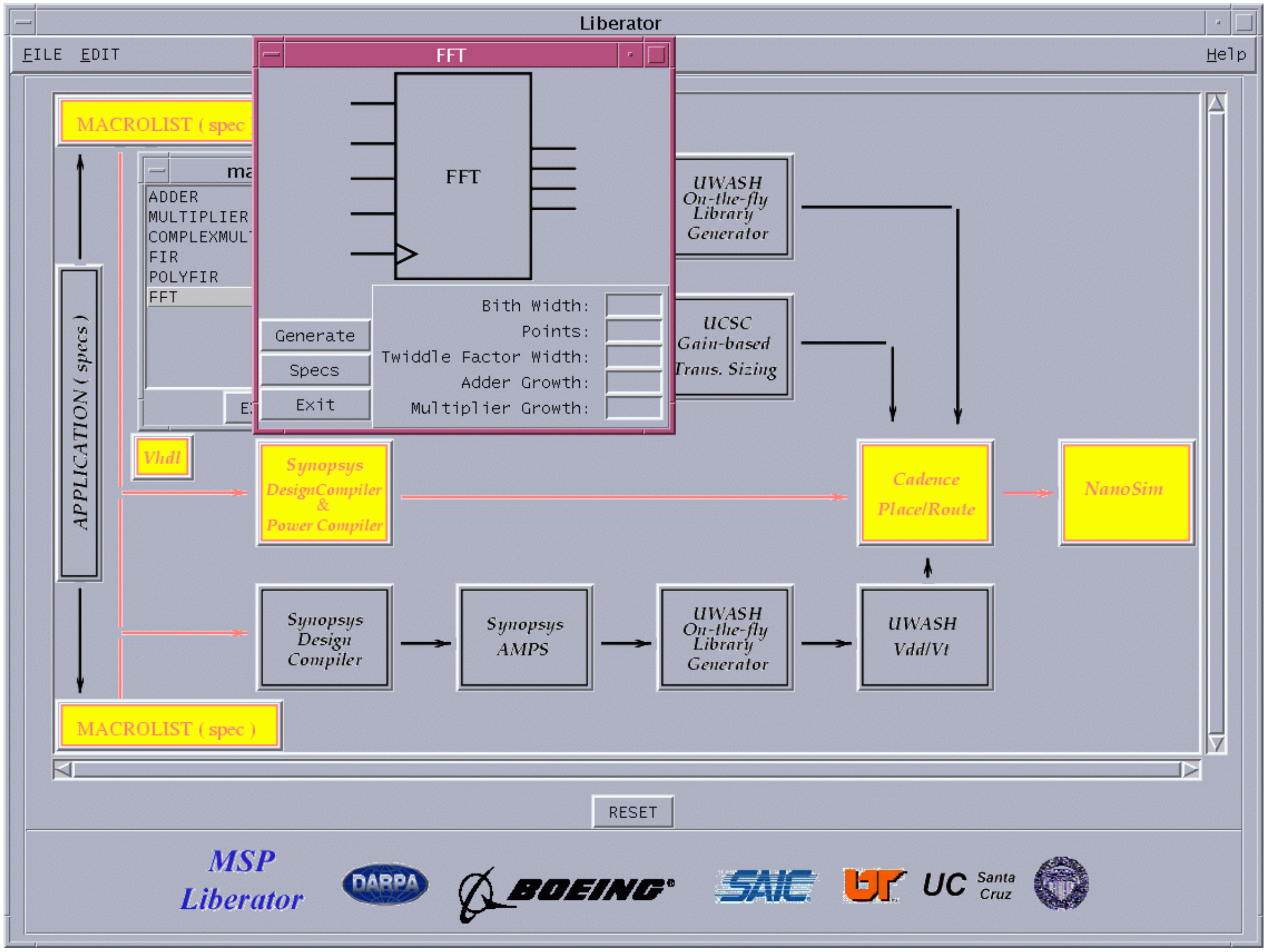
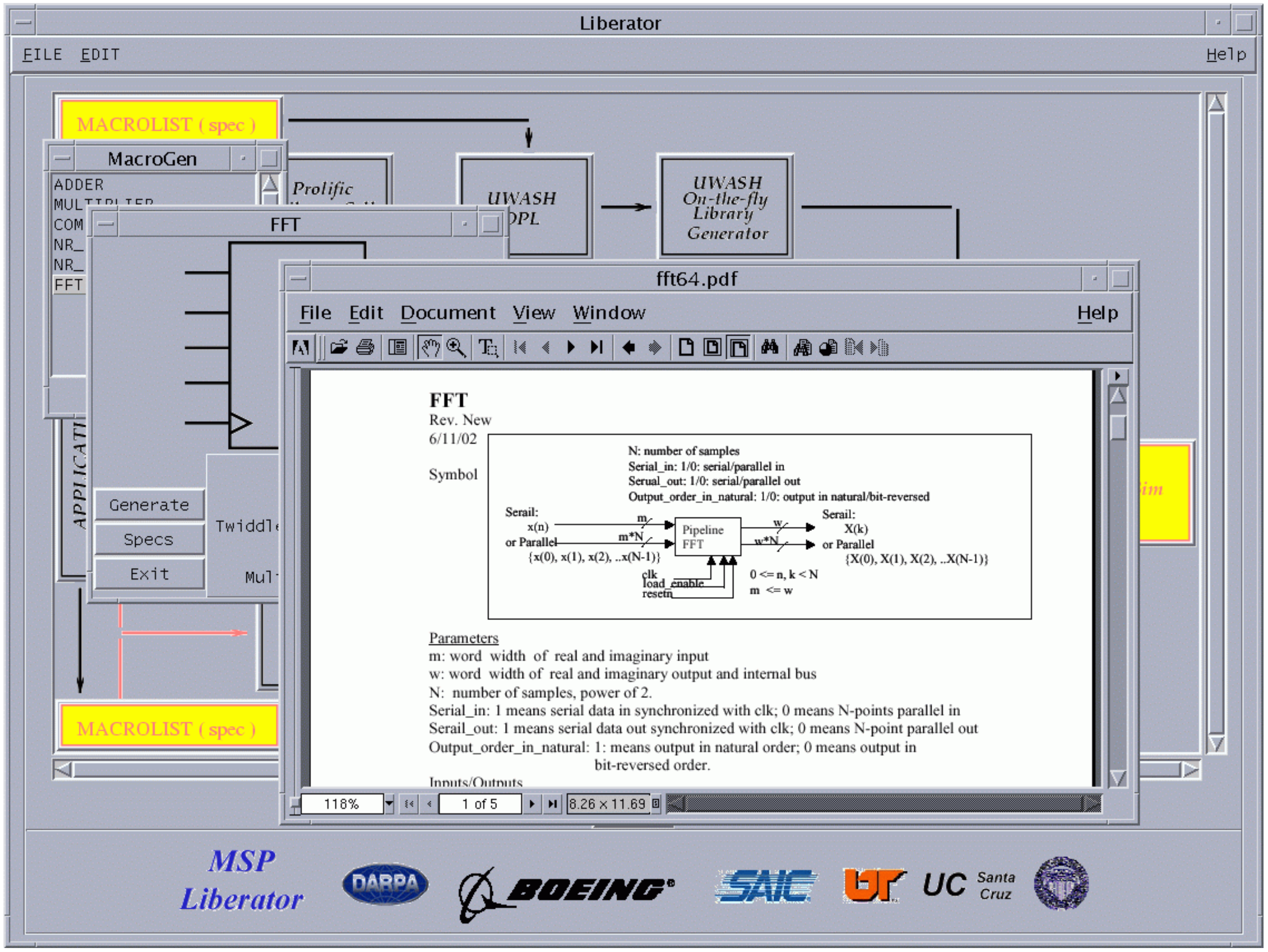


Figure 5.3: Selecting a macro from MacroGen.

DSP macro are entered through this window. Specification of the selected DSP macro can be viewed simply by clicking on the ”Specs” button in the macro window (Figure 5.4). After entering values for the generic parameters, clicking on the ”Generate” button (Figure 5.5) will generate the VHDL file for the macro according to the user specifications. The VHDL file for the macro can be viewed by clicking on the ”VHDL” icon in the main window (Figure 5.6).

5.2 Initiating Design Space Search

The Synopsys synthesis tools used during the technology mapping phase can generate, depending on the target technology library and user constraints, many design points with different power, area, delay values. Finding the best combination of power, area, delay may require several iterations. This iteration process can be very time-consuming and tedious. Several algorithms (DSSA, PDSSA, SA) have been developed at University of Tennessee to assist the synthesis tool to perform an automated design space search. This design space search can be performed according to user-defined parameters and is initiated by clicking on the ”Synopsys Design Compiler Power Compiler” icon. A new dialog window will pop-up (Figure 5.7) to let the user enter user-defined parameters for the design space search. The target technology can be selected from the radio button labeled ”Technology”. The desired run time (unit for the desired run time has to be selected using the radio button labeled as ”Unit” (Figure 5.7)), number of iterations, improvement ratio and step size can be entered using this dialog window. The figure of merit for the search can be selected using the



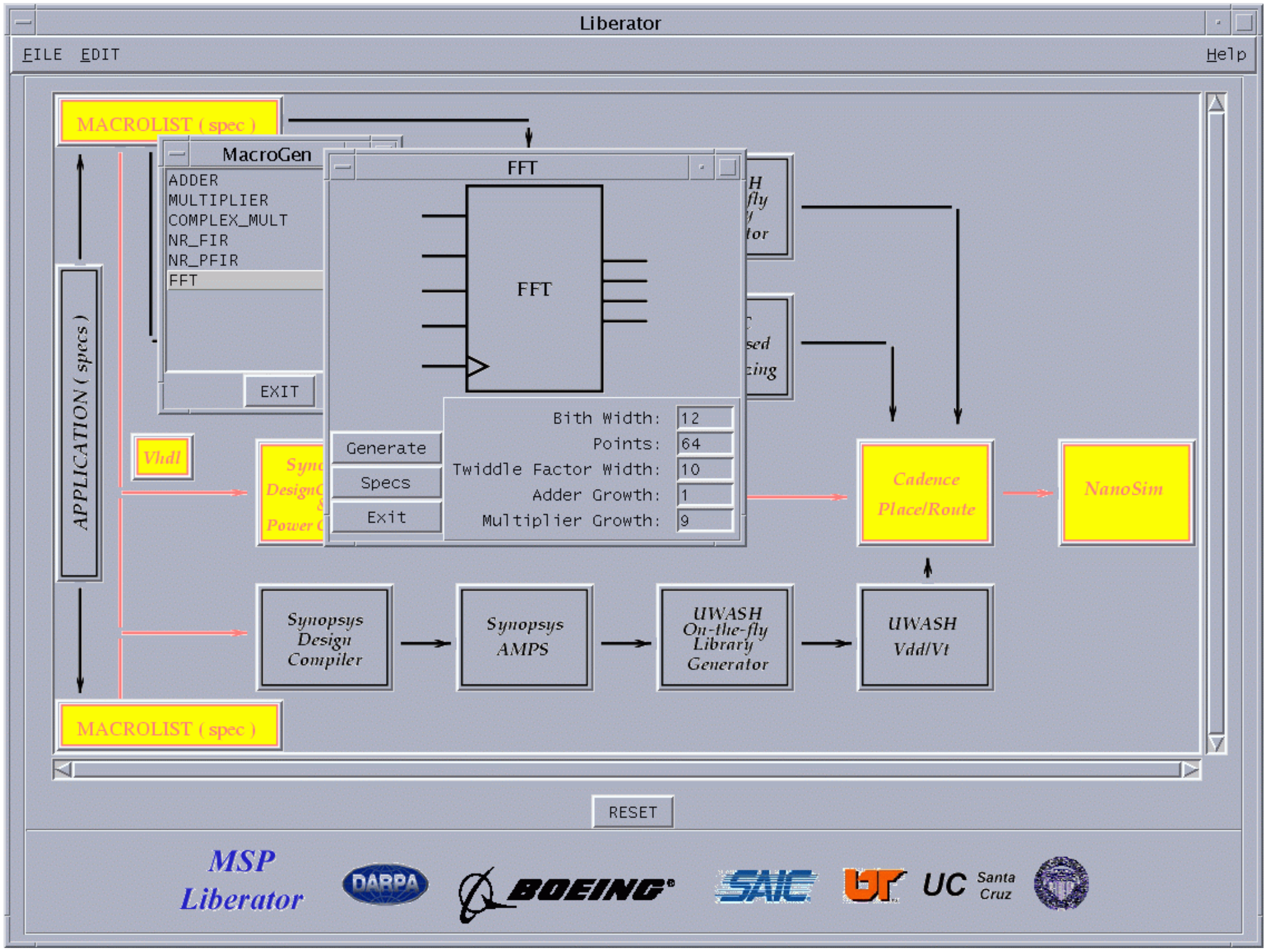
FFT
Rev. New
6/11/02

Symbol

N: number of samples
Serial_in: 1/0: serial/parallel in
Serual_out: 1/0: serial/parallel out
Output_order_in_natural: 1/0: output in natural/bit-reversed
Serail:
x(n)
or Parallel
{x(0), x(1), x(2), ..x(N-1)}
m
m*N
Pipeline
FFT
w
w*N
Serail:
X(k)
or Parallel
{X(0), X(1), X(2), ..X(N-1)}
clk
load_enable
resetn
0 <= n, k < N
m <= w

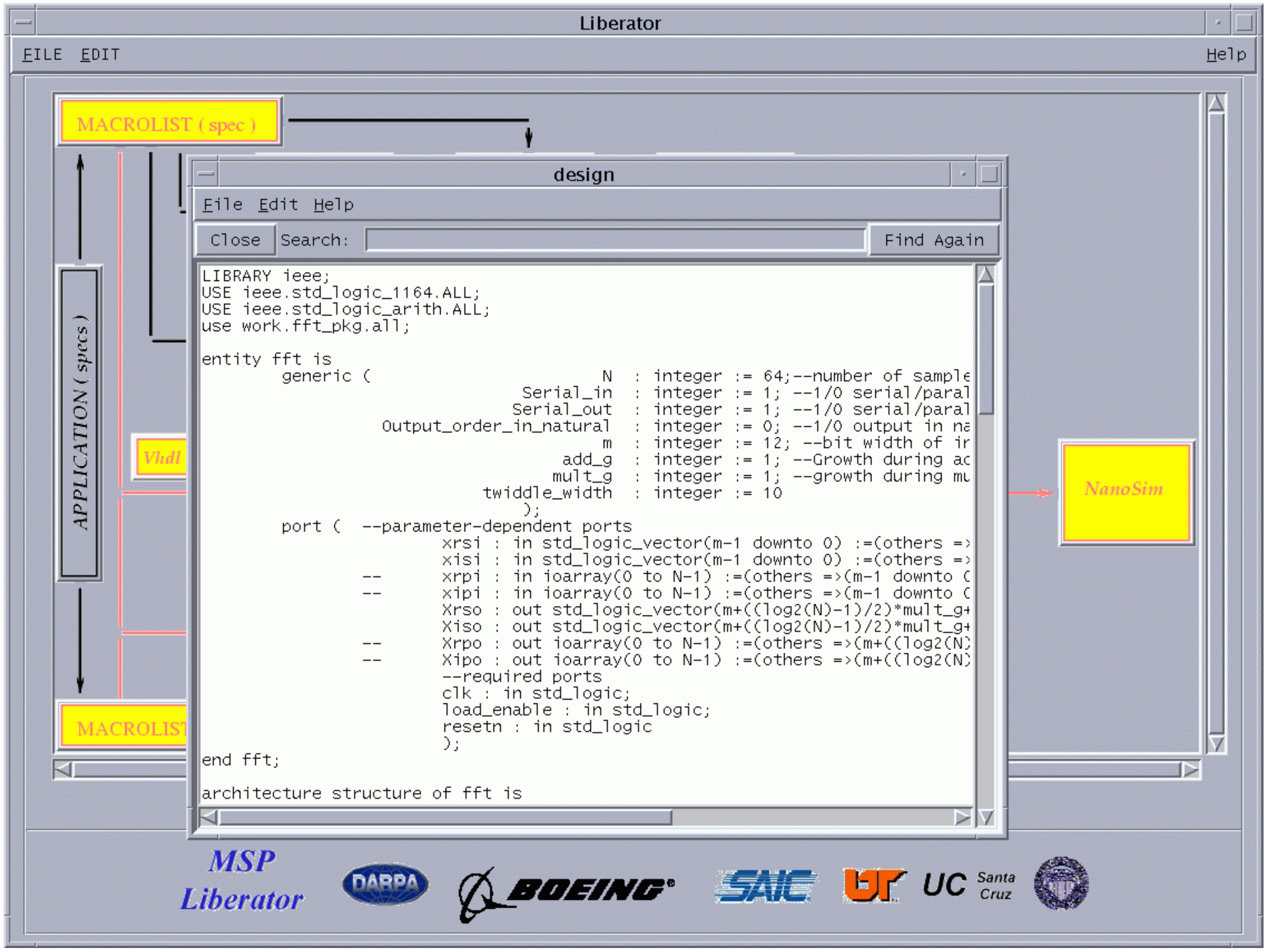
Parameters
m: word width of real and imaginary input
w: word width of real and imaginary output and internal bus
N: number of samples, power of 2.
Serial_in: 1 means serial data in synchronized with clk; 0 means N-points parallel in
Serail_out: 1 means serial data out synchronized with clk; 0 means N-point parallel out
Output_order_in_natural: 1: means output in natural order; 0 means output in bit-reversed order.
Inputs/Outputs

Figure 5.4: Viewing specifications for the selected macro.



Bith Width: 12
Points: 64
Twiddle Factor Width: 10
Adder Growth: 1
Multiplier Growth: 9

Figure 5.5: Entering values for the generic parameters of the selected macro.

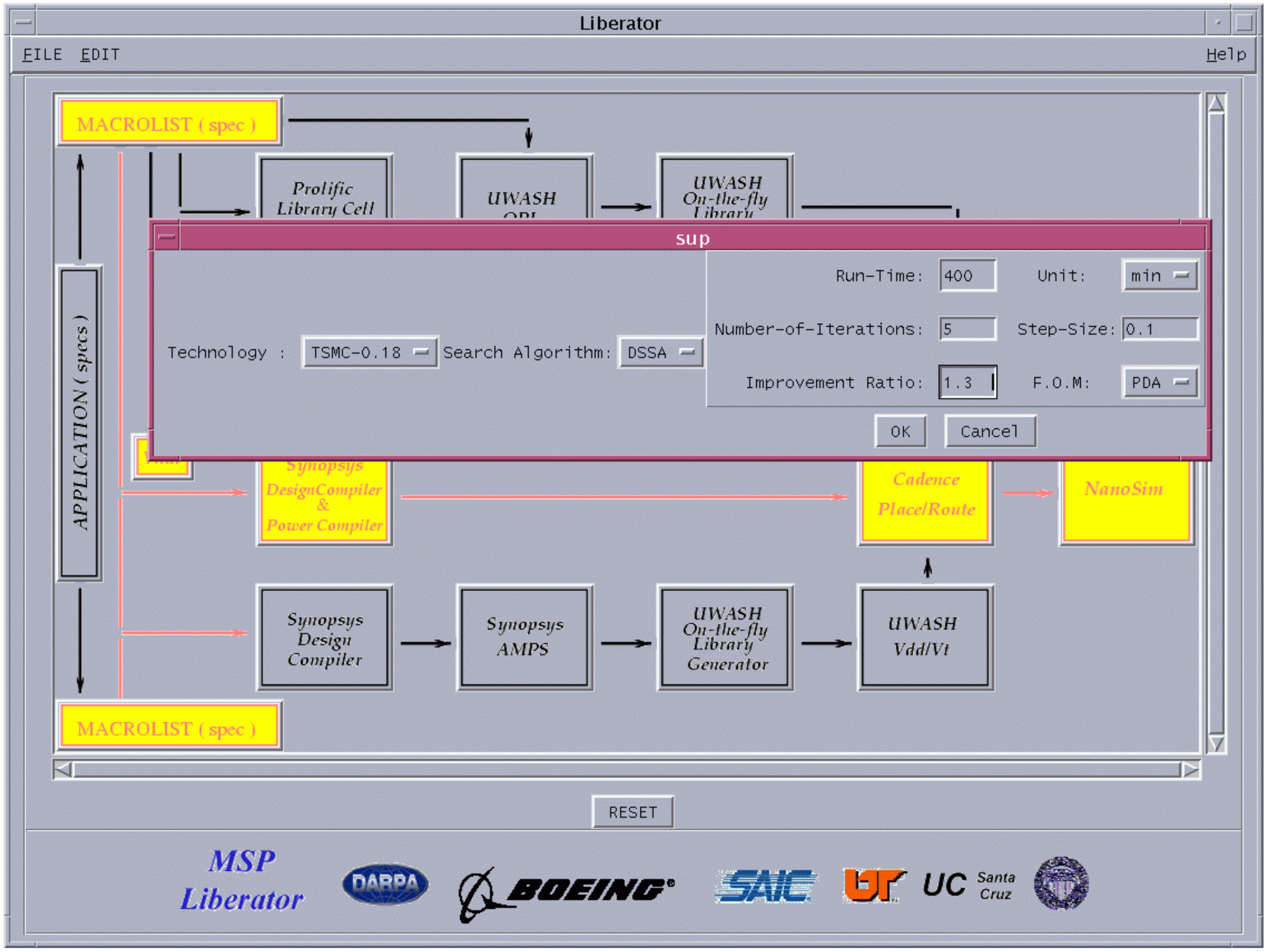


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
use work.fft_pkg.all;

entity fft is
        generic (                       N  : integer := 64;--number of sample
                                Serial_in  : integer := 1; --1/0 serial/paral
                               Serial_out  : integer := 1; --1/0 serial/paral
                  Output_order_in_natural  : integer := 0; --1/0 output in na
                                        m  : integer := 12; --bit width of ir
                                    add_g  : integer := 1; --Growth during ac
                                   mult_g  : integer := 1; --growth during mu
                            twiddle_width  : integer := 10
                                );
        port (  --parameter-dependent ports
                        xrsi : in std_logic_vector(m-1 downto 0) :=(others =>
                        xisi : in std_logic_vector(m-1 downto 0) :=(others =>
                --      xrpi : in ioarray(0 to N-1) :=(others =>(m-1 downto (
                --      xipi : in ioarray(0 to N-1) :=(others =>(m-1 downto (
                        Xrso : out std_logic_vector(m+((log2(N)-1)/2)*mult_g+
                        Xiso : out std_logic_vector(m+((log2(N)-1)/2)*mult_g+
                --      Xrpo : out ioarray(0 to N-1) :=(others =>(m+((log2(N)
                --      Xipo : out ioarray(0 to N-1) :=(others =>(m+((log2(N)
                        --required ports
                        clk : in std_logic;
                        load_enable : in std_logic;
                        resetn : in std_logic
                        );
end fft;

architecture structure of fft is
```

Figure 5.6: Viewing VHDL file of the selected macro.



Technology : TSMC-0.18

Search Algorithm: DSSA

Run-Time: 400 Unit: min

Number-of-Iterations: 5 Step-Size: 0.1

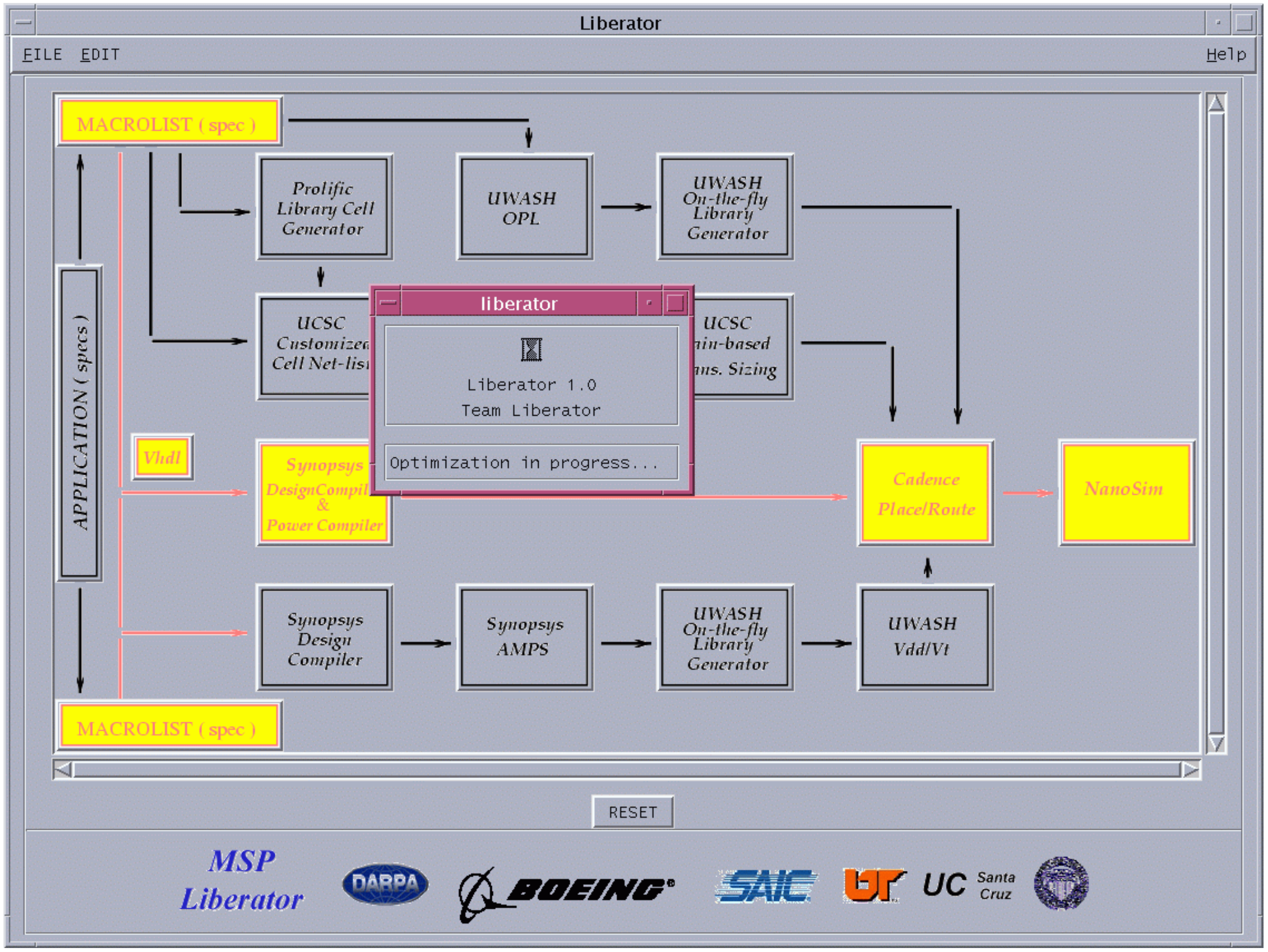
Improvement Ratio: 1.3 F.O.M: PDA

Figure 5.7: Initiating a design space search.

radio button labeled ”F.O.M” (Figure 5.7). The desired algorithm (DSSA, PDSSA, SA) for the search is selected using radio button labeled ”Search Algorithm”. Then, the automated design space search for the selected macro can be initiated by pressing the ”OK” button in the dialog window. After a few seconds, the dialog window will disappear and several splash windows will start popping up. The last splash window (Figure 5.8) will stay there until the search is complete. Depending on the user-defined parameters, this step may take several minutes or even hours. But since the process is completely automated, it can be left to run by itself.

5.3 Reporting and Analyzing Run Results

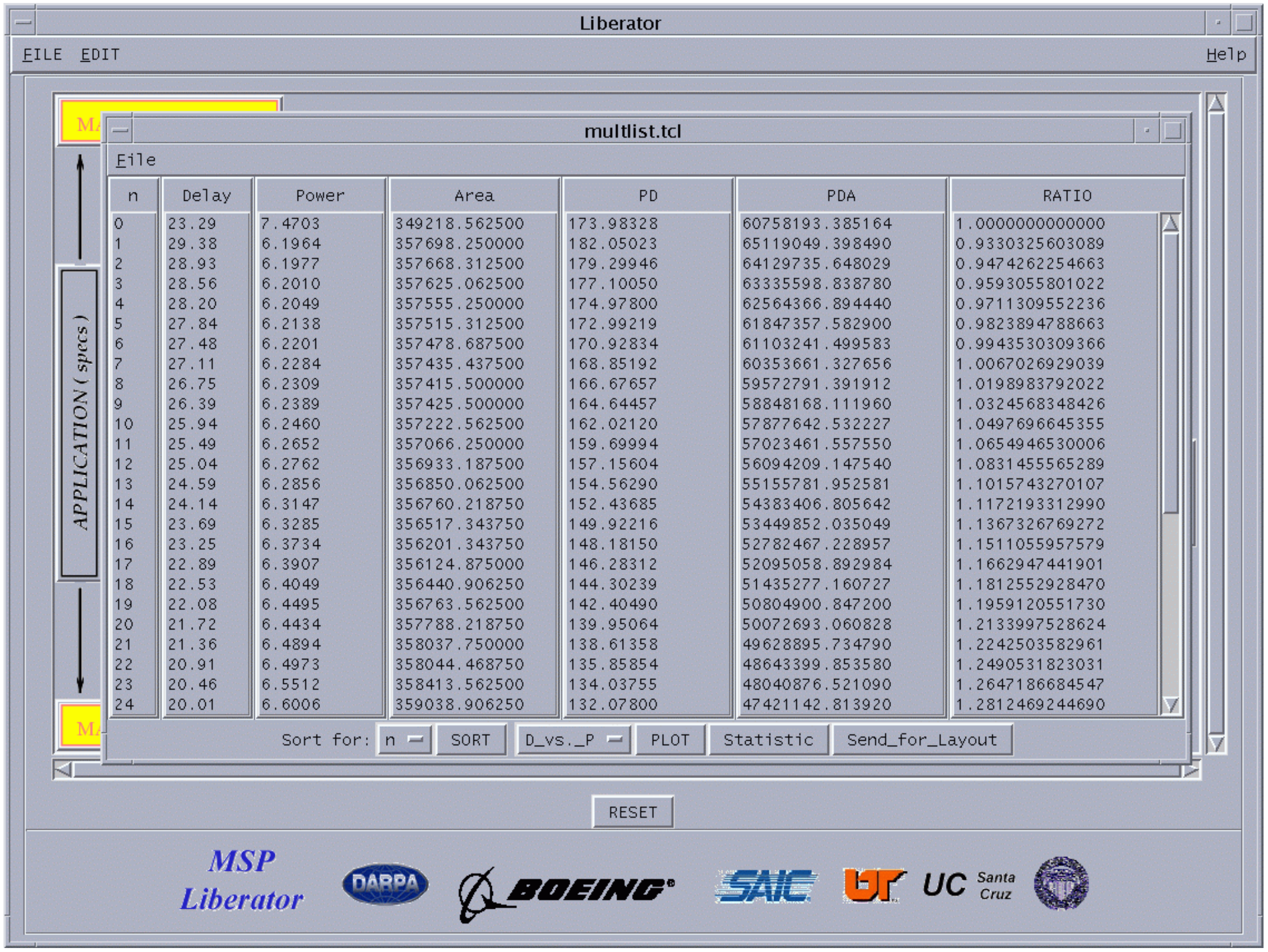
As soon as the design space search is over, a new window will pop up with search results (Figure 5.9). From this window one can sort the results with respect to the chosen figure of merit (power, delay, area, power-delay product, power-delay-area product). It is also possible to plot the results. To initiate a plot an option (D_vs_P, D_vs_A, P_vs_A, or D_vs_PDA) has to be selected from the radio button (Figure 5.9) located on the results window. After selecting the option, clicking on the ”PLOT” button will pop up a new window with the desired plot (Figure 5.10). Plotting the search results will help the designer to visualize the design space. A Pareto curve fitting algorithm has been also integrated into the GUI. Fitting a Pareto curve to the desired plot can be done simply by clicking on the ”Pareto_Curve” (Figure 5.11) button in the plot window. A statistic about each design point can be obtained using ”Statistic” button in the results window. This button will pop up a text window with the report (Figure 5.12) which



Liberator 1.0
Team Liberator

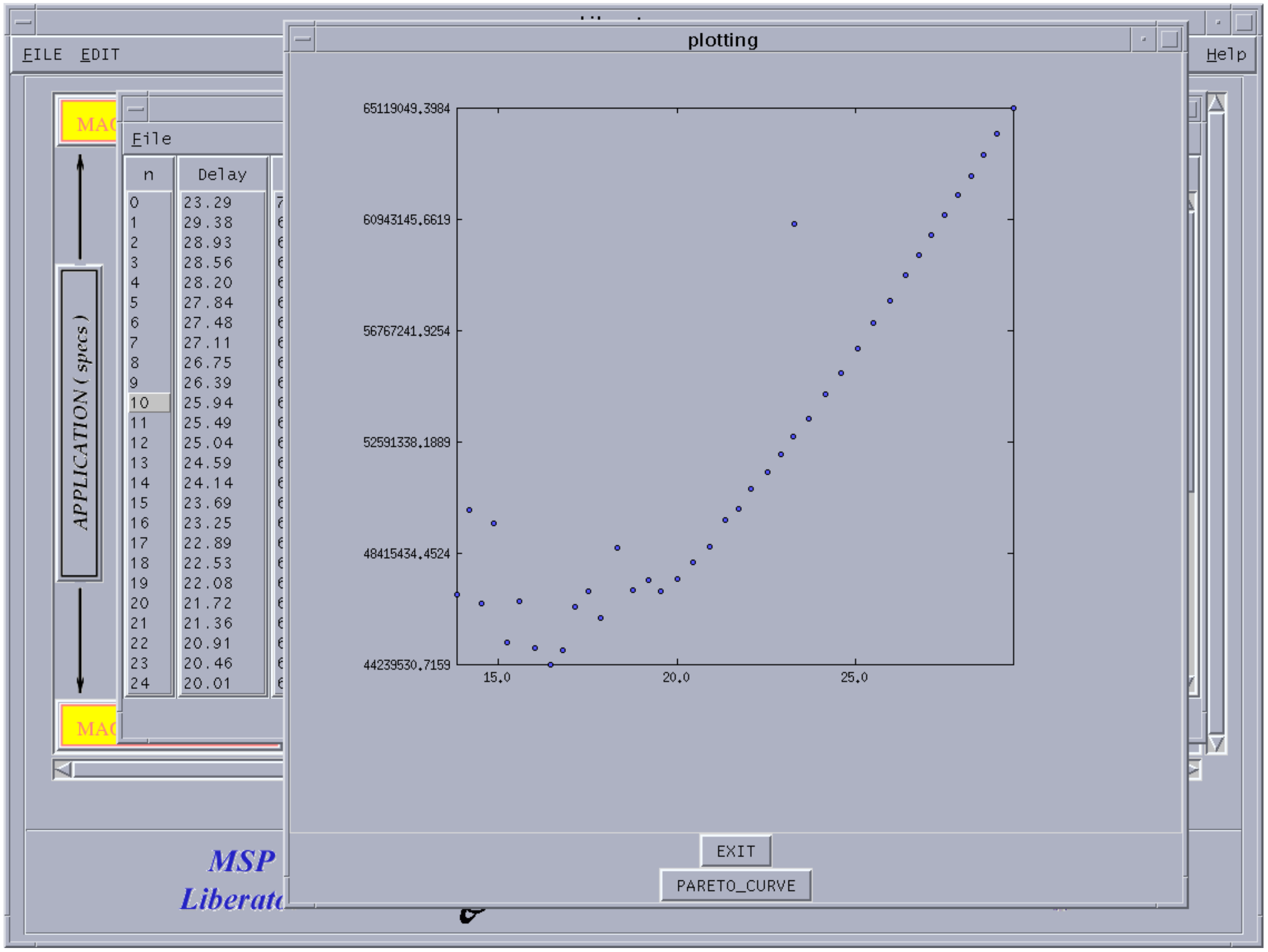
Optimization in progress...

Figure 5.8: Design space search in progress.



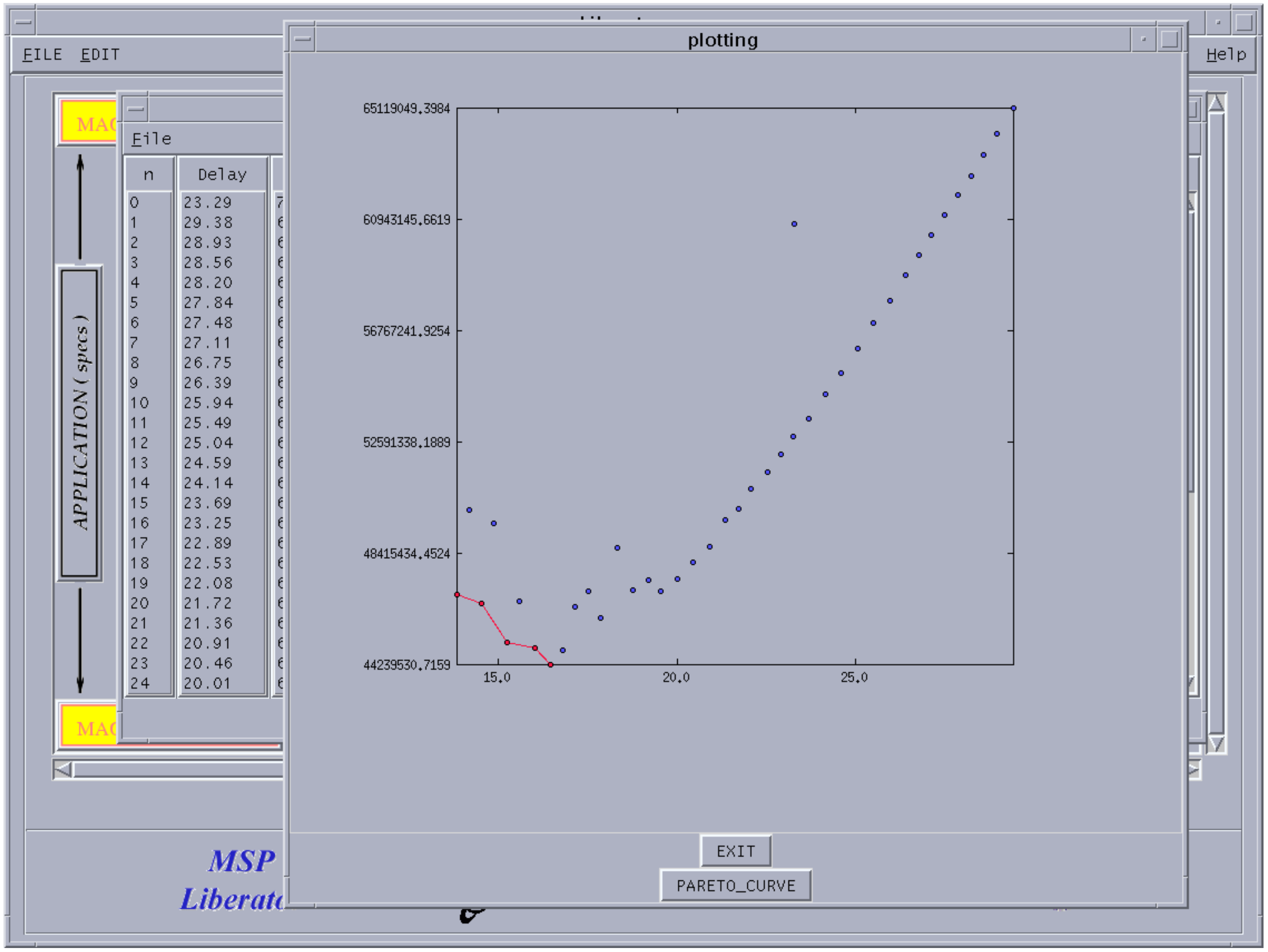
n	Delay	Power	Area	PD	PDA	RATIO
0	23.29	7.4703	349218.562500	173.98328	60758193.385164	1.0000000000000
1	29.38	6.1964	357698.250000	182.05023	65119049.398490	0.9330325603089
2	28.93	6.1977	357668.312500	179.29946	64129735.648029	0.9474262254663
3	28.56	6.2010	357625.062500	177.10050	63335598.838780	0.9593055801022
4	28.20	6.2049	357555.250000	174.97800	62564366.894440	0.9711309552236
5	27.84	6.2138	357515.312500	172.99219	61847357.582900	0.9823894788663
6	27.48	6.2201	357478.687500	170.92834	61103241.499583	0.9943530309366
7	27.11	6.2284	357435.437500	168.85192	60353661.327656	1.0067026929039
8	26.75	6.2309	357415.500000	166.67657	59572791.391912	1.0198983792022
9	26.39	6.2389	357425.500000	164.64457	58848168.111960	1.0324568348426
10	25.94	6.2460	357222.562500	162.02120	57877642.532227	1.0497696645355
11	25.49	6.2652	357066.250000	159.69994	57023461.557550	1.0654946530006
12	25.04	6.2762	356933.187500	157.15604	56094209.147540	1.0831455565289
13	24.59	6.2856	356850.062500	154.56290	55155781.952581	1.1015743270107
14	24.14	6.3147	356760.218750	152.43685	54383406.805642	1.1172193312990
15	23.69	6.3285	356517.343750	149.92216	53449852.035049	1.1367326769272
16	23.25	6.3734	356201.343750	148.18150	52782467.228957	1.1511055957579
17	22.89	6.3907	356124.875000	146.28312	52095058.892984	1.1662947441901
18	22.53	6.4049	356440.906250	144.30239	51435277.160727	1.1812552928470
19	22.08	6.4495	356763.562500	142.40490	50804900.847200	1.1959120551730
20	21.72	6.4434	357788.218750	139.95064	50072693.060828	1.2133997528624
21	21.36	6.4894	358037.750000	138.61358	49628895.734790	1.2242503582961
22	20.91	6.4973	358044.468750	135.85854	48643399.853580	1.2490531823031
23	20.46	6.5512	358413.562500	134.03755	48040876.521090	1.2647186684547
24	20.01	6.6006	359038.906250	132.07800	47421142.813920	1.2812469244690

Figure 5.9: Tabulated design space search results for the selected macro.



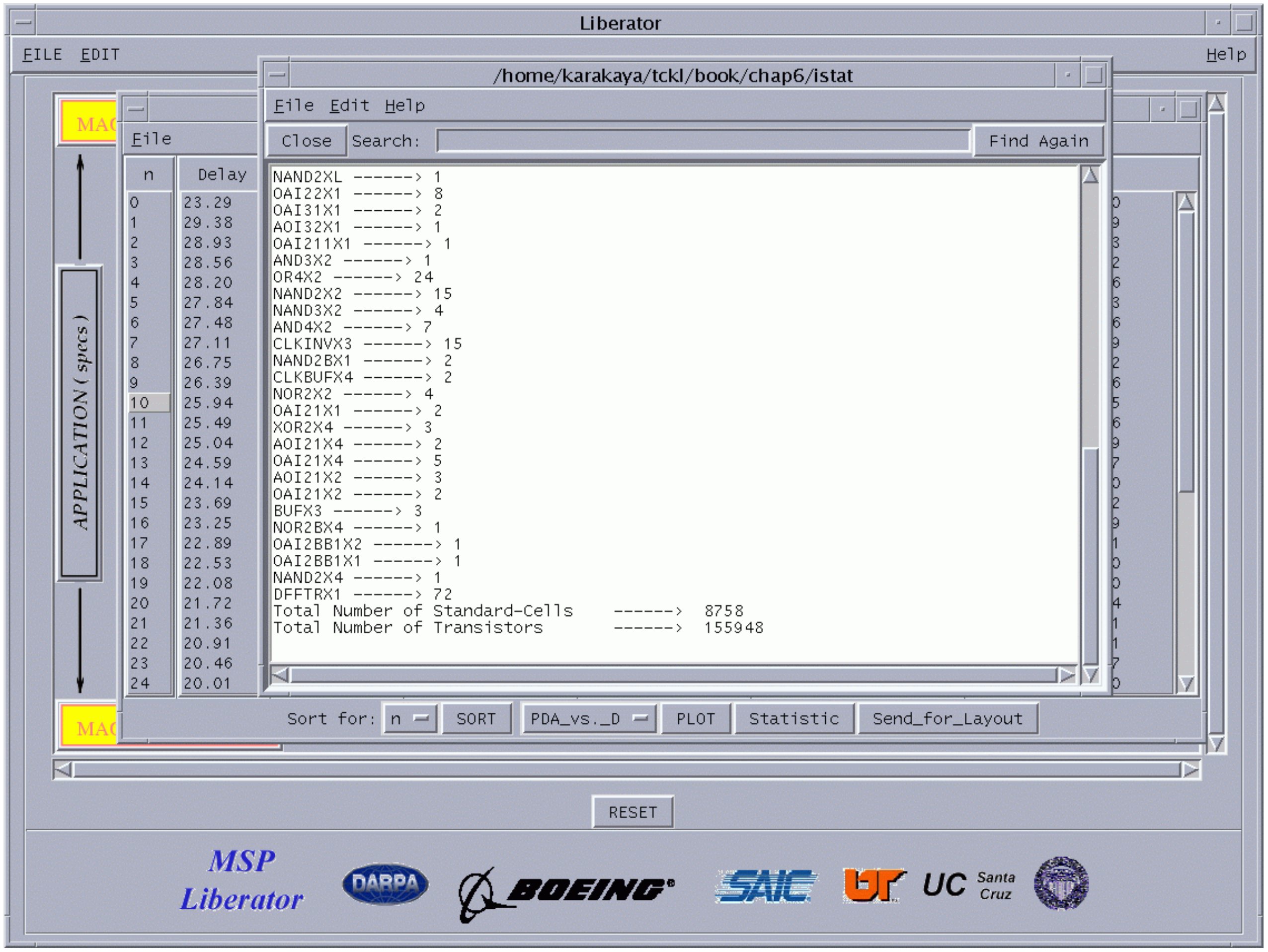
n	Delay
0	23.29
1	29.38
2	28.93
3	28.56
4	28.20
5	27.84
6	27.48
7	27.11
8	26.75
9	26.39
10	25.94
11	25.49
12	25.04
13	24.59
14	24.14
15	23.69
16	23.25
17	22.89
18	22.53
19	22.08
20	21.72
21	21.36
22	20.91
23	20.46
24	20.01

Figure 5.10: Plotting search results for the selected parameters.



n	Delay
0	23.29
1	29.38
2	28.93
3	28.56
4	28.20
5	27.84
6	27.48
7	27.11
8	26.75
9	26.39
10	25.94
11	25.49
12	25.04
13	24.59
14	24.14
15	23.69
16	23.25
17	22.89
18	22.53
19	22.08
20	21.72
21	21.36
22	20.91
23	20.46
24	20.01

Figure 5.11: Generating Pareto curve.



n	Delay
0	23.29
1	29.38
2	28.93
3	28.56
4	28.20
5	27.84
6	27.48
7	27.11
8	26.75
9	26.39
10	25.94
11	25.49
12	25.04
13	24.59
14	24.14
15	23.69
16	23.25
17	22.89
18	22.53
19	22.08
20	21.72
21	21.36
22	20.91
23	20.46
24	20.01

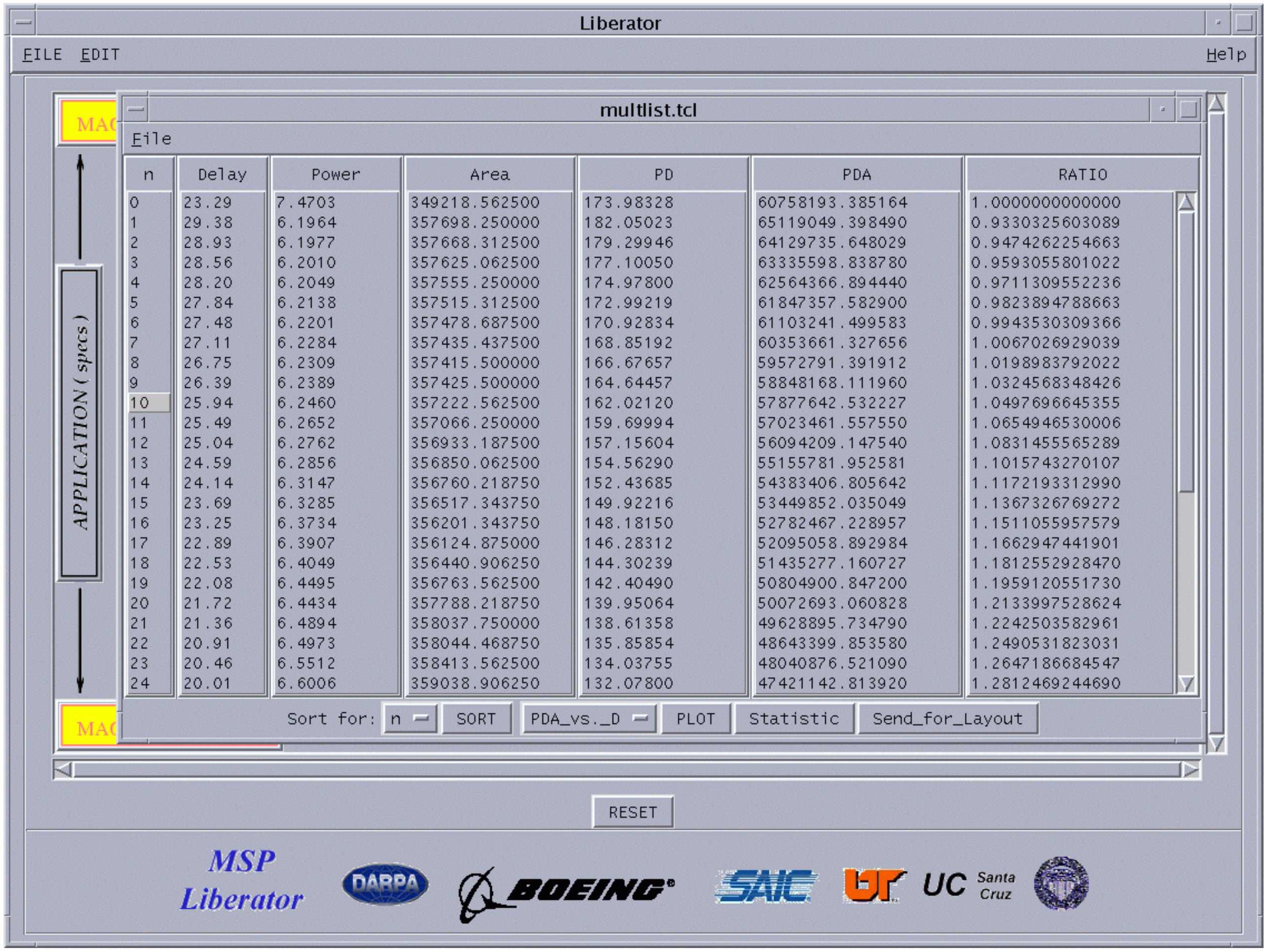
```
NAND2XL ------> 1
OAI22X1 ------> 8
OAI31X1 ------> 2
AOI32X1 ------> 1
OAI211X1 ------> 1
AND3X2 ------> 1
OR4X2 ------> 24
NAND2X2 ------> 15
NAND3X2 ------> 4
AND4X2 ------> 7
CLKINVX3 ------> 15
NAND2BX1 ------> 2
CLKBUFX4 ------> 2
NOR2X2 ------> 4
OAI21X1 ------> 2
XOR2X4 ------> 3
AOI21X4 ------> 2
OAI21X4 ------> 5
AOI21X2 ------> 3
OAI21X2 ------> 2
BUFX3 ------> 3
NOR2BX4 ------> 1
OAI2BB1X2 ------> 1
OAI2BB1X1 ------> 1
NAND2X4 ------> 1
DFFTRX1 ------> 72
Total Number of Standard-Cells     ------>  8758
Total Number of Transistors        ------>  155948
```

Figure 5.12: Viewing the statistics of the standard-cells, number of transistors for the selected design point.

will include each standard-cell used in the design and its number of instances, the total number of standard-cells used, and the total number of transistors for the selected design point. It is also possible to select a design point from the list and send it for layout. The desired design point can be selected by clicking on its iteration number (column ”n” (Figure 5.13)). After selecting the design point, it can be sent to layout by clicking on the ”Send_for_Layout” button in the results window.

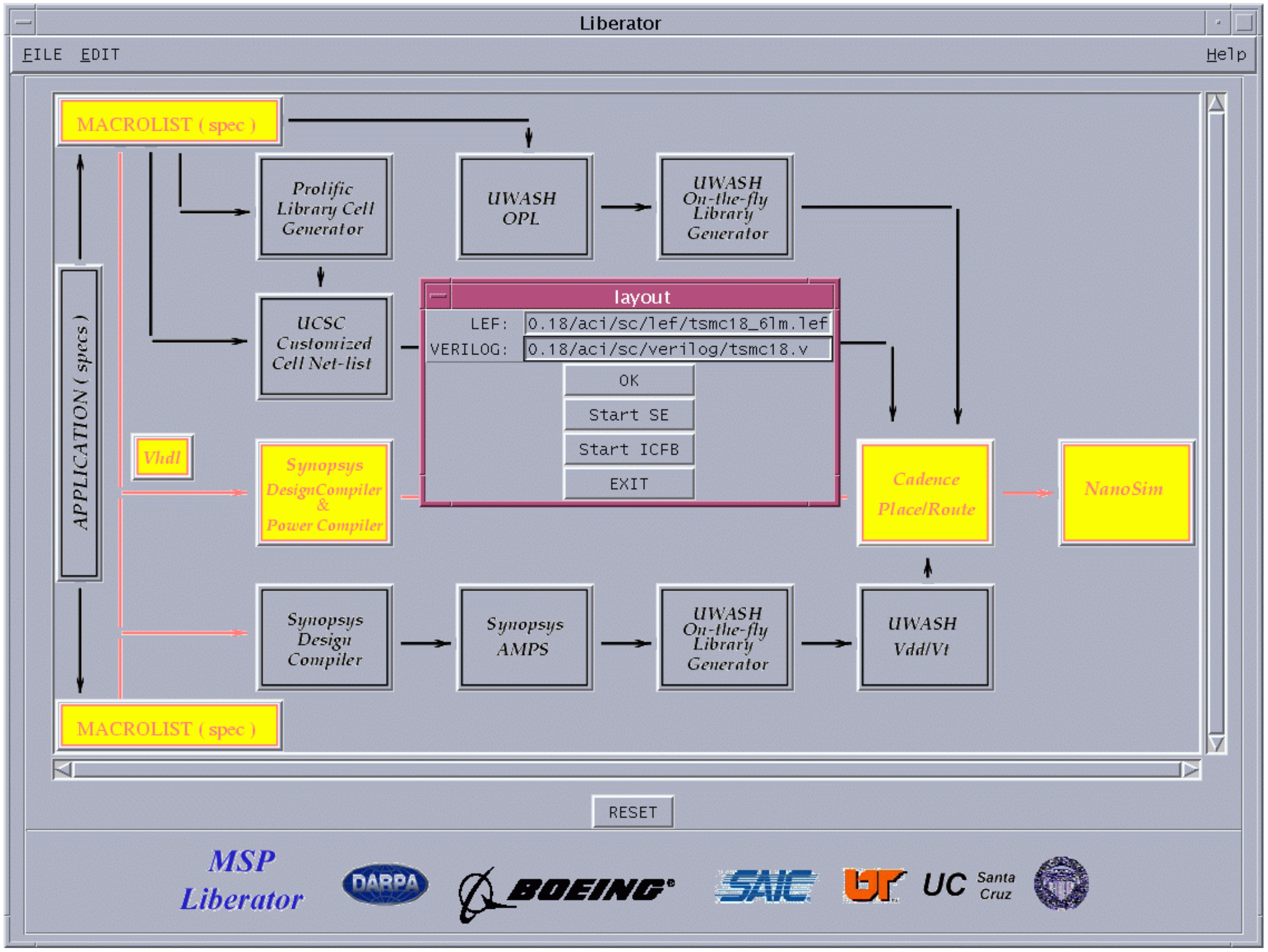
5.4 Initiating Layout

The layout process can be started by clicking on the ”Cadence PlaceRoute” icon. This will pop up a layout dialog window (Figure 5.14). After all the necessary files are entered, one can start Silicon Ensemble from the ”Start SE” button in the layout window (Figure 5.14). After starting SE, the designer may have to wait several seconds because SE has to read in the LEF file, the library VERILOG file, and the selected design point. All of these actions will be completed automatically. When all of the files have been read, the designer can interact with Silicon Ensemble to complete the rest of the place and route process. After the place and route is complete (Figure 5.15), the design has to be exported as a DEF file. The name of the DEF should be the same as the macro name. To finalize the layout process, the designer should click on the ”Start ICFB” button in the layout dialog window (Figure 5.14). This will initiate the Cadence Design Framework and the DEF file for the macro will be read in automatically. After importing the DEF file is complete, the final layout for the macro will pop up (Figure 5.16).



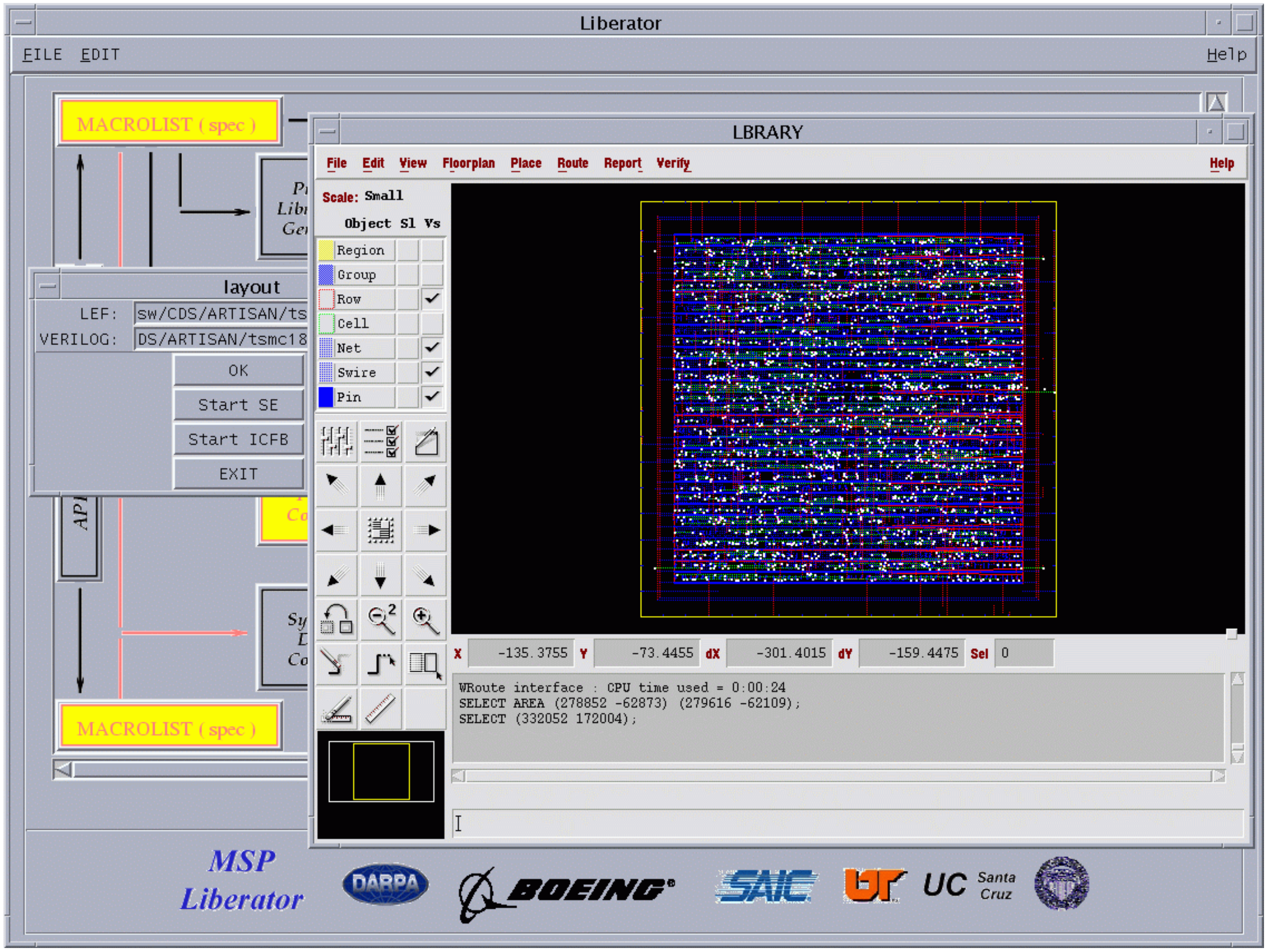
n	Delay	Power	Area	PD	PDA	RATIO
0	23.29	7.4703	349218.562500	173.98328	60758193.385164	1.000000000000
1	29.38	6.1964	357698.250000	182.05023	65119049.398490	0.9330325603089
2	28.93	6.1977	357668.312500	179.29946	64129735.648029	0.9474262254663
3	28.56	6.2010	357625.062500	177.10050	63335598.838780	0.9593055801022
4	28.20	6.2049	357555.250000	174.97800	62564366.894440	0.9711309552236
5	27.84	6.2138	357515.312500	172.99219	61847357.582900	0.9823894788663
6	27.48	6.2201	357478.687500	170.92834	61103241.499583	0.9943530309366
7	27.11	6.2284	357435.437500	168.85192	60353661.327656	1.0067026929039
8	26.75	6.2309	357415.500000	166.67657	59572791.391912	1.0198983792022
9	26.39	6.2389	357425.500000	164.64457	58848168.111960	1.0324568348426
10	25.94	6.2460	357222.562500	162.02120	57877642.532227	1.0497696645355
11	25.49	6.2652	357066.250000	159.69994	57023461.557550	1.0654946530006
12	25.04	6.2762	356933.187500	157.15604	56094209.147540	1.0831455565289
13	24.59	6.2856	356850.062500	154.56290	55155781.952581	1.1015743270107
14	24.14	6.3147	356760.218750	152.43685	54383406.805642	1.1172193312990
15	23.69	6.3285	356517.343750	149.92216	53449852.035049	1.1367326769272
16	23.25	6.3734	356201.343750	148.18150	52782467.228957	1.1511055957579
17	22.89	6.3907	356124.875000	146.28312	52095058.892984	1.1662947441901
18	22.53	6.4049	356440.906250	144.30239	51435277.160727	1.1812552928470
19	22.08	6.4495	356763.562500	142.40490	50804900.847200	1.1959120551730
20	21.72	6.4434	357788.218750	139.95064	50072693.060828	1.2133997528624
21	21.36	6.4894	358037.750000	138.61358	49628895.734790	1.2242503582961
22	20.91	6.4973	358044.468750	135.85854	48643399.853580	1.2490531823031
23	20.46	6.5512	358413.562500	134.03755	48040876.521090	1.2647186684547
24	20.01	6.6006	359038.906250	132.07800	47421142.813920	1.2812469244690

Figure 5.13: Selecting a design point to send for layout.



LEF: 0.18/aci/sc/lef/tsmc18_6lm.lef
VERILOG: 0.18/aci/sc/verilog/tsmc18.v

Figure 5.14: Initiating place and route procedure.

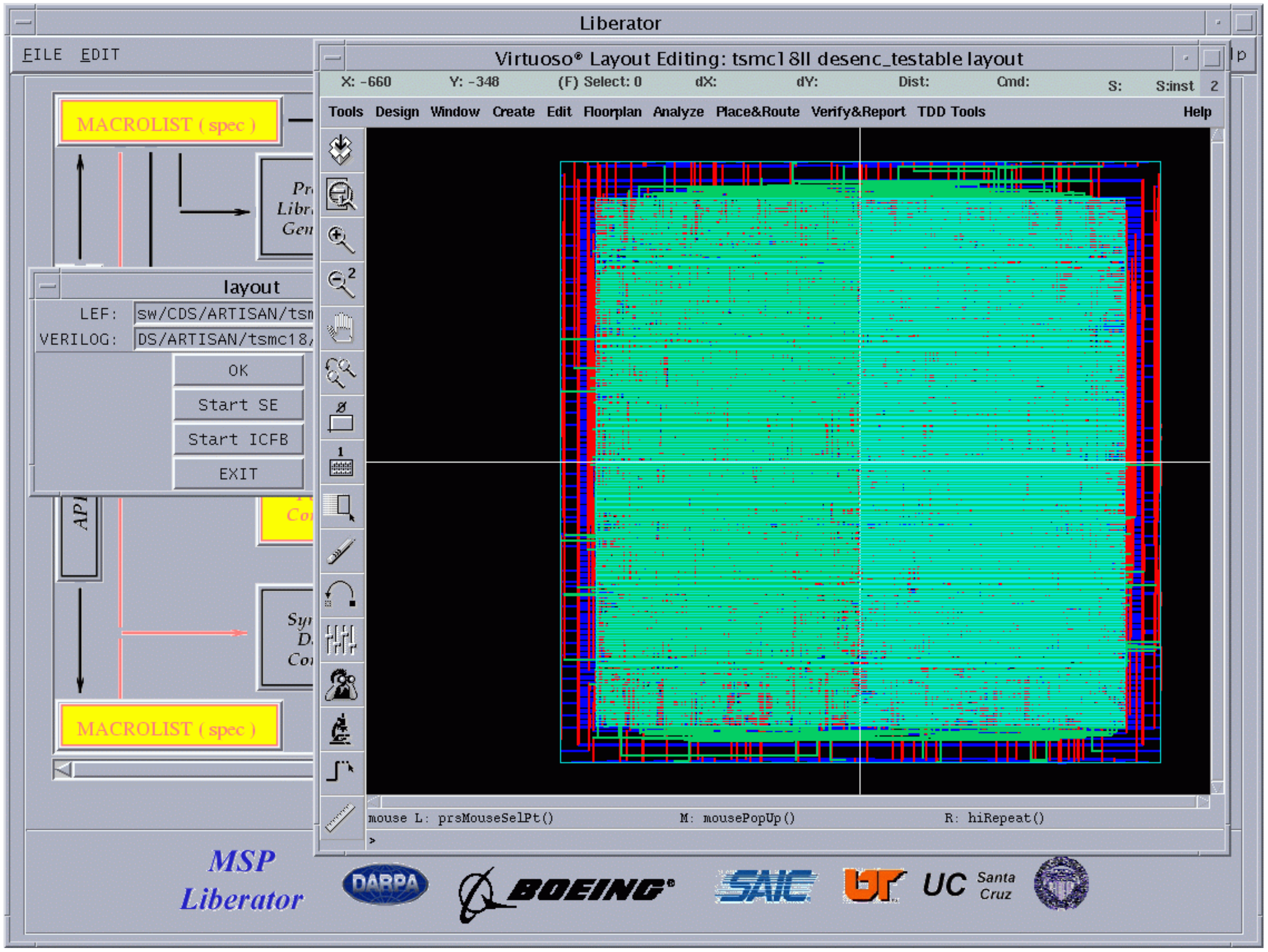


```
LEF:     sw/CDS/ARTISAN/ts
VERILOG: DS/ARTISAN/tsmc18
```

```
X -135.3755  Y -73.4455  dX -301.4015  dY -159.4475  Sel 0
```

```
WRoute interface : CPU time used = 0:00:24
SELECT AREA (278852 -62873) (279616 -62109);
SELECT (332052 172004);
```

Figure 5.15: Selected design point after placement and routing.


Liberator
FILE EDIT
Virtuoso® Layout Editing: tsmc18ll desenc_testable layout
X: -660 Y: -348 (F) Select: 0 dX: dY: Dist: Cmd: S: S:inst 2
Tools Design Window Create Edit Floorplan Analyze Place&Route Verify&Report TDD Tools Help
MACROLIST (spec)
layout
LEF: sw/CDS/ARTISAN/tsm
VERILOG: DS/ARTISAN/tsmc18/
OK
Start SE
Start ICFB
EXIT
MACROLIST (spec)
MSP Liberator

```
mouse L: prsMouseSelPt()          M: mousePopUp()          R: hiRepeat()
>
```

Figure 5.16: Final layout for the selected design point.

CHAPTER 6

Summary and Future Work

Two important parameters to measure integrated circuit efficiency are operations per second per watt (OPS/W) and operations per second per square micron (OPS/μ^2). The first parameter is an indication of energy efficiency of the chip and the second parameter is an indication of the area efficiency of the chip. Both parameters are closely related to the power, delay and area values of the design. OPS/W is inversely proportional to the product of delay and power (DPP), and OPS/μ^2 is inversely proportional to the product of area and delay. To increase operations per second per watt, DPP has to be minimized but at the same time silicon area also has to be kept in mind. Thus, minimization of the product of power times delay times area (PDA) can be a more efficient means of increasing OPS/W without hurting OPS/μ^2 efficiency. Experimental results presented in this dissertation show that optimizing the design for PDA gives the best GOPS/W with an insignificant effect on GOPS/μ^2. Experimental results also indicate that maximum GOPS/W occurs when the PDAP is minimum. As a result for an efficient chip design, designer has to find the design point on the design space where PDAP is minimum. To do that the designer may have to search the entire design space. Searching the design space manually can be a very time-consuming and tedious process. An automated design space search can be a life saver for the designer.

Unfortunately today's sophisticated synthesis tools do not provide any solution for this problem.

To overcome this obstacle, several algorithms have been developed and implemented at the University of Tennessee at Knoxville and presented in this thesis. The main contributions of this dissertation include:

- Developed an algorithm (DSSA) to assist or to guide a commercial tool to do a fully automated design space search.
- The developed algorithm searches the design space with user guidance in terms of:
  1. search time,
  2. number of iterations, and/or
  3. improvement ratio.
- A parallel version of DSSA (PDSSA) has been implemented to reduce the search time.
- Several optimization methods at the RTL level have been investigated and three of them have been applied to the selected designs to conduct design space searches.
- To minimize the design space which characterizes the design, a Pareto point selection algorithm has been added to the DSSA and PDSSA.

- The developed search techniques are integrated into an automated ASIC design flow.
- A search technique which utilizes Simulated Annealing has been investigated.
- A graphical user interface (GUI) has been developed for the automated flow.
- All developed algorithms, the automated ASIC flow and the GUI have been tested and verified using several DSP macros.

In general, the design space search algorithms, the automated ASIC flow, and GUI introduced in this thesis have created a faster and automated way of searching the design space for hardware implementation. In some cases, the found PDA product was 1.9 times better than the initial baseline solution as a result of the optimization techniques applied during the design space search. The parallel search algorithm exhibited 9x speed up when executed on 10 machines simultaneously.

One possibility of extending this dissertation is to include the design space search at the behavioral level and transistor level.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] ASIC Design Methology Primer. *IBM*, 1998.

[2] *Low Power Training Course Material*. Synopsys, Inc., 1998.

[3] *Synopsys Design Compiler Reference Manual: v2001.08*. Synopsys, Inc., 2001.

[4] *Synopsys Power Compiler User Guide v2001.08*. Synopsys, Inc., 2001.

[5] TSMC 0.18 μm Process Standard Cell Library Databook. *Artisan Components*, 2001.

[6] P. Banerjee, M. Jones, and J. Sarjent. Parallel Simulated Annealing Algorithms for Cell Placement. In *IEEE Transaction on Parallel and Distributed Systems*, volume 1, pages 91–105, January 1990.

[7] M. S. Bright and T. Arslan. Multi-Objective Design Strategy for High-Level Synthesis DSP Systems for Low Power. In *ISCAS 99*, volume 1, pages 80–83, 1999.

[8] M. S. Bright and T. Arslan. A Genetic Algorithm for the High-Level Low Power Design of DSP Systems. In *Proc. IEE/IEEE Conf. on Genetic Algorithms in Engineering Systems*, pages pages 174–179, Sept. 1997.

[9] D. Brooks and M. Martonosi. Value-Based Clock Gating and Operation Packing: Dynamic Strategies for Improving Processor Power and Performance. In *ACM Transactions on Computer Systems*, volume 18, pages 89–126, May 2000.

[10] D. Bruni, A. Bogliolo, and L. Benini. Statistical Design Space Exploration for Application-Specific Unit Synthesis. In *Design Automation Conference*, 2001.

[11] A. P. Chandrakasan and R. Brodersen. Minimizing Power Consumption in CMOS Circuits. In *Proceedings of IEEE*, volume 83, pages 498–523, 1995.

[12] S. Devadas and A. R. Newton. Algorithms for allocation in datapath synthesis. In *IEEE Transaction on CAD of Integrated Circuit and Systems*, volume 8, pages 210–215, 1991.

[13] S. Gailhard, O. Sentieys, N. Julien, and E. Martin. Area/Time/Power Space Exploration in Module Selection for DSP High Level Synthesis. In *Int. Workshop, PATMOS'97*, pages 35–44, September 1997.

[14] D. A. Hodges and H. G. Jackson. *Analysis and Design of Digital Integrated Circuits*. -Hill, 1983.

[15] P. C. Kao, C. K. Hsieh, C. F. Su, and C. H. Wu. An RTL Design-Space Exploration Method for High-Level Applications. In *IEICE Trans. Fundamentals*, volume E84-A, pages 2648–2654, Novomber 2001.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated Annealing. In *Science*, volume 220, pages 671–680, 1983.

[17] P. E. Landman. *Low-Power Archhitectural Design Methodologies*. PhD thesis, University of California at Berkeley, Berkeley, CA, August 1994.

[18] A. Maheswari, W. Burleson, and R. Tessier. Trading off Reliability and Power-Consumption in Ultra-Low Power Systems. In *International Symposium on Quality Electronic Design*, May 18-21, 2001.

[19] R. Martin and J. Knight. Power-profiler: Optimizing ASICs power consumption at the behavioral level. In *Design Automation Conference*, 1995.

[20] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. In *Journal of Chem. Phys.*, volume 21No, pages 1087–1092, 1953.

[21] M. Munch, B. Wurth, R. Mehra, J. Sproch, and N. Wehn. Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths. In *IEEE Design Automation and Test in Europe*, pages 624–631, March 2000.

[22] C. P. Ravikumar. *Parallel Algorithms for VLSI Physical Design*. Ablex Publishing Corporation, 1996.

[23] P. Rezvani, A. H. Ajami, M. Pedram, and H. Savoj. LEOPARD: A Logical Effort-Based Fanout Optimizer for Area and Delay. In *ICCAD*, 1999.

[24] A. Sirinivasan, G. D. Huber, and D. P. LaPotin. Accurate Area and Delay Estimation from RTL Descriptions. In *IEEE Transactions on VLSI Systems*, volume 6, pages 168–172, March 1998.

[25] W. Wolf. *Modern VLSI Design: A systems Approach*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1994.

[26] Q. Wu, M. Pedram, and X. Wu. Clock-Gating and Its Application to Low Power Design of Sequential Circuits. In *IEEE Transactions on Circuit and Systems*, volume 47, pages 415–420, March 2000.

VITA

Fuat Karakaya was born on January 1st, 1970 in Aksaray, Turkey. He received his primary and secondary education in Ankara. He studied at Hacettepe University, in Ankara-Turkey, where he received his Bachelor of Science degree in Electrical Engineering in May 1993. He received his Master of Science degree in Electrical Engineering from Illinois Institute of Technology, Chicago-USA, in May 1998. He then began working toward his Doctor of Philosophy degree at University of Tennessee, Knoxville-USA. While pursuing his doctarate degree, he was a research assistant for the Industrial Plasma Laboratory and the Microelectronic Systems Research Laboratory. He received his Doctor of Philosophy degree in Electrical Engineering in May 2004.