To the Graduate Council:

I am submitting herewith a dissertation written by Mourad B. Takla entitled "A Quantitative Approach for Managing the Multi-Dimensional Design Space of VLSI." I have examined the final 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

Don Bouldin, Major Professor

We have read this dissertation
and recommend its acceptance:

Marshall Pace

Robert M McConnel

Daniel B. Koch

Kenneth E. Kirby, IE

Accepted for the Council:

C. W. Minkel

Associate Vice Chancellor
and Dean of The Graduate School

A QUANTITATIVE APPROACH FOR MANAGING THE MULTI-DIMENSIONAL DESIGN SPACE OF VLSI

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Mourad B. Takla

May, 1992

To the memory of my parents

Bedour B. Fahmy & Bushra S. Takla

ACKNOWLEDGEMENTS

A number of people have had a great influence on this work and its completion. The author would like to thank Drs. D. Bouldin, D. Koch, M. Pace, R. McConnel, and K. Kirby who serverd in his advisory committee and contributed with their time and effort.

The author would like to express his deepest appreciation to his major professor, Dr. D. Bouldin, for the help, guidance, encouragment, support, patience, and friendship that he provided during the course of this work. Without his support this work could not have been completed. Also, the author would like to thank Dr. J. Googe and Dr. D. Koch for the various ways that they helped this research and for their financial support. This research was supported in part by a grant from the Bell South Foundation.

Special thanks to many of the peers and friends, particularly M. Abdel-Ghafour, Y. Azmy, D. Bargiotas, P. Dehkordi, P. Mukund, H. Saraf, M. Sidani, T. Sawan, M. Sharobeam, and staff of the Center for International Education for their encouragement and suggestions over the course of this research effort. Lastly, and most importantly, he would like to thank his wife Mary and his family for their help, support, encouragement and understanding when all were in great demand; projects such as this would otherwise be impossible.

ABSTRACT

The design of integrated circuits has evolved in the last few years to become a complex and diverse task. The designer deals with a problem that has multiple dimensions. Some of these dimensions are: area, time delay, power consumption, and design time. The designer's task is to find the optimum design within this multi–dimensional design space such that it can be achieved using the available resources.

Given a certain set of specifications, the designer is to select any of the available algorithms to achieve the required goals. Starting from a behavioral description of the problem, various software tools can synthesize a chip that performs a specific function. However, a more global level is needed to deal with the available algorithms on a higher level while still preserving the multi–dimensional perspective of the problem. This level is used to compare the possibility of implementing the various algorithms without the need of generating the physical layout.

This dissertation describes a methodology that is capable of evaluating algorithms on the highest level for VLSI design. A cost function that utilizes techniques from the management science area is proposed and is used to relate the various dimensions of the design space.

Several examples are introduced to show the strength of this methodology. A comparison between the direct implementation of the fast Fourier transform

and a novel implementation of the distributed arithmetic Fourier transform is performed using the proposed cost function. Other examples that deal with high level design issues and the selection of a design from the highest level possible are also presented.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE **PAGE**

LIST OF TABLES

CHAPTER 1

Introduction

The design of integrated circuits (ICs) is a complex and diverse task. The designer is challenged to produce the optimum design that best solves the problem in hand. The rapid advancement in IC technology presents designers with bigger and more complex problems as more and more elements are included on a single chip and faster clock rates can be achieved. Digital Signal Processing (DSP), image processing, and digital systems are some of the areas that have benefited from this rapid advancement.

In the first generation of ICs, designers drew each layer for all the elements in the design. This task typically took several man–months [1,2,3]. Now the advancement in lithography, manufacturing, and other aspects related to the fabrication of ICs allows thousands of transistors to be integrated on the same area that used to include only a limited number of transistors; hence, the time required to perform the design at this low level is often prohibitive.

The complexity of the problem does not stop at the size of the design. Several factors affect the design and have to be considered and optimized simultaneously. Some of these factors are: area, time delay, power consumption, temperature, input/output overhead, and design time. These factors can be viewed as the

various dimensions of a multi-dimensional design space. The design problem can be restated as selecting the design which is represented by a single point in the design space that minimizes a given cost function.

Most reported work lists the total area used, operating frequency, number of input–output pins, or estimated power dissipation as the criteria to differentiate between various designs [4,5]. Some authors attempt to develop metrics that depict some aspects of the design, such as throughput [1,6,7]. Others deal only with one aspect of the problem, such as area, throughput, or information content [7], and fail to give a more general definition that may guide designers in comparing various ideas that have not been implemented completely. Viewing the design problem as a multi-dimensional design problem allows the definition of a cost function that will give a more global and consistent definition of the term *best.*

The design process takes several stages (Figure 1.1). The basic goal is to transform a given algorithm into hardware. This can be done on a single chip or a complete system based on the complexity of the algorithm and the amount of the data handled by the system. Silicon compilers transfer a behavioral description of a system into hardware [8,9,10]. One reported work transferred `FORTRAN` programs into silicon [11]. Other work concentrated on optimizing several elements and transferred all the functions in the algorithm to be performed by these elements [12,13].

Often, the designer is given a certain set of specifications and is asked to select

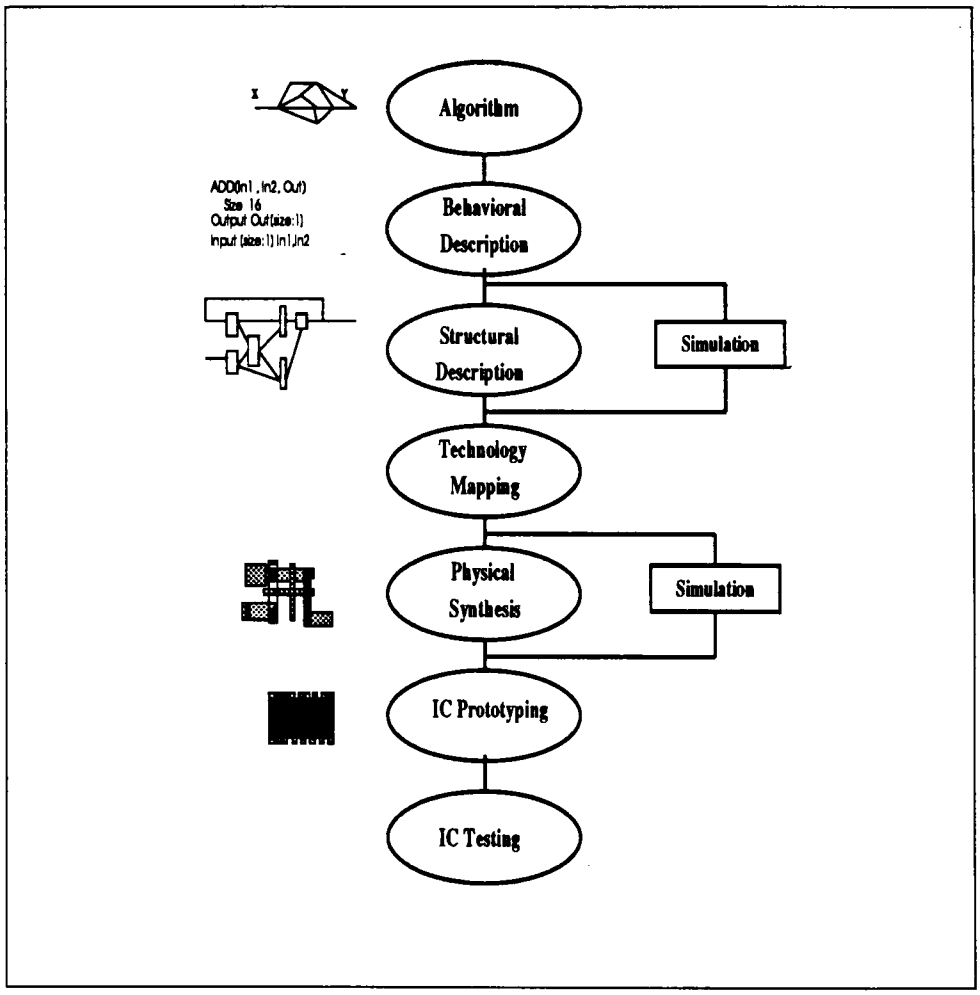

Figure 1.1: The various design stages as depicted in the literature.

the algorithm that utilizes the available resources within the given constraints. However, the lower levels of the design are not needed at such a stage. Essentially no work has been reported in the literature to guide the designer in deciding how to map the given algorithm into hardware, how to develop the appropriate behavioral description that implements the given algorithm in specific logic or register transfer language (RTL), how to put it in the proper signal flow graph representation, or whether a single chip or several chips will perform the required task better and still give an economical design.

A more global approach is needed to guide the designer in managing the available resources and developing the optimum system, with respect to the various dimensions, within the given constraints. The goal of this work is to develop a design methodology that will enable the designer to approach the global design problem as a multi-dimensional problem and allows him to select the optimum design at the highest level possible. Meanwhile, the definition of optimum will be flexible enough to accommodate the designer's needs and the different design conditions.

As the size of the problem tackled by the designer increases, two additional problems arise. The first problem is to define a design methodology that guides the design process, provides alternative solutions, and brings out critical portions of the design that need to be optimized. The second problem is to define the terms *optimum, efficient,* or *best* design.

The next chapter introduces the background of the various work reported in

the areas of DSP system design and evaluation metrics for VLSI designs. Following that is a chapter that describes the design methodology. Next, a chapter with several examples to demonstrate the strength of the proposed methodology and the various ways of using it to make important decisions during the design task. The last chapter includes the conclusions and directions for future work.

CHAPTER 2

Background and Overview

In this chapter, a quick overview of the design problem will be introduced. Then a review of the various work in the area of the VLSI synthesis is given. The various techniques and concepts adopted from the management science and computer science areas will then be introduced.

2.1 Overview of the Problem

Digital Signal Processing is one of the many fields that have benefited from the advancement in the various areas of VLSI [8,14]. DSP algorithms usually perform a fixed function. They are independent of the type of data entered. The algorithms usually operate in either a recursive fashion or in a vector format. These properties make the use of an application–specific integrated circuit (ASIC) architecture more attractive. The overall cost of an ASIC not only includes the silicon cost, but more importantly the cost of the design time.

Including tens of thousands of transistors in a single design is not a simple task. Designers prefer to deal with the highest level of abstraction possible. Dealing with an adder or a multiplier as a unit is much more convenient than dealing with hundreds of gates that perform these functions. A higher level of

abstraction that treats higher level operations, such as the Fast Fourier Transform (FFT) or windowing, as a single unit is preferred over dealing with the various operations that perform such a task [9,15]. Tools that help the designers of such systems have been developed [8,9,16]. Most of these systems deal with mapping a given algorithm into a single chip.

The synthesis problem treated in the literature can be divided into a limited number of steps:

- To map the algorithm into separate operations that will be performed in sequence or in parallel.
- To allocate resources and schedule events to these resources (function units).
- To design and implement the various function units and connections.
- To verify and simulate the design obtained.

The various issues that need to be considered to develop an optimum design make the design space a multi–dimensional one. Seeking the best design will require the design to be optimized along the various axes of this space. Designers, usually tackle the design problem in a two–dimensional approach in which they only investigate the relationship between two aspects of the design such as area and time delay or time delay and and power consumption. Some of the published work failed to treat the problem as a multi–dimensional design problem and usually treat the area–time or time–power planes only [17,18,19,20,21]. The

reason designers have taken this approach is the need to optimize the design in a given dimension. An example would be a constraint to design the fastest chip that performs a certain task or to obtain the design with the least area. However, optimizing the design relative to one dimension of the space and not observing the absolute coordinates of the design within the design space may lead to inferior designs. Ideally, a design should be obtained that is *close to optimum* along all dimensions and optimum along a single dimension that is of greatest perceived importance. Figure 2.1 depicts some examples for the relationship between the various dimensions. As can be seen, the design problem can be viewed in one dimension, two dimensions, or multiple dimensions.

In this work, a cost function that relates the various dimensions of the design space into a single figure is introduced. The goal of the designer is to come up with the design that minimizes this cost.

2.2 Background and Literature Survey

Before describing the various design methodologies and the various tools used for VLSI design, an overview is given for the different steps taken in mapping an algorithm into a physical layout.

Most of the CAD tools describe the algorithms that will be synthesized internally in one of several ways: as a signal flow graph (SFG), as a parsed tree, or as a combination of both [8]. The work reported by several authors

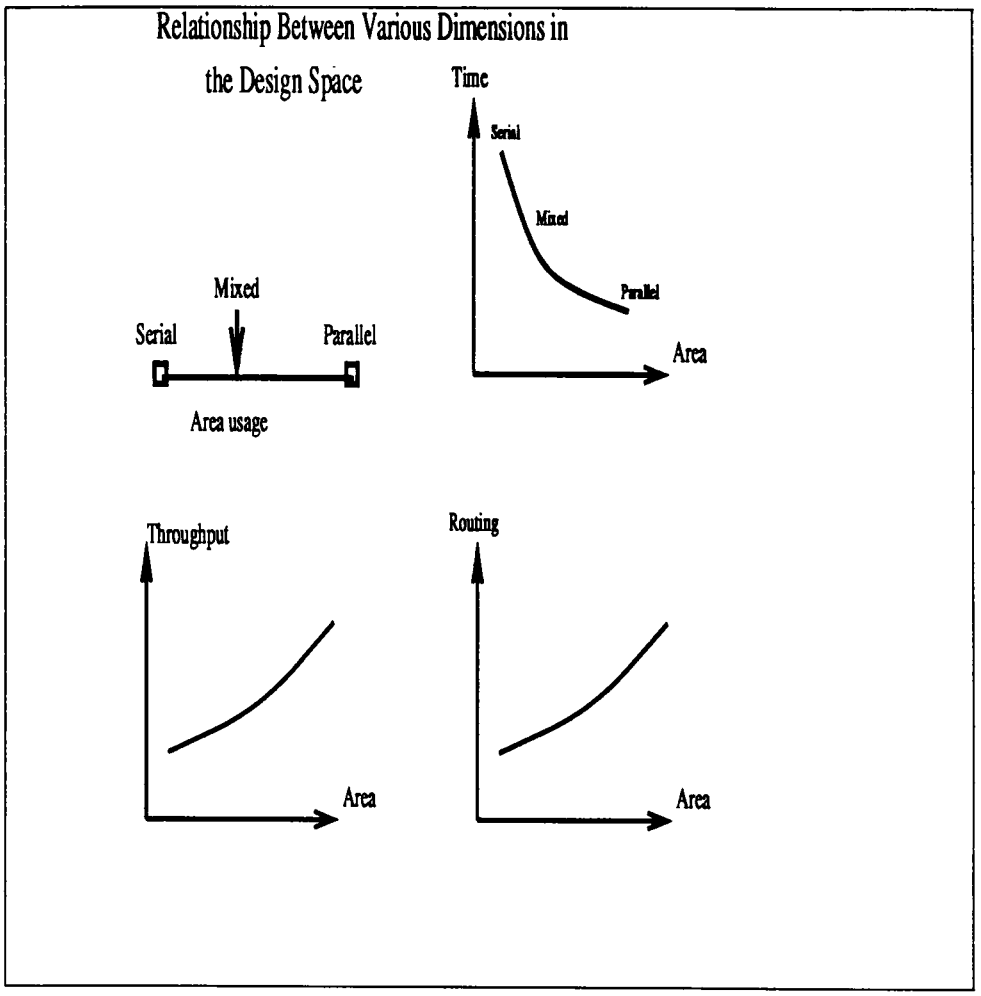

Figure 2.1: The relationship between various dimensions in the design space.

[11,15,17,18,22,23,24,25,26,27] used an SFG to represent the algorithm. In this approach, operations such as addition, subtraction and multiplication, are represented as nodes. The directed branches represent the direction of data flow. The nodes are arranged in the same order as the data that will go through them. The work reported in [28,29,30] uses graphs for grouping various operations with function units (FUs) and scheduling of various tasks on the available resources.

Scheduling and resource allocation are two mutually dependent tasks. The allocation problem is to assign the various operations to a specific number of FUs and to minimize the overall area. Based on the available resources, scheduling of events has to be performed such that global timing constraints are met. On the other hand, based on the nature of the FUs, the result of the scheduling will be affected. In the case that the global time constraints are not met, different function units need to be added to the design or some FUs need to be redesigned and the scheduling reiterated. The solution to this problem depends on the specific software implementation, as will be shown later.

Function units are then implemented using available libraries or standard cells [8,16]. Special tools are used to layout and connect the various parts. **Lager** and LASSIE are such tools [24,28]. Similarly, design verification is a very important problem. However, both the general problems of design implementation and design verification will not be discussed here as they are beyond the scope of this work.

Next, the various work in the area of IC synthesis will be described. The work

published can be placed into five categories. The first category are those papers that discuss the synthesis problem from a tutorial point of view, or an overview of the global design problem or portions of it [8,9,10,16,31,32]. The second category consists of papers that deal with a specific synthesis system or a synthesis approach from a global point of view [11,15,17,18,33,34]. The next group deals with portions of synthesis systems, such as resource allocation, scheduling, or area estimation [19,20,23,24,25,26,27,28,29,30,35,36,37]. The fourth group of papers treats the synthesis problem from a specific point of view, such as mapping the problem into a specific architecture, rewriting the algorithms in a way more suitable for VLSI design, or simply the various ways of implementing specific elements [12,13,14,38,39,40,41,42]. The last group deals with system evaluation and metrics that compare the various performance issues [3,6,21,43].

2.2.1 Tutorials and Global Considerations in Synthesis

McFarland *et al.* [8,44] published a tutorial paper that deals with high–level synthesis. The author define high–level synthesis as the task that takes a set of specifications and converts them into a structure that meets the specified goals and constraints. The authors justify the reasons behind studying high–level synthesis as:

- Shorter design cycle: by automating most of the design procedure.
- Fewer errors: the design will be verified and tested before fabrication.

- Capability of exploring various designs and search the design space in a reasonable time.
- Self–documentation
- Availability of IC technology to more people as more design experience is moved to the synthesis system.

The authors describe the various operations involved in synthesis, such as data representation, scheduling, and allocation. These papers provide a good introduction to the area of high–level synthesis.

Gajski and Thomas [45] describe the various stages of silicon compilation. The authors define three different domains for representing the various designs: *a behavioral domain, a structural domain*, and *a physical domain.* Within each of these domains, several levels of description are defined. These levels are system, algorithmic, microarchitectural, logic, and circuit levels. To demonstrate this categorization consider the algorithmic level. In the behavioral domain this is represented by manipulation of data structures. In the structural domain, this is represented by hardware modules and data structures and in the physical domain this is represented by clusters of function units. This type of classification is too detailed. The difference between the domains and levels is hard to establish. In the other published work [44,46], a mix of the domains and the levels are usually used. They define the function of synthesis tools as mapping between the different domains or different levels. The authors also compare fixed layout

architecture to flexible architecture.

Keutzer [9] defines ASICs as a marketing term that is given to a segment of the market that lies between software programmable generic components, such as microprocessors, and gate–level programmable devices. The author reports a summary of various types of ASIC designs implemented in standard cells by AT&T. They had the following common characteristics:

1. Control dominated
2. Little arithmetic
3. Speed/Area were limited (10,000 logic transistors and operating frequency was around 10MHz)
4. Regular structures such as RAMs, ROM's, and comparators were used
5. Analog interfaces

The author defines three different approaches to ASIC design: architectural synthesis, logic synthesis, and module generation. Architectural synthesis is defined as: the translation process of the desired behavior for a circuit, given in a program–like form, into an architecture described as a register transfer level (RTL) structure. This category includes tasks such as converting filter equations into hardware. Logic synthesis is the transfer of a register–transfer level description into combinational logic and registers. The module generation approach includes silicon compilers where the designer has minimum interaction with the system generation.

In this paper, Keutzer also identified three major challenges for ASIC design:

1. The lack of education and people trained in the area of ASIC design.

2. The lack of synthesis and verification procedures of asynchronous and analog partitions of the circuits.

3. Finally, verification of the design after the design task is completed.

Sangiovanni–Vincentelli [10] reviewed the synthesis problem of Programmable Logic Arrays (PLAs) and Random Logic. For PLA synthesis the author emphasized three major points:

1. Logic optimization: to reduce the number of terms representing the function.

2. Topologicalcoptimization: to elimtnate unused space inside the core of the PLA.

3. Layout and Circuit optimization: To place and size devices and drives optimally.

The author also described briefly the various tools used in the above tasks.

Urbanski *et al.* [32] provide a practical insight into the design methodology. They differentiate between two different targeted designs. The first category is high volume chip design, in which every small improvement in performance is required. The second category targets special purpose chips, where the design

should be highly automated to reduce the design cost. The authors partition the design of VLSI systems into the following phases:

1. System specification
2. System design
3. Chip design (functional design and geometrical design)
4. Block design (functional design and geometrical design)
5. Routing
6. Layout design

The authors describe each of these steps briefly without specifying an algorithm for performing it.

Cai and Hegge [31] compared various methods for *floorplanning*. Floorplanning is the task of arranging the various FUs within the chip's physical dimensions. The authors define the common objectives as: chip size, chip aspect ratio, and total net length. The min–cut algorithm, the force–directed algorithm, simulated annealing, and sequence heuristic algorithm are considered with some practical examples. The authors observed that in most cases the min–cut algorithm performed better than the force–directed algorithm. It also performed better than the other two algorithms if the CPU time is limited. Starting from a *good* initial floorplan, simulated annealing performed better than the other programs. However, this good initial floorplan was not given in general terms.

Borriello and Detjens [16] discussed the development of benchmark test cases at a workshop to compare various synthesis tools. These benchmarks include

1. Serial–line controller (Intel 8251)
2. A small microprocessor (MC6502)
3. An elliptical wave filter
4. A large microprocessor (MC68000)

The conclusion of this workshop regarding benchmarks and other issues related to high level synthesis were:

- Most designers are not pushing at the edges of the speed/power curve.
- There have to be two levels of benchmark examples: compulsory and free style. The free style benchmark is to show special features of the synthesis algorithms that perform better in special domains.
- It is very difficult to standardize the hardware description language or control data flow graph (CDFG).
- Test vectors have to be supplied along with the design examples to verify the designs obtained.
- Real–life design constraints, such as limited area, time delay, or power constraints, have to be included.

- In order to compare two systems, curves, such as area versus speed, need to be compared and not just single points.

2.2.2 Complete Synthesis Systems

Jain *et al.* [17] describe the ADAM (Advanced Design AutoMation) synthesis system that manipulates several tools together to perform synthesis tasks, including area and time prediction. Three inputs are required by the ADAM system: a data flow graph representing the behavioral specification, a design library, and a set of design constraints. The first element in the system is SIMOS that selects modules that maximize performance or minimize area. This theoretical model uses an area–time predictor in order to choose a pipelined or non–pipelined design. If a non–pipelined design is targeted the MAHA program for scheduling will be used; otherwise the SEHWA program performs functional pipelining. Various predictors are used in this synthesis system in order to select the best modules that implement a specific function. Two specific predictors are used for pipelined and non pipelined design. PLEST is used to predict the wiring area, and PASTA is used to predict the area needed by the PLA that might be used for the control part. A simple filter example is given and the various results of the different programs are described.

DeMan *et al.* [47,48] describe the CATHEDRAL project. The authors approach the design problem for DSP systems from the perspective that general purpose hardware compilers are not efficient and using such an approach will produce

inferior designs. The authors divided DSP systems into four groups and designed a special compiler for each group. These groups are:

- A hardwired, bit–serial architecture: Each arithmetic operation of the algorithm is implemented as a separate one–bit operator in the architecture. Signals are processed bit by bit, the least significant bit first.

- Microcoded Processors: These are oriented towards low to medium rate algorithms, combining heavy decision making and computation intensive array or matrix manipulations.

- Bit–sliced multiplexed data–paths: The data path is matched to the signal flow in the algorithm. This usually leads to a cluster of application–specific circuits.

- Regular array architectures: This type of architecture has a regular network of processing elements, mostly localized communication, and distributed storage.

Each of the above architectures has a dedicated compiler. Realizing that a human designer acquires better insight than a CAD program into the complexity and computational bottle-necks of a DSP algorithm, human interaction is provided in several parts of the compiler's task. The authors provide no rigorous method in selecting one class over the other. No pre–processing is available for the designer to take a decision in implementing a certain algorithm one way or the other.

Shung *et al.* [49] approach the problem of designing DSP systems from an approach opposite to that of [47]. The authors argue that diverse algorithms can be implemented with a single set of hardware modules and the re–use of these hardware modules greatly reduces the design time. The authors describe **LAGER**, an integrated CAD system for ASIC design. This system is divided into two parts. A behavioral mapper that maps the behavior onto a parameterized structure to produce micro code and parameter values. Then a silicon assembler translates the filled–out structural description into a physical layout. The authors also describe the various sub–parts and give two design examples.

Ashby *et al.* [15] describe an integrated environment that is dedicated for signal processing system design. The ADS (ALPS Design System) accepts the input algorithms through a graphic entry capture. The target methodology for ADS is ALPS (Alternate Low–level Primitive Structures), which is a self–synchronizing, self–scheduling, distributed control architecture. A signal flow graph is used to describe data manipulation and operations performed on them. Operations such as windows, FFTs and FIR filters, are used as blocks to represent the algorithm. One important criterion used for resource allocation and scheduling is power conservation. Operations that are encountered more frequently or high duty cycle primitives should be implemented *efficiently* in terms of both speed and power dissipation. Two levels of simulation are supported by this system: coarse level and fine level simulations.

Tanaka *et al.* [11] report a FORTRAN–to–silicon compiler. Starting from a

FORTRAN program it generates an RTL description. HARP consists of three major parts: a data path synthesizer, a sequence controller synthesizer, and an RTL translator. The first synthesizer creates a data flow graph (DFG) and a data path synthesis. A finite word length synthesizer (FWLS) is used interactively to determine the various word lengths used. No special guidance for word length selection was given in this paper. Operations in the data flow graph are assigned to function units. A special mathematical function that describes the mutual correlation between various FUs is defined. Evaluating this function for all possible combinations and then choosing the minimum value will lead to the appropriate function units that would be merged. Allocation of storage elements is done after evaluating the variable's life–time tables during the various execution steps. The authors identify the following problems as areas that need to be worked on in their system:

- A clock synthesis scheme should be realized. A multiple phase clock is not yet realized.
- The ability to implement a different strategy for pipelined designs is not yet implemented.
- HARP currently is unable to incorporate ROM/RAM allocation.

Moldovan and Varma [33] describe a methodology dedicated for the design of algorithmically–specialized VLSI devices. The algorithm is divided into two major steps. The first step maps the algorithm into a global model that de-

scribes the overall structure and operation of a VLSI array. The second is to design the processing cells of the array based on the information contained in the global model. The global model is characterized by a network geometry, a set of functions performed by the processing units, and the network timing. The processing cells that they designed have the following characteristics.

1. A set of registers to hold the input data.
2. A set of ALUs.
3. A set of register files used as delay elements to synchronize the output data.
4. A set of scratchpad registers for temporary results.
5. Interconnections between registers and ALUs for proper routing of the data.
6. Control logic to synchronize operations within the cell and with the rest of the design.

Haroun and Elmasry [18] report on SPAID, a synthesis tool dedicated for DSP algorithms. The DSP algorithm is captured in the form of a generalized signal flow graph (GSFG) through a hierarchical graphical interface, and then finally mapped to a multibus architecture. The user indicates the throughput and latency constraints by attributing them to the terminal nodes of the graph. Three steps are followed to explore the design space. Different predicates are

used to select the FUs and the clock period. Structural transformation is applied globally and locally on the graph to optimize/satisfy the throughput and/or latency. Different concurrent execution of operations are explored to achieve minimum cost architecture that satisfies the throughput.

The **System Architect's Workbench** is a synthesis system reported by Thomas *et al.* [34]. Two different design approaches are followed in this system. The first synthesizes a style–specified (application–specific) chip, while the second approach deals with the general synthesis problem. In designing a style–specific circuit, previous knowledge available about this specific application should be included in the synthesis program to obtain a more efficient design.

Microprocessor design is different from the design of a chip that implements a DSP algorithm. The knowledge needed to design the first is different from that of the latter. The **SUGAR** program follows this style approach, where previous knowledge about the application is included in the program to guide the design. **SUGAR** is dedicated to the design of microprocessors. The second approach is oriented toward solving the more general problem of synthesis that can be used with a range of design styles. The specific information about the domain can be included in a database that helps the synthesis tools while performing their functions and not embedded in the tool itself as is the case with **SUGAR**. This knowledge is used when the behavioral transformation is performed. The **CSTEP** and **APARTY/EMUCS/BUSSER** programs are used in this approach. An example of

an elliptic filter is used to demonstrate the second approach and the design of the MC6502 demonstrated the use of SUGAR.

The MICON system by Birmingham *et al.* [36] is a system dedicated to the synthesis of small computer systems from high–level specifications. This approach is different from the above synthesis systems as it uses off–the–shelf elements, such as the Intel 80386 and Motorola 6809, 68008, and 6810, as part of the design. Also, the targeted goal in this case is a complete microcomputer and not a single chip. Artificial intelligence is being used to handle the synthesis task and the knowledge base available for the system. M1 is a knowledge–based synthesis tool and CGEN is an automated knowledge acquisition tool. Both represent the nucleus of the MICON system.

Kowalski [50] describes the experience gained while developing the Design Automation Assistant (DAA). The goal of this approach is *"to aid the designer by producing data paths and control sequences that implement algorithmic system description within supplied constraints"*. A knowledge–based expert system is developed using the experience of human designers. The author lists speed, area, power, schedule, cost, drive capabilities, and bit–width as constraints mentioned by the designers. Various tradeoffs between design rules, circuits used, and functionality had to be made in order that a working design be achieved. He compared the output design of the DAA system with those produced by human designers in order to judge the quality of the system produced.

2.2.3 Scheduling, Allocation and Area Estimation

In this subsection, a brief description of some of the work done in the areas of event scheduling, hardware allocation and area estimation is presented. A comprehensive literature review of these topics is not intended, rather an overview that depicts various methods to solve this problem is presented to provide a global understanding of the various problems involved in the design.

Paulin and Knight [29,30] introduced a new cheduling approach that is based on combining force–directed scheduling (FDS), and list scheduling (LS). This approach is implemented in a system called HAL. In the LS technique, a hardware constraint is specified and the algorithm attempts to minimize the total execution time by using a local priority function. In the FDS technique, a global time constraint is specified and the algorithm tries to minimize the resources required to meet the global time constraint. The proposed scheduling technique, forced–directed list scheduling (FDLS), attempts to solve the dual problem: the determination of a schedule with a near minimal number of control steps (c–steps) given fixed hardware constraints. The first step of this algorithm is to determine the time frames of each operation by evaluating them as soon as possible (ASAP), and as late as possible (ALAP). Following that, a distribution graph (DG) is created. The DG indicates the concurrency of similar operations by summing the probabilities of each type of operation for each control step of the control DFG. Forces are defined in a way that reflect the effect of moving an operation from one c–step to the other, such that the force associated with

an attempted deferral of a critical operation is equal to infinity. The force is then used as a priority function. Whenever a hardware constraint is exceeded in the course of regular scheduling, force calculations are used to select the best operation to be deferred whenever needed.

Devadas and Newton [23] used simulated annealing for hardware allocation. They represent the allocation problem as being a two–dimensional placement problem of microinstructions in space and time. These dimensions represent the hardware used by the microinstruction and the execution time of the microinstruction. The problem then becomes a minimization of an arbitrary function that relates both execution time and overall hardware cost. The resource cost represents the layout area required to implement the different modules. The basic tradeoff in hardware allocation is between serial implementation (minimum area and maximum execution time) and complete parallel implementation (maximum area and minimum execution time). The cost function used in this implementation relates the areas taken by the various elements used, such as registers, ALUs and interconnect, to the number of units used. Simulated annealing attempts to reach a global minimum of a given function [51,52]. This is done by introducing a perturbation to the system. If the difference in energy between the two states is less than a random number, the change is rejected, else the change is accepted and the algorithm continues. The selection of the cost function and the temperature gradient affects the performance greatly.

Parker *et al.* [25] reported on the `MAHA` program for data path allocation. The

Modified Automatic Hardware Allocator (MAHA) program is a part of the ADAM synthesis system [17]. This algorithm is based on the linear hardware assignment to critical path nodes, followed by a cost–based assignment to the other nodes off the critical path using the concept of freedom. This program either minimizes the cost subject to a time constraint, or maximizes speed subject to a cost constraint. Nodes on the critical path are assigned first to the function units. These events are sequential since they are on the same path. Multiple operations can share resources as long as they are mutually exclusive. The path is divided into n time steps of equal duration. The nodes that are not located on the critical path can be assigned to a time step based on the notion of freedom. The freedom of a node is defined as the difference between the time when the input values are needed by a given operator and the time when the result of the operation is required less the propagation delay. Resources are added whenever it is necessary. If at some point MAHA runs out of resources, the critical path is partitioned into smaller steps.

Lagnese and Thomas [28] describe the Architectural PARTYtioner (APARTY) that supports synthesis in the System Architect's Workbench [34]. Architectural partitioning provides high–level structural information about the design that can be used in the process of translating a behavioral description into a structural design. Again, the behavioral description of the algorithms is presented by a data flow graph. In order to decrease the total area and optimize the performance, some functions are grouped together into one function unit. In

order to perform this grouping, a multi–stage clustering algorithm is used. This algorithm performs clustering for the various operations based on some similarity criterion. Clustering can stop at any required level, or more clustering can be performed based on a different criterion, or the same one used in the previous stage. The authors define the similarity criterion as a probabilistic function. This function specifies the probability that one operation will be activated given that the one before it has been activated. Also, a measure of the common values between two operations is considered in proportion to the total number of values flowing from and to the specific operations considered. By grouping the appropriate operations together, the interconnection area, and hence the overall area, can be reduced greatly.

Potkonjak and Rabaey [24] deal with the problem of scheduling hierarchical flow graphs. When representing an algorithm by a flow graph, some nodes could be representing multiple operations or subroutine–like functions. If the flow graph is flattened, its size becomes too large to be handled conveniently. The proposed scheduling algorithm uses a scheduling scheme for flat flow graphs at its core. For each subtask or subroutine, both extreme costs are found, the cheapest and maximally fast implementation. This is done by using the non–hierarchical scheduling algorithm. A global hardware graph is generated using the result of both implementations. Execution time of the total program is evaluated based on the data gathered from the previous step. If the global time constraint is not exceeded, pruning in the global hardware graph is performed

to find a more suitable realization. Only these blocks that are affected will be rescheduled until the criteria sought for the whole design are satisfied.

Konstantinides *et al.* [37] describe a scheduling and task allocation algorithm that is based on the branch and bound algorithm [53]. They differentiate between two different types of algorithms. One type requires a complete set of data before producing an output, *e.g.*: FFT. This is called a block–type task. The second type requires only a portion of the data before it starts producing the output, *e.g.*: filter operations. This type of algorithm is called a stream–type task. Allocation and scheduling are performed for both processors and input–output units.

Another aspect of the synthesis problem is to estimate the chip area without actually performing the physical layout of the chip. Kurdahi and Parker [26] describe a system that estimates the area of standard cell layouts. This is done as a part of the more global area estimation package used by the **ADAM** system [17]. PLEST (PLotting ESTimator) estimates the total area needed to layout a group of standard cells comprising a chip design. Given the number of rows required in the design, estimates of the track size and the total wiring space are determined. It was observed that the routing area grows by about $O(N^{1.5})$, where N is the total cell size, and is bounded by the limits $O(N)$ for optimal placement and $O(N^2)$ for random placement. The algorithm starts by assuming that all cells are laid down in a single row followed by folding into multiple rows. Experimental results validated the estimates to be within a 10% error range.

Another work that follows the same line of thought is that done by Jain *et al.* [20]. In this paper, a theoretical model for predicting the area–time tradeoff curve for non–pipelined data paths is presented. This model does not synthesize the circuit before it produces the A–T curve, rather it takes a data flow graph as its input and estimates the clock cycles that produces the best possible design. Two equations are developed to describe the theoretical lower bound:

$$AT = \frac{kC}{t}$$

$$AT = k \times max(d_i)$$

where A is the total area, T is the time delay, k is a constant, C is the critical path delay, t is the number of time steps the DFG is partitioned into, and d_i is the delay for each module type i. The theoretical bound is checked against the designs generated by `MAHA` [25], which is also a part of the `ADAM` system [17].

2.2.4 Specialized Synthesis Approaches

In this section various works that deal with the synthesis problem from a point of view different from the one presented above are examined.

Lee *et al.* [41] developed a system called `GABRIEL`, a design environment for programmable DSPs. The target of this work is to use DSP processors already available commercially, and then build the necessary assembly code or microcode that executes the intended algorithm. The Motorola *DSP56001, DSP56000*, and AT&T *DSP32* processors are in the core of the design. `GABRIEL` is the second

generation after BLOSOM from the same group. Function blocks are called *stars*, a group of *stars* is called a *galaxy*. *Galaxies* can be treated as *stars*. Both graphical and textual descriptions are used to represent the given algorithm.

Ercegovac and Lang [38] tackle the synthesis problem from a different perspective. The authors advocate the development of new algorithms that best utilize the properties of a VLSI system. They demonstrate such an idea by using "on–line arithmetic" to compute the term $(A^2 + B^2)^{1/2}$. In the traditional implementation of arithmetic operations, least significant digit is treated first (LSDF). However, in this approach, the most significant digit is treated first (MSDF). This allows the on–line arithmetic algorithms to operate in a digit–serial manner beginning with the MSD and also allows overlapping of operations instead of waiting for the complete result to be generated before advancing to the next stage. The authors also presented an example of an IIR (Infinite Impulse Response) filter.

Iwasaki *et al.* [42] describe a hard disk controller and the design effort involved in this system. The authors emphasized several facts related to the design process including the partitioning of large systems into several modules and that a hierarchical design approach must be followed. To achieve such hierarchy, the module interface must be simplified to allow parallel implementation by different designers. Most modules should be reconfigurable to allow more flexibility, so ROMs and PLAs are used more. Various clocking problems are also discussed. This paper does not provide a clear algorithm for the design problem similar

to that presented in [32]. However, since the authors are an industrial–based group, a practical approach to the design problem is given.

Chu *et al.* [39] deal with another aspect of the synthesis problem, which is building self–timed modules. In this paper, the authors describe a methodology to design self–timed circuits. The system is divided into two subgroups: data path and system controller. A design example of a 2×2 packet router is also given.

Tsuda and Mochida [12] realize that multipliers are used intensively in DSP systems, so they describe a pipeline approach for a 16–bit multiplier with fixed point arithmetic.

Smith and Morgan [14] have adopted the gated, latched full–adder as the basic computational element in their pipeline approach. Most of the paper is dedicated to describing the format of the data, but not manipulating the data itself. The authors give a good summary for properties of DSP algorithms.

Wu and Cappelo [13] map IIR filters into groups of second–order sections. The authors developed the part of a second–order section efficiently in a bit–level systolic array architecture. This is another example of systems that develop one element and make it very efficient. Then the algorithm is partitioned in a way that uses this element.

Kuroda *et al.* [40] investigated the effect of feature size scaling and the use of different technologies. The authors investigated device performance and the limiting factors on size reduction. Also they incorporated the use of various

technologies such as CMOS and Bi–CMOS to reach comparable performance.

2.2.5 Performance Measures

This group of papers deals with system evaluation. Given two implementations of the same algorithm, what should be the criteria used to prefer one design over the other? Several authors [3,4,5,54,55] reported on the result of their work and used area, delay time, power consumed, throughput, latency, and the total number of gates/elements used in the design as being the basic criteria that describe their work. However, these values are not sufficient to describe and compare complicated systems, since the amount of control circuitry versus the amount of computational elements, and the efficiency of using the elements in the design are not reflected in the above figures. In large computers MIPS (millions of instructions per second) and MFLOPS (millions of floating point operations per second) are used as figures of merit. These figures are not true indicators of the system performance as they may represent peak performance values with a specific data set and not the true performance measure [6,7,43].

Yu *et al.* [7] use the principle of information theoretic to describe their system. The authors define DSP tasks as being information preserving (example: FFT), or information reducing (example: FIR filters). The system then becomes a communication network between the various subtasks. System performance is evaluated based on the processing gains (dB), algorithm efficiency (gate–Hz/dB), technology used (gate–Hz/watt), and architecture efficiency (% duty cycle). The

authors state that total power dissipation relates the design space of physical size, weight, cost, and implementation technology for that system.

Ward *et al.* [6] define several cost functions and use them to select the best FFT and DFT implementations. The system's cost function is defined as the product of the network cost and the technology price, where a network means both the algorithm and the architecture used. Inefficiency is measured by the proportion of time during which gates do not potentially change states and memory is not being accessed. The final cost includes portions that reflect the size of the memory (RAM and ROM) and their access speed, and the total number of gates used in the architecture, all normalized by the throughput. Also the technology factor is included in the total cost as watts/gate–Hz, or silicon area/gate–Hz.

2.2.6 PERT

Project Evaluation and Review Technique (PERT) is a technique from the management science area that uses a signal flow graph represent the project (system) in hand (Figure 2.2). Nodes represent events that take place after all the branches leading to them occur. Branches represent the direction where the data flow. The addition of dummy branches is allowed to preserve precedence of events. A forward pass consists of traversing the graph from the source node to the sink (terminal) node [56]. During the forward pass, the source node (initial event) is set to the start time 0 (Figure 2.3). All activities are assumed to start

as soon as possible (ASAP), *i.e.* as soon as all their predecessor activities are completed.

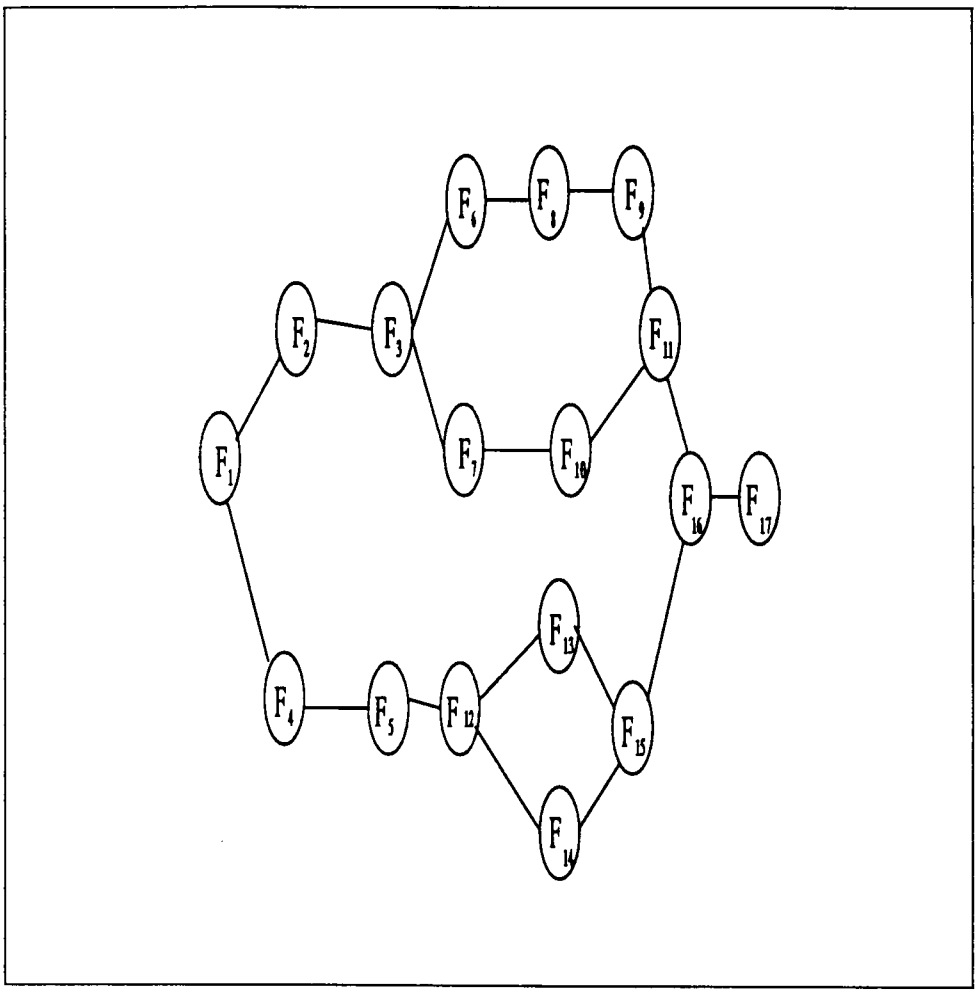

Figure 2.2: A SFG that is used to depict the various concepts of PERT.

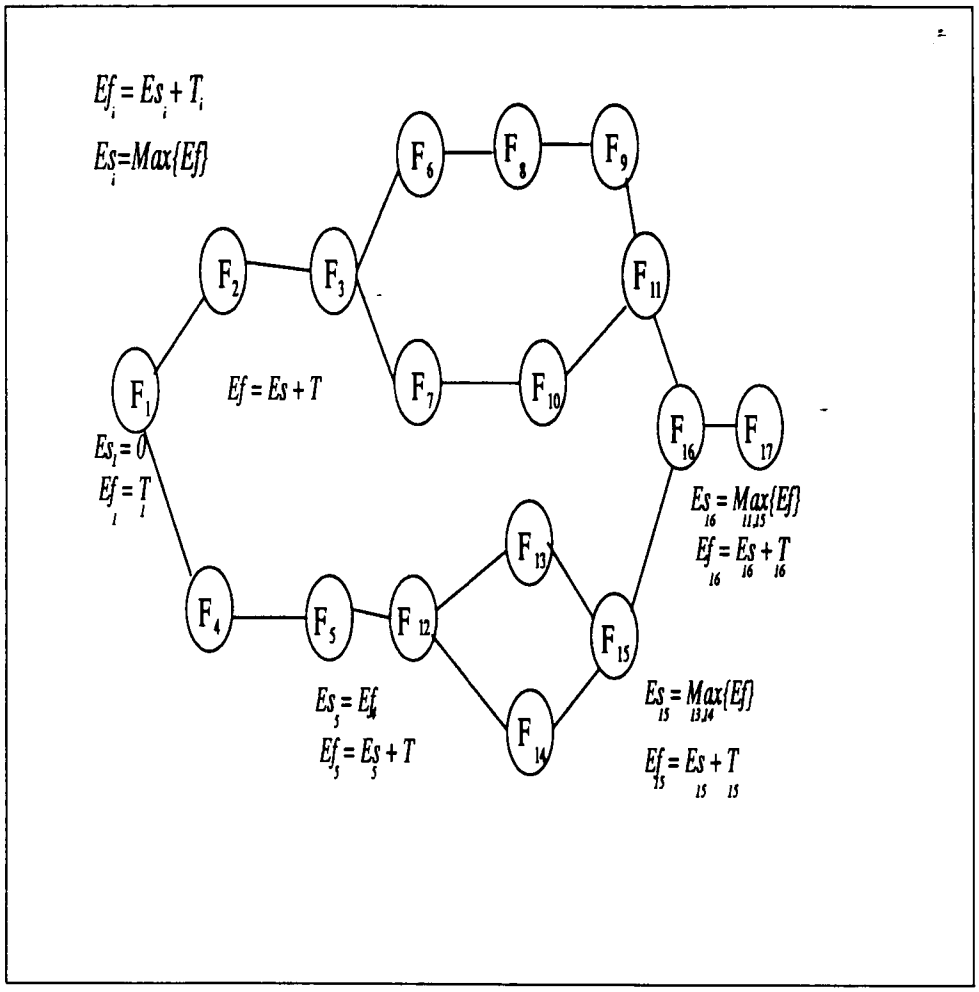

Figure 2.3: The calculation performed during the forward path.

The following values are obtained from the forward pass for each event:

- *Earliest start time* (ES): the maximum of the earliest finish times of activities preceding it.
- *Earliest finish time* (EF): the sum of the early start time of an activity and the the estimated duration of the activity.

A backward pass occurs when the SFG is traversed from a sink node to the source node (Figure 2.4). The following values may be calculated during a backward pass:

- For the sink node, the latest occurrence and earliest occurrence are equated, this is known as the *zero-slack convention.*
- *Latest allowable finish time* (LF): smallest or earliest of the the latest allowable start times of its successor activities.
- *Latest allowable start time* (LS): the latest allowable finish time minus the activity duration.

Obtaining the above values leads to the calculation of *slack* (float) time (s). The slack time is the amount of time by which the actual completion time of an activity can exceed its earliest expected completion time without causing the duration of the overall project to exceed its scheduled completion time. The slack of an event i can be calculated in one of two ways.

$$S_i = LS_i - ES_i \tag{2.1}$$

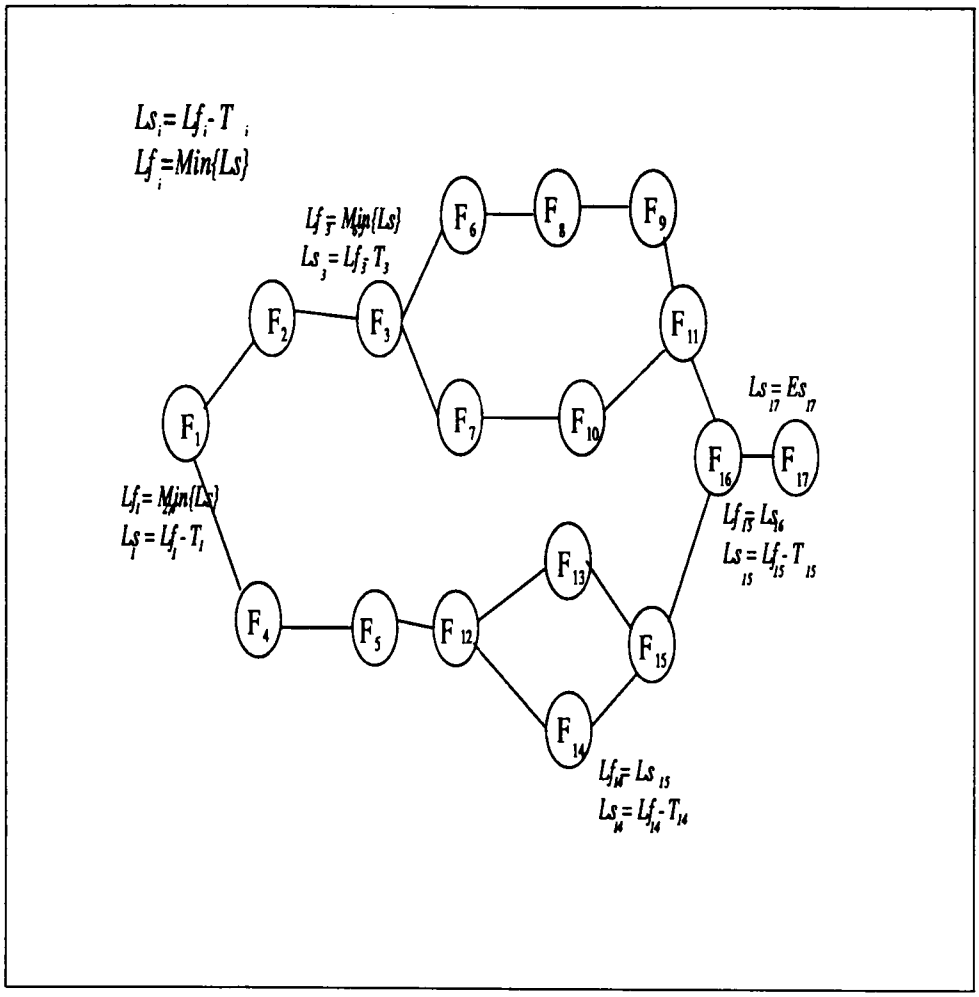

Figure 2.4: The calculation performed during the reverse path.

$$= LF_i - EF_i \tag{2.2}$$

A walk through the graph will identify the path taking the longest time for the total project to be executed. This path is known as the *critical path* and it has to be executed in the same sequential order as depicted by the order of the nodes in the graph. The critical path can also be defined as the path whose slack value equals zero (Figure 2.5). Other events existing off the critical path can be identified.

The time needed to execute a task can be assumed to be a constant value (deterministic) or as a probabilistic value based on *normal, optimistic,* and *worst case* times. In the probabilistic approach, the mean and variance are normally used for the calculations of the stack values. In this work, deterministic values are used. An extension to include the probabilistic approach is possible without a change in the approach taken.

2.2.7 CPM

Critical Path method (CPM) is a technique from management science in which the cost of an event is included when evaluating the total time needed to execute the project. Estimates of both the *normal cost* and the *crash cost* are evaluated. The crash cost of an event is the cost of that event if it is executed in the shortest time possible irrespective of the cost involved. The normal cost is the expected cost when the event is executed in its normal expected time. An optimization is performed based on the target of the project which may be a point between

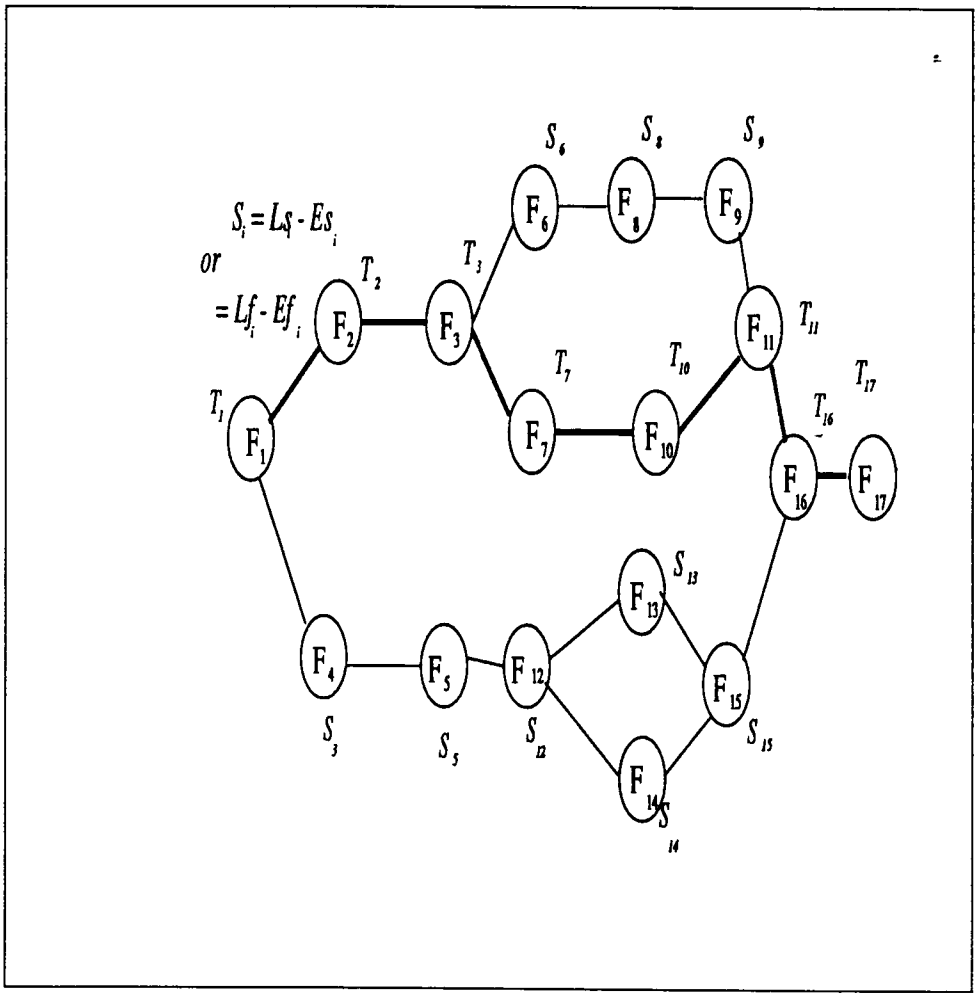

Figure 2.5: The critical path and the slack time found by using the calculations from the forward and reverse paths.

crash time/cost and normal time/cost. After identifying the critical path, the total time delay needed to complete the project is estimated by calculating the time along the critical path (T_{CP}). However, two issues arise. The first issue is when T_{CP} exceeds the allowed time delay for the complete system (T_d), and the second issue occurs when T_{CP} does not exceed T_d but improvement in the overall system performance in other aspects is needed. Several decisions that affect both the cost and the T_{CP} need to be evaluated before changing the system's design.

In CPM, the cost of a project is divided into two parts, direct cost and indirect cost. The direct cost is assumed to decrease with the delay of the project termination while the indirect cost increases (Figure 2.6).

2.2.8 Amdahl's Law

Amdahl's law can be stated as *"The improvement in performance to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used"* [43]. In the context of VLSI design, Amdahl's law indicates that only a certain amount of overall system speedup can be gained by optimizing a portion of the system. This gain in speedup could be sufficient to satisfy the global time constraint. When the goal is to improve the system's efficiency, this gain might be negligible (Figure 2.7).

In this chapter various works related to ASIC design has been introduced. The four major groups are: tutorials and global considerations, complete CAD tools systems, scheduling and allocation, and mapping to specific architectures.

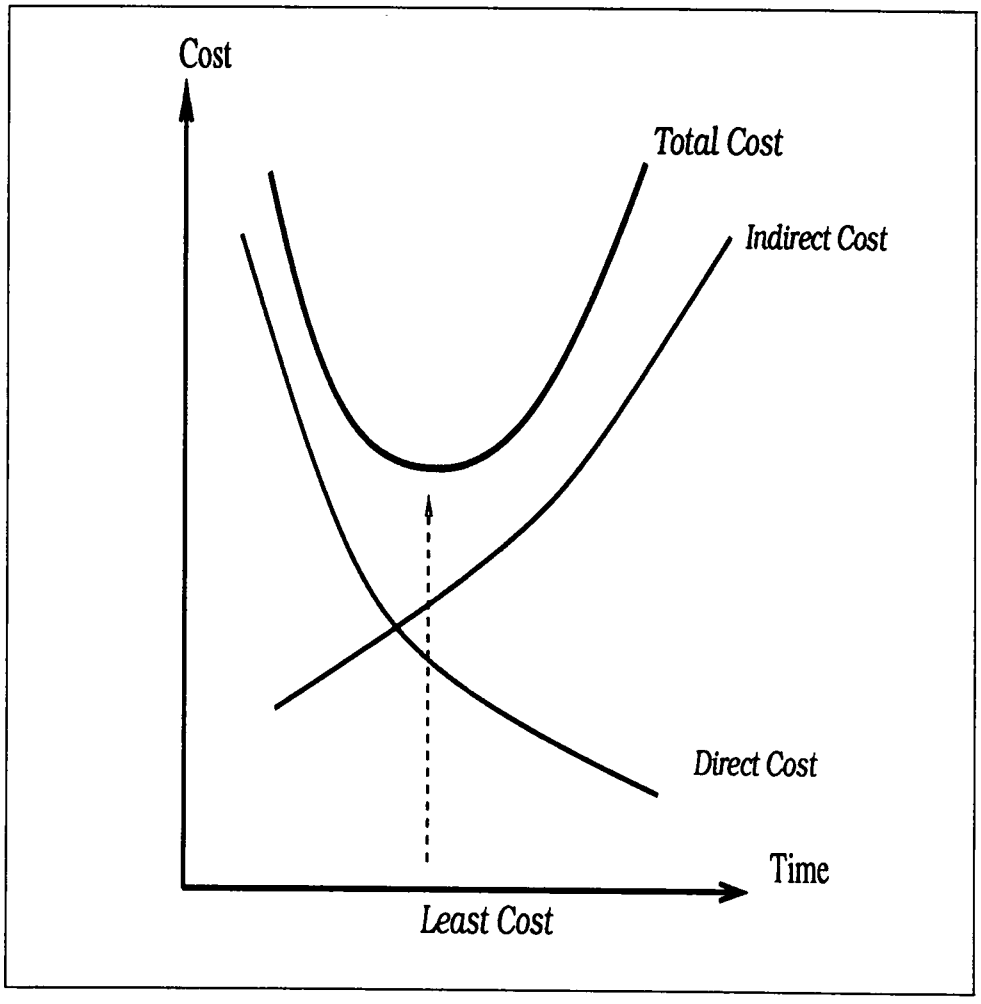

Figure 2.6: The cost function as it changes with time.

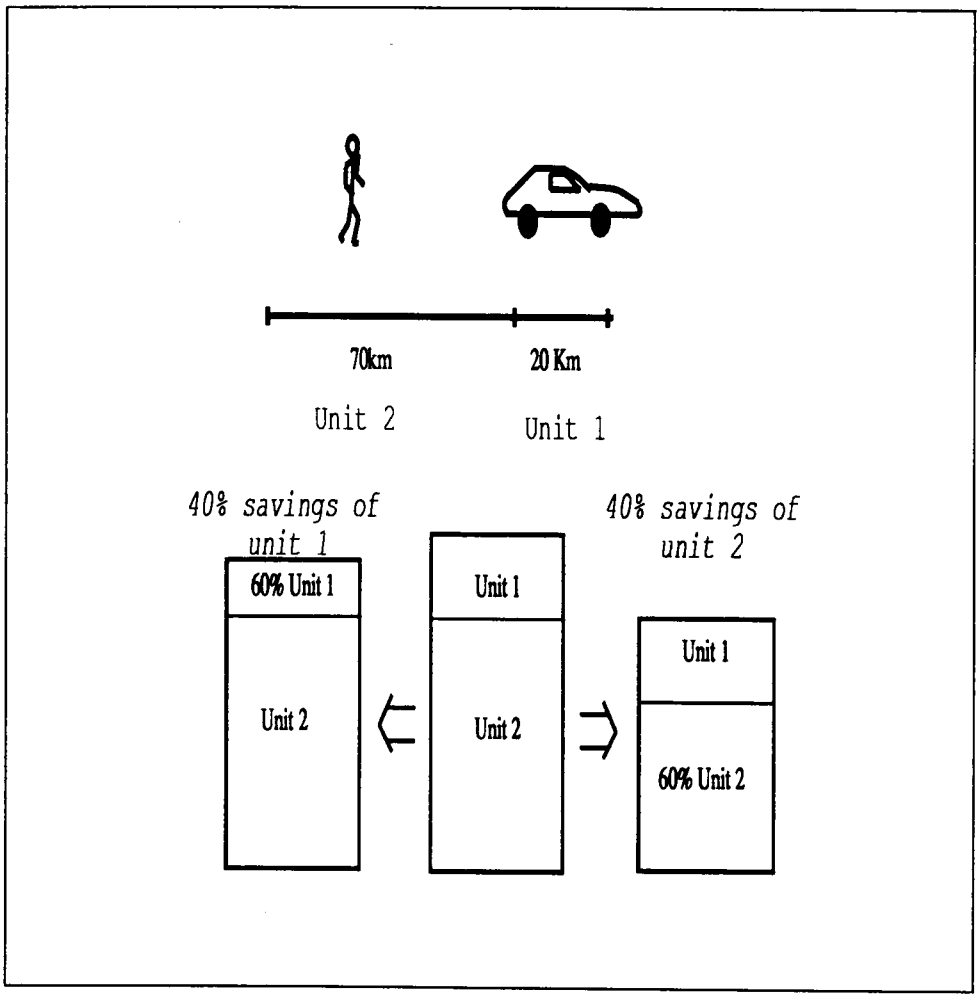

Figure 2.7: Improvement of a specific portion of the design (unit 2) can have more impact on the design than improvement to less important portions (unit1).

Each group was defined and the relevant papers were introduced. It is clear that not enough work has been done to guide the designer in selecting the most suitable algorithm on the highest level possible. Also, other tools that are going to be used in this dissertation were introduced. The next chapter introduces the design methodology proposed and explains how it fits in the previously discussed groups.

CHAPTER 3

The Global Design Methodology

In this chapter, a new design methodology is introduced and in the following chapter, several examples of using this methodology are presented.

As described earlier, the designer is required to select one of several algorithms in order to achieve an optimum design. SFGs have proven to be a popular and convenient method for representing algorithms. Also, in PERT, critical path calculations and the slack time are based on an SFG representation of the design. Selecting SFGs as the method for representing the algorithms in this work follows naturally. An SFG can represent the algorithm at any level of detail, depending on the amount of information present. The SFG can be a simple behavioral representation that relates the input to the output or, if enough details are included, the SFG becomes close to the structural or circuit level description.

The mapping process to an SFG forces the designer to select one of many ways to implement certain portions of the algorithm. Hence, several design alternatives arise. In this work, no specific method for mapping the algorithm into an SFG is introduced. Producing strict guidelines will prevent the designer from being creative in achieving better designs. However, in order to guide the designer in evaluating the resulting SFGs, two figures of merit, other than the

cost function, will be introduced. These figures are related to the slack time.

3.1 Figures of Merit Based on the Value of Slack Time

The value of the slack time is usually used to relax the design constraints of a particular element or to facilitate the scheduling of several operations on the same function unit. The final design should lead to a minimum value of total slack time. If the designer seeks to implement the algorithm in parallel to increase speed, then the total length of the slack of the non critical paths should be minimized. The above facts can be expressed mathematically in the following way: Let the slack of an element i that is off the critical path be defined as s_i. Then first figure of merit (F_1) can be defined as

$$F_1 = \sum_{\forall i} s_i \tag{3.1}$$

F_1 is the sum of all the slack time in the SFG. This gives the designer an idea of how efficient the mapping of the problem into an SFG is. The second figure of merit relates the length of the shortest path to the length of the critical path. Let T_{CP} be the total time along the critical path and S_j be the total amount of slack time along a noncritical path j

$$S_j = \sum_{\forall k} s_k \tag{3.2}$$

where k is any element along the path j. Then F_2 can defined as

$$F_2 = \max_j \left\{ \frac{S_j}{T_{CP}} \right\} \tag{3.3}$$

Another method used to represent the same figure of merit is given by

$$F_2' = \min_j \left\{ \frac{T_j}{T_{CP}} \right\} \tag{3.4}$$

where T_j is the total time along a noncritical path j. F_2 gives an indication of how well balanced the SFG is and whether the mapping process of the algorithm has distributed the work load efficiently over the various paths. These figures will allow the designer to evaluate various ways of mapping the same algorithm into SFGs.

3.2 SET: SFG Evaluation Technique

The design methodology proposed in this work is called SET (SFG Evaluation Technique). It can be summarized in the following steps:

- Map the available algorithms into SFGs. (This mapping may result in SFGs with various options depending on the interaction between various elements or the selection of special processing elements.)
- For each SFG, evaluate the critical path and then evaluate the overall cost using the cost function defined next.
- Based on the cost, select the *best design* (*i.e.* the SFG with the least cost).
- Explore the possibilities of enhancing the selected SFG or some of the building blocks used within the SFG and then calculate the relative gain achieved.

- From the designs obtained in the last step, select the one with the least cost for full implementation.

As can be seen from the above procedure, the selection of the best algorithm depends on the proper definition of the cost function. In this work a cost function similar to the cost function used in CPM will be used. The proposed cost function (c) is divided into two parts. The *direct cost* which includes the cost of the area used, power consumed, design time, and other similar factors that can be included as well. The second part of the cost function, named the *time cost* in this paper, reflects the time delay along the critical path. This cost can be defined for an element within an SFG as well as for the complete SFG. The separation of the time cost and the direct cost allows ease of movement between different levels of details. The element's cost is defined as

$$c_i = c_{\text{time } i} + c_{\text{direct } i} \tag{3.5}$$

and

$$c_{\text{time } i} = w_t T_{di} \tag{3.6}$$

$$c_{\text{direct } i} = w_a A_i + w_d D_i + (w_p P_i)N \tag{3.7}$$

where T_{di}, A_i, D_i, and P_i are the predicted time delay, area, design time and power consumption of element i in the signal flow graph, w_k is the appropriate weight factor, and N is the number of times the data uses this element.

The separation between the time cost and the direct cost is important to be able to evaluate a design with multiple levels of detail. Isolating the time cost

allows the element under evaluation to be included in a larger system. Only those elements along the critical path will affect the total delay of the system. Introducing N in conjunction with the predicted value of the power consumed by each element reflects the importance and the utilization of such an element. This factor is not introduced with the area or the design time since the element has already been designed and the amount of area it occupies is not affected whether the element is used one time or several times.

The total cost (c_T) of the given SFG can be calculated as follows

$$c_{T\ \text{time}} = \sum_{\forall j} N c_{\text{time}\ j} \tag{3.8}$$

$$c_{T\ \text{direct}} = \sum_{\forall i} c_{\text{direct}\ i} + w_d D_T \tag{3.9}$$

$$c_T = c_{T\ \text{direct}} + c_{T\ \text{time}} + I/O\ \textit{overhead} \tag{3.10}$$

where i is any element in the SFG, j is any element along the critical path of the SFG, and *I/O overhead* is the appropriate cost of I/O pins or connections to other components in the system calculated in the same way. Obtaining the final result as a single value satisfies the need to relate the various aspects of the design together. Hence, when the designer attempts to change the coordinates of the design in the design space, the global direction of the movement of the cost function within the space is reflected by the change in the total value of the cost function. A close observation of the various contributing factors will lead to the identification of the factor that affects the design more.

The motivation for introducing weight factors is to include the relative importance of some of these factors or constraints. In one design, the time delay might be the most important factor to be considered while in a different design and under different circumstances, power consumption is of more importance. The ability to reflect such importance mathematically is very important and provides significant flexibility from one design to the next.

When dealing with a large system, several SFGs can be included, along with their individual cost functions so that the cost of the complete system is evaluated. If only one part of the system is changed, only the cost function related to it will be different. However, improving the performance of one element within the design will require time and effort. Amdahl's law is then used to evaluate the improvement gained for the complete system. This gain may give an indication to the designer on the usefulness of spending the time and effort to achieve such an improvement.

The ratio between the change in cost and the change in the factor sought to be improved will give a good indication of where to direct the efforts of the designer. As an example, assume that to reduce the time delay of an element x in the SFG by 1 *nsec* corresponds to a 2% reduction in the overall cost. This reduction might be attributed to both time and direct costs. On the other hand reducing the time delay of another element by 1 *nsec* may cause a 5% reduction in the cost of the same system. The ratio of "Δ *cost*/*nsec*" will give a clear indication to the designer on where to spend the time and effort needed

to enhance the design in hand. A similar comparison can be made between speeding up an element and trimming down its size.

Enhancing one aspect of the design may lead to a degradation in other aspects. If a designer would like to implement a portion of the design in parallel or some other special way to achieve a faster operating frequency, he may need to increase the area occupied by that portion. Grouping all these aspects together in a single cost function (c_T) will keep the designer aware of the effect of various trade-offs made on the location of the design within the design space. It may happen that speeding up an element may cause the area to increase in such a way that the gain due to the speed increase is lost due to an increase in area which increases the total cost. Meanwhile, using weighting factors still provides enough flexibility to reflect the designer's needs and personal philosophy. By utilizing the experience gained from designing other projects, the designer can direct his attention to that aspect with the greatest impact instead of using the traditional trial and error approach or heuristics. Adopting such a technique may help an organization (or a design group) to capture the experience of other designers.

3.3 Including Other Dimensions in the Cost Function

The cost functions given by Equations 3.5 and 3.10 include the dimensions most frequently used by the designers. However, under different conditions, the

need to consider other factors arises. In this section a description of the method used to include such factors will be described by two examples. The first example is related to the cost of routing the various elements within the IC. The second example treats the throughput of the design.

Several researchers [21,26] reported on the dimensions of the silicon area needed to route the signals between different elements. Kurdahi and Parker [26] suggest that the routing area grows by about $O(N^{1.5})$ of the size of the design. Rent's rule [21] was originally developed to describe the relationship between the number of pins to a logic block (N_p) and the number of logic gates within the block (N_g). It was found that

$$N_p = k_p N_g^{\beta} \tag{3.11}$$

where k_p is a constant and β is known as *Rent's constant.* The constants k_p and β take different values depending on the type of organization and technology used. The same relationship can be extended to describe the amount of interconnect area required for VLSI designs. Bakoglu [21] gives a table with different values for the k_p and β constants for different types of architectures.

In this work, two methods are suggested for including the effect of routing. The first method lets the designer estimate the total area and delay needed to route two elements and then insert these predictions as individual blocks between the various elements. In this case, the cost function defined earlier in Equations 3.5 and 3.10 will not change and only more elements will be added to the SFG.

The alternate method to include the effect of routing is to increase the size of the predicted area for each element by an appropriate factor or to be raised to a selected exponent in the range of 1.5 to 2.0 [26]. Again, the definition of the cost function will not change in incorporating such modification.

The second example for increasing the scope of the cost function involves the throughput. The throughput value represents the amount of data that can be processed by the circuit per second. The speed at which the circuit can handle new data depends on the speed of the slowest element along the critical path. Since the cost function is designed to allow multi–level design, the appropriate place to include the effect of the throughput is within the time cost. In this case the total time cost which was defined by Equation 3.8 becomes:

$$c'_{T\ \text{time}} = \sum_{\forall j} N c_{\text{time}\ j} + \max_{i} \{w_{th} T_{d_i}\} \tag{3.12}$$

where T_{d_i} is the time delay of element i along the critical path. Since the maximum value has to be kept separate in order to allow integration into a larger system, the throughput value can be added as a separate value to the cost function. This can be done in the following manner, let $c_{\text{time}\ j}$ and $c_{\text{direct}\ j}$ be defined as in Equations 3.6 and 3.7. Let $c_{\text{throughput}\ j}$ for an element j be defined as

$$c_{\text{throughput}\ j} = w_{th} T_{d_j} \tag{3.13}$$

The total throughput cost for a complete SFG is then defined by

$$c_{T\ \text{throughput}} = \max_{j} \left\{ c_{\text{throughput}\ j} \right\} \tag{3.14}$$

where j is an element along the critical path and the total cost becomes

$$c_T = c_{T\ \text{throughput}} + c_{T\ \text{time}} + c_{T\ \text{direct}} + I/O\ overhead \tag{3.15}$$

Other dimensions of the design space can be added in a similar manner.

In this chapter, SET was introduced along with two figures of merit. In this design methodology, a cost function was introduced to relate the various dimensions of the design space. A discussion on how to use both Amdahl's law and the cost function in order to select the portion of the circuit that needs to be enhanced was presented. Also discussed was a mean to evaluate the return on investment when time and effort are spent on enhancing one aspect of the design.

CHAPTER 4

Design Examples

In this chapter, several examples which demonstrate the strength of the proposed approach are introduced. The first example describes an enhancement for the design of discrete Fourier transform chip. The second example is related to an SFG evaluation while the third example is a study of the tradeoff in using different elements for the same SFG.

4.1 Design of a Fourier Transform Chip

There exist several ways to implement a Discrete Fourier Transform (DFT). Several authors have reported an implementation of the Fast Fourier Transform (FFT) [4,57,54] while others have reported a more direct implementation of the DFT [58,59,60,61]. The goal of this dissertation is not to discuss the various implementations of the DFT or the FFT but to demonstrate how to use the proposed design methodology in evaluating two different designs. The interested reader is referred to [57,62,63]. Two different designs are introduced; one that is a direct implementation of the FFT and the other a distributed arithmetic implementation to directly calculate the DFT [64].

The DFT of a signal $x(n)$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{4.1}$$

for $k = 0, 1, 2, \cdots, N-1$, where $W_N^{kn} = e^{-j2\pi nk/N}$. The basic idea of the FFT is to split the N–point summation into two $N/2$ summations. An 8–point FFT is depicted in Figure 4.1. The basic operation in the FFT is known as the *butterfly* which is described by the following equations

$$R = A + BW_N^{nk} \tag{4.2}$$

$$Q = A - BW_N^{nk} \tag{4.3}$$

Figure 4.2 depicts the basic operations for a complex butterfly (a butterfly with complex input data). As can be seen, four multipliers, three adders, and three subtracters are needed. If the input data are real while W_N^{nk} remains complex, the hardware is reduced to two real multipliers, an adder, and a subtractor. The total number of butterflies needed are 12, which include 48 multipliers and 72 adders. Using the distributed arithmetic approach to perform the multiplication operation will lead to

$$x(n) = \sum_{m=0}^{P-1} x_m(n) 2^m \tag{4.4}$$

$$X(k) = \sum_{n=0}^{N-1} \sum_{m=0}^{P-1} x_m(n) W_N^{nk} 2^m \tag{4.5}$$

where P is the number of bits representing $x(n)$ and $x_m(n)$ is the m^{th} bit of $x(n)$ with value either 0 or 1.

The operation $2^m W_N^{nk}$ represents a left shift of the quantity W_N^{nk} by m bits. The representation $SL(W_N^{nk}, m)$ will be used to indicate this shift. Note that this

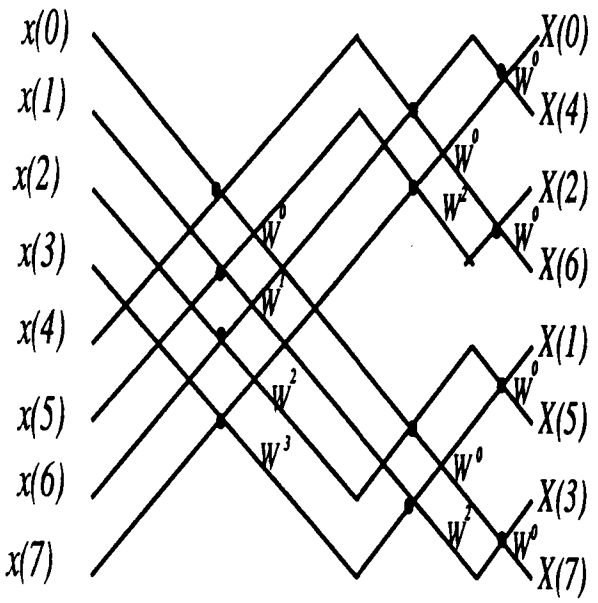

Figure 4.1: The 8–point FFT algorithm using decimation in frequency.

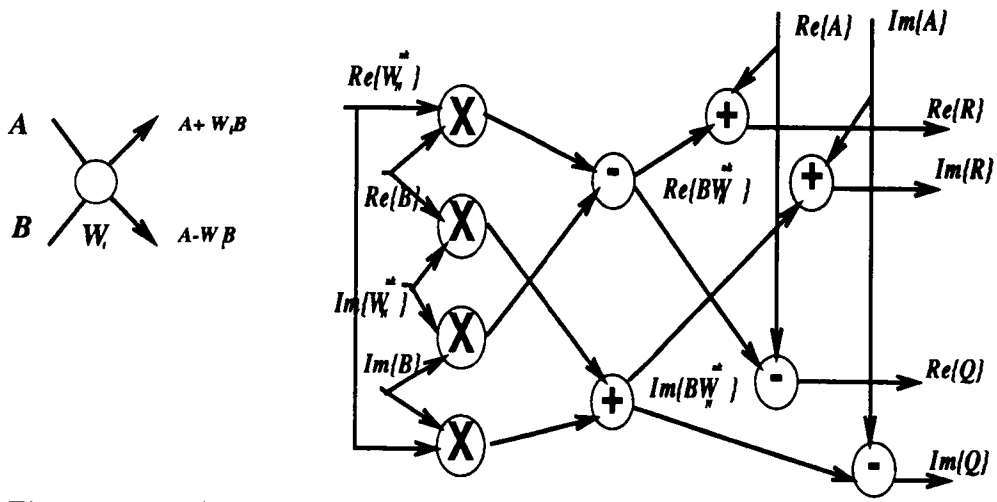

Figure 4.2: The butterfly used in FFT computations and its hardware implementation.

shift need not be implemented using a shift register since a hardwired connection to the appropriate portion of the input bus can be done without incurring no significant delay.

Figure 4.3 depicts a block diagram representation of the circuit needed to calculate the real or the imaginary parts of one frequency domain point using distributed arithmetic. Figure 4.4 depicts the lattice structure for which a group of adders are used to replace the multiplier. The total number of adders needed per lattice cell is equal to $P-1$. M copies are needed to calculate the M–point distributed arithmetic DFT (DADFT). The total number of adders needed becomes

$$\begin{aligned} ADD_{FFT} &= 2(M(P-1)+M) \\ &= 2MP \end{aligned} \tag{4.6}$$

In the following discussion, the direct implementation of the FFT will be referred to as the FFT while the DADFT will refer to the implementation described by Equations 4.4 and 4.5.

It is known that the FFT requires $\frac{N}{2}\log_2 N$ complex additions and $\frac{N}{2}\log_2 N$ complex multiplications, while the DFT requires N^2 complex additions and N^2 complex multiplications [57,62,65]. However, these calculations are only valid for a single processor machine. A question arises whether the above direct implementation of the FFT, which is the simplest from the point of view of the control circuitry, is more efficient for VLSI implementation than the DADFT.

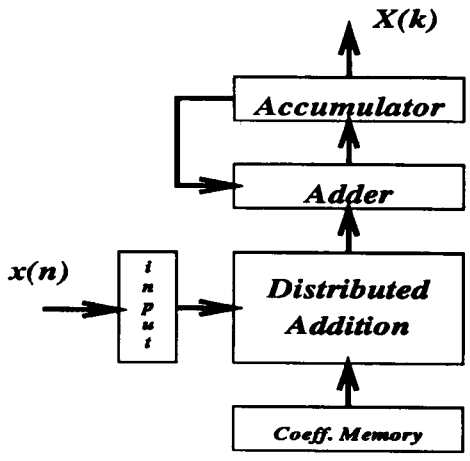

Figure 4.3: The block diagram of the circuit needed to calculate one frequency domain point.

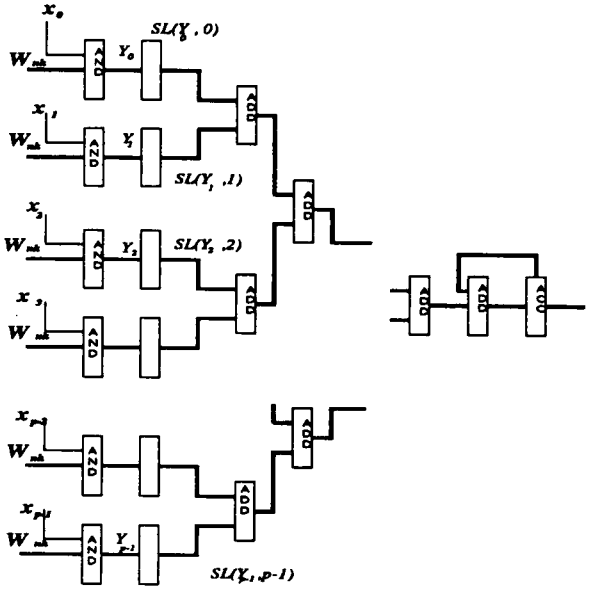

Figure 4.4: The use of distributed arithmetic to replace the use of multipliers by a group of adders.

In order to answer this question, one has to either design both circuits and compare them at the final stage or select only one and pursue it to a complete design saving the design time and effort of working on two designs fully. Using the methodology described in the previous chapter, such a decision can be made without further considering more details.

Using the Lager design tools [49], the basic components were developed using standard cells. Table 4.1 includes the various parameters needed for calculating the cost function. For simplicity, the design time needed and power consumed were neglected. Calculating the cost for the FFT requires the detection of the critical path for the complete SFG and that of the basic elements, the butterfly in this case. In order to calculate the critical path the various paths through which the data flow must be traced. As can be seen in Figure 4.2, the longest path is that of $Re\{W_N^{nk}\}$, $Im\{W_N^{nk}\}$, $Re\{B\}$, or $Im\{B\}$ where these data values have to be processed by one multiplier and two subtracters. Adding the time required to perform these operations and neglecting time delay and loading effects this will lead to 62 *nsec*. This time is bigger than the time required by the value of $Re\{B\}$ to go through one multiplier, one adder, and one subtracter which give a total delay of 60 *nsec*. Similarly, the value of $Im\{A\}$ will go through two adders and a subtracter leading to a total delay of 50 *nsec*. Hence, the critical path is along the path that includes one multiplier and two subtracters (see Figure 4.5). For the 8–point FFT depicted in Figure 4.1, any time domain point has to go through three butterflies giving equal paths for the data. Hence, all the paths

are equally long and each is three butterflies long. Tables 4.2 and 4.3 depict the calculations for the cost function for the complex and real butterflies. Tables 4.4 and 4.5 depict the cost functions for the 8–point FFT with complex and real input data. Note that the direct cost, which represents the area, is expressed in terms of 10^6 units. For the DADFT all data points have to go through one group of AND gates, four adders, and one accumulator. Hence, all paths are equally long and the critical path is chosen to be any of the them. The cost of one copy of the lattice cell is depicted in Table 4.6.

In order to calculate the 8–point DADFT either eight copies of the lattice cell are used simultaneously or one copy is used sequentially. The cost of each implementation is depicted in Table 4.7. In both cases two sets are needed to calculate the real and imaginary parts. As can be seen, the parallel implementation of the DADFT costs less than the other two implementations (real FFT and serial implementation of DADFT).

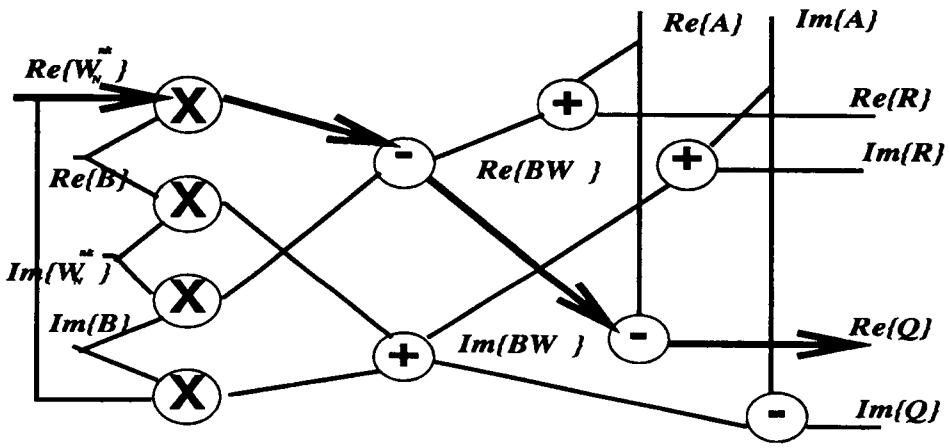

Figure 4.5: The critical path within a butterfly.

Table 4.1: The basic blocks used and their parameters.

unit name	Area (λ^2)	Time delay ($nsec$)
adder (16 bits)	900 × 1000	16
subtractor (16 bits)	1000 × 1000	18
multiplier (8 bits)	1800 × 1800	26
register (16 bits)	500 × 120	7
AND gates (16 bits)	400 × 60	5

Table 4.2: Calculating the cost function for a complex butterfly.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	4	3	3	–
Critical path	1	0	2	–
Direct Cost	12.96	2.7	3	18.66
Time Cost	26.0	0.0	36.0	62

Table 4.3: Calculating the cost function for a real butterfly.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	2	1	1	–
Critical path	1	0	1	–
Direct Cost	6.48	0.9	1	8.38
Time Cost	26.0	0.0	18.0	44.0

Table 4.4: Calculating the cost function for an 8–point FFT (complex data).

units	Complex Butterflies	Real Butterflies	Subtotals
Total number	12	0	–
Critical path	3	0	–
Direct Cost	223.92	0.0	223.92
Time Cost	186	0.0	186

Table 4.5: Calculating the cost function for an 8–point FFT (real data).

units	Complex Butterflies	Real Butterflies	Subtotals
Total number	8	4	–
Critical path	2	1	–
Direct Cost	149.28	33.52	182.80
Time Cost	124.0	44.00	168.0

Table 4.6: Calculating the cost function for one lattice cell.

units	Adders	AND	Register	Subtotals
Total number	8	8	1	–
Critical path	4	1	1	–
Direct Cost	7.2	0.19	0.06	7.45
Time Cost	64.0	5.0	7.0	76.0

Table 4.7: The cost function of both the parallel and serial implementations of the DADFT.

Type of implementation	Parallel	Serial
Direct Cost	59.62	7.45
Time Cost	76.0	608.0
Total Cost (2 copies)	195.23	622.9

In case the area is more critical than the time delay the weights can be set to reflect such a situation. Let $w_T = 0.2$ and $w_A = 1$. This is reflected in the cost function of the designs as appears in Table 4.8. It can be seen that the serial implementation, though slow, is the more efficient implementation in this case. Table 4.9 shows the result when more importance is given to the time delay ($w_T = 1.0$ and $w_A = 0.5$). Figure 4.6 is the DADFT implementation using Lager CAD tools.

Table 4.8: Adjusting the weights to reflect the importance of the area ($w_A = 1.00$ and $w_T = 0.2$).

Algorithm	Direct Cost	Time Cost	Total
Complex FFT	223.92	37.2	261.12
Parallel DADFT	238.48	15.2	253.66
Serial DADFT	29.8	121.6	151.41

Table 4.9: Adjusting the weights to reflect the importance of the time delay ($w_A = 0.5,\ w_T = 1.0$).

Algorithm	Direct Cost	Time Cost	Total
Complex FFT	111.96	168.0	297.96
Parallel DADFT	119.24	76.0	195.23
Serial DADFT	13.92	608.0	622.9

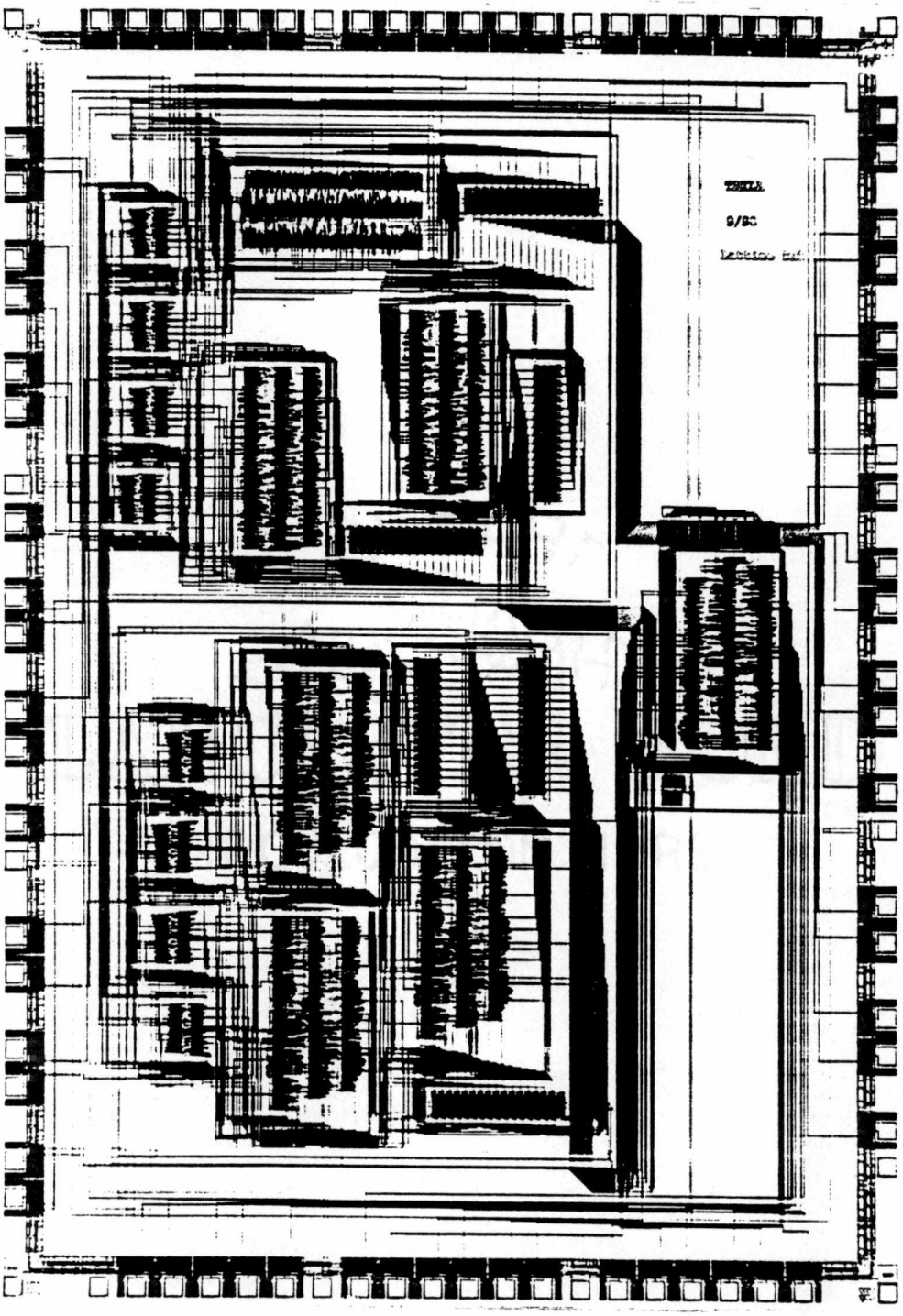

Figure 4.6: Part of the chip that is designed using **Lager** cad tools.

4.2 Signal Flow Graph Analysis for the DCT

Another example that depicts the strength of the proposed methodology is presented next. In this example, several techniques to calculate the Discrete Cosine Transform (DCT) are depicted by their SFGs. A cost function is used to select the best algorithm for hardware implementation by analyizing the various SFGs.

The DCT is a popular transform as it behaves in a manner close to the optimal Karhunen–Loeve Transform (KLT) [5,66,67,68,69]. It has also gained more attention since being selected as the standard for video eompression by the CCITT/ISO (*Consulative Committee for International Telegraphy and Telephony/International Standard Organization*).

The basic operation of the DCT is described by the following equation

$$X(m) = \left(\frac{2}{N}\right)^{1/2} k_m \sum_{n=0}^{N} k_n cos\left(\frac{\pi mn}{N}\right) x(n) \tag{4.7}$$

for $m = 0, 1, ..., N$, $p = 0, 1, ..., N$ where k_p is defined as

$$\begin{aligned} k_p &= \frac{1}{\sqrt{2}} \quad \text{when} \quad p = 0 \text{ or } N \\ &= 1 \quad \text{when} \quad p \neq 0 \text{ and } N \end{aligned}$$

The SFGs depicted in Figures 4.7, 4.8, 4.9, 4.10, 4.11, and 4.12 are variations to calculate the same DCT. These SFGs are from [67,70]. The critical path was calculated for each of these SFGs. These are depicted in Figures 4.13, 4.14, 4.15, 4.16, 4.17, and 4.18. Tables 4.10, 4.11, 4.12, 4.13, 4.14 and 4.15 depict the cost

function for each of the SFGs using the same building blocks described in Table 4.1. These results are summarized in Table 4.16. As can be seen, SFG 6 has the least time cost and the least total cost. However, SFG 2 has the least direct cost. SFG 1 is the most expensive method to evaluate a 16–point DCT.

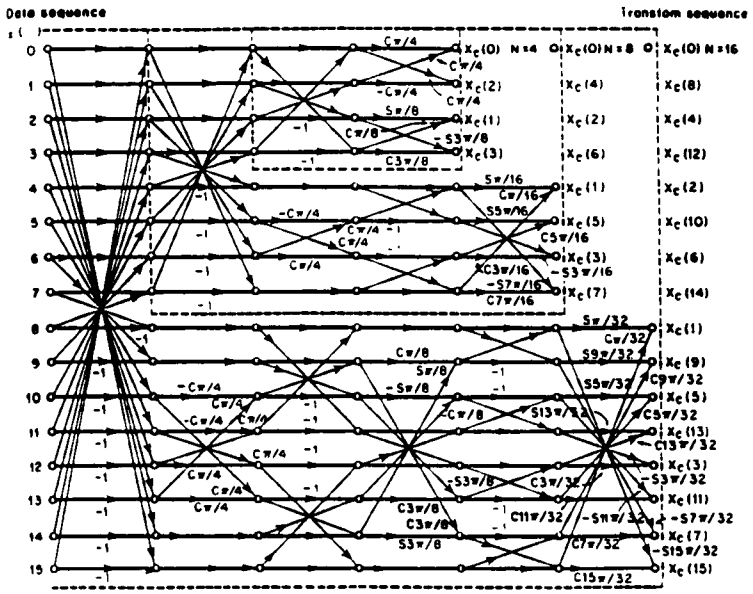

Figure 4.7: The SFG 1 for DCT, $N = 16$.

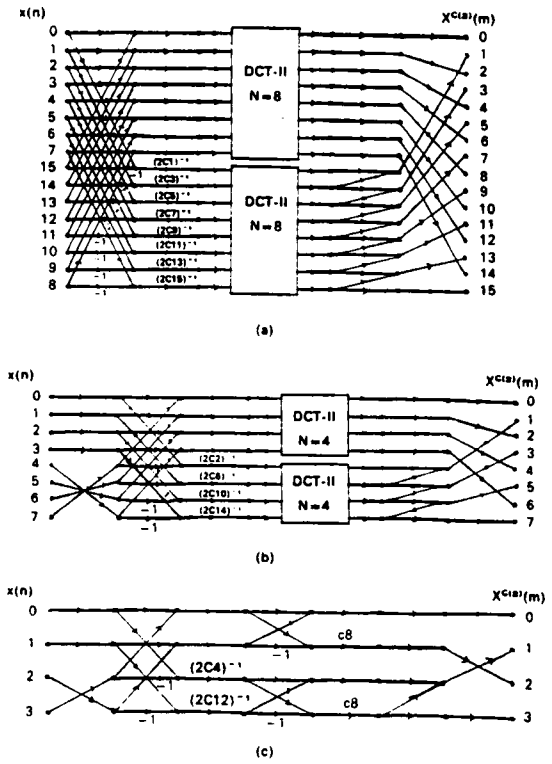

Figure 4.8: The SFG 2 for DCT, $N = 16$ using 8–point and 4–point DIF DCTs as building blocks.

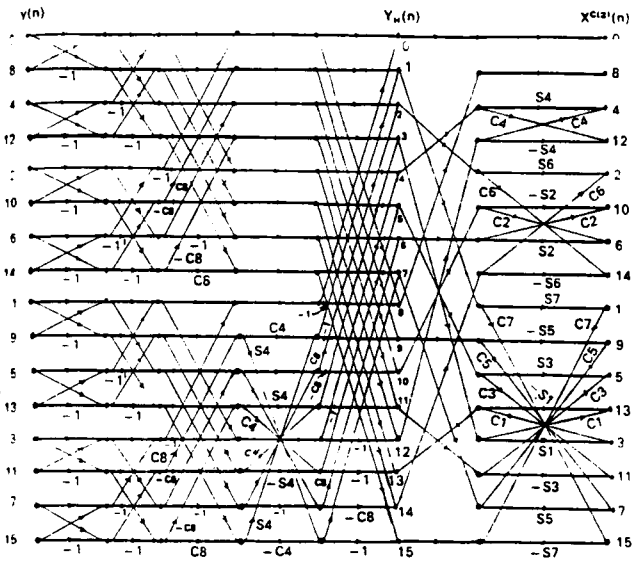

Figure 4.9: The SFG 3 for DCT via a Discrete Hartley Transform, $N = 16$.

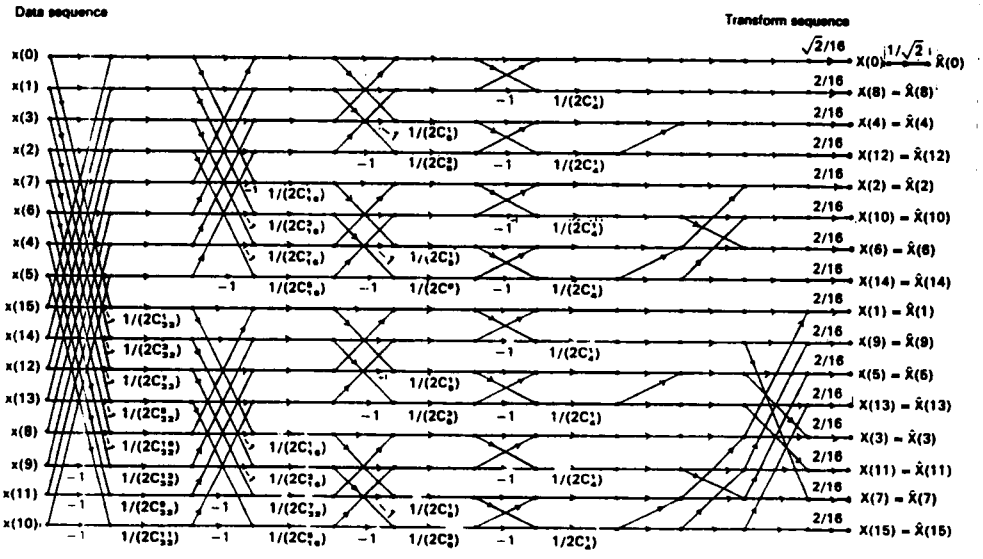

Figure 4.10: The SFG 4 for DCT based on Lee's algorithm, $N = 16$.

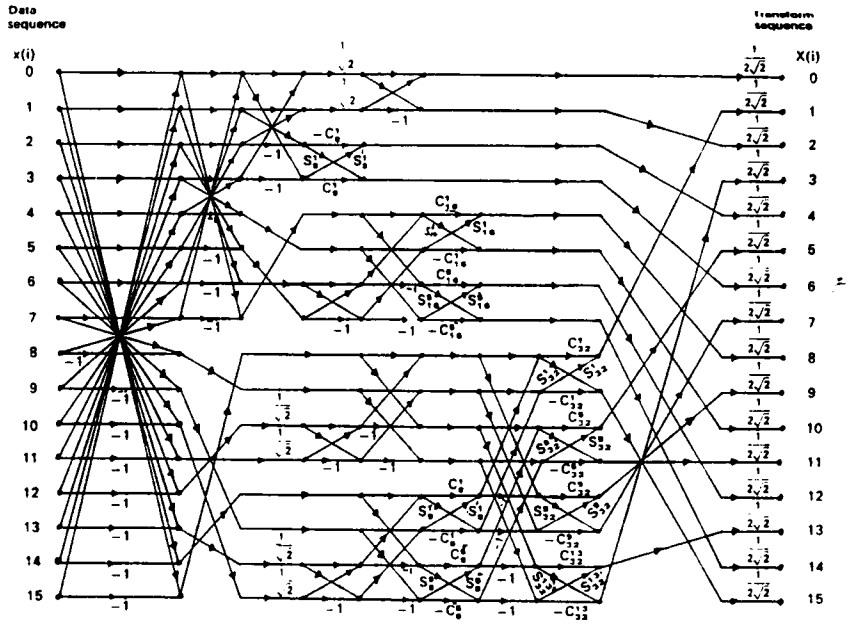

Figure 4.11: The SFG 5 for DCT based on Wang's algorithm, $N = 16$.

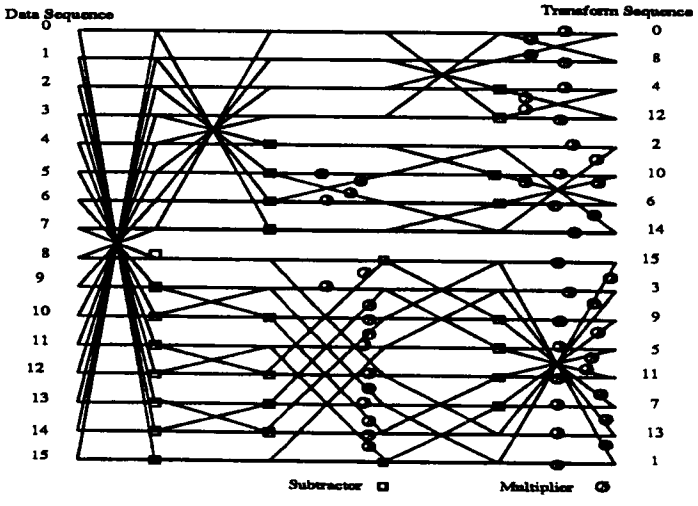

Figure 4.12: The SFG 6 for DCT based on Justand's implementation, $N = 16$.

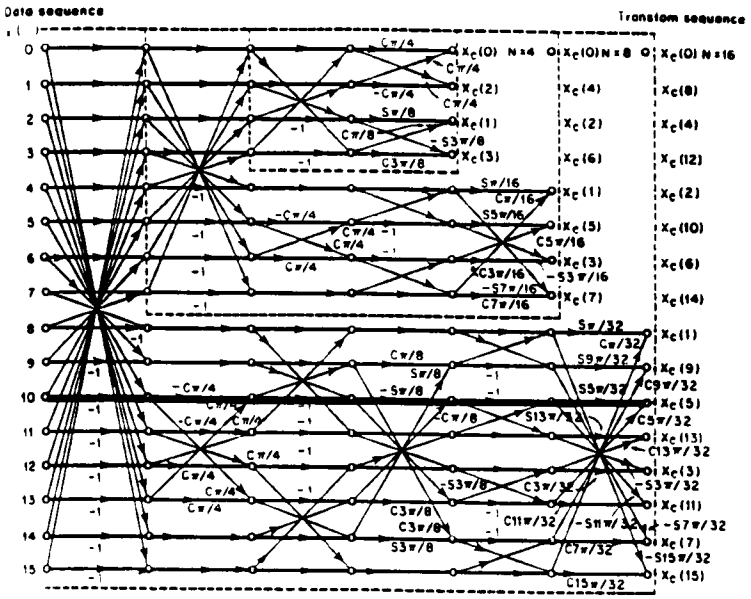

Figure 4.13: The critical path for SFG 1 for the DCT.

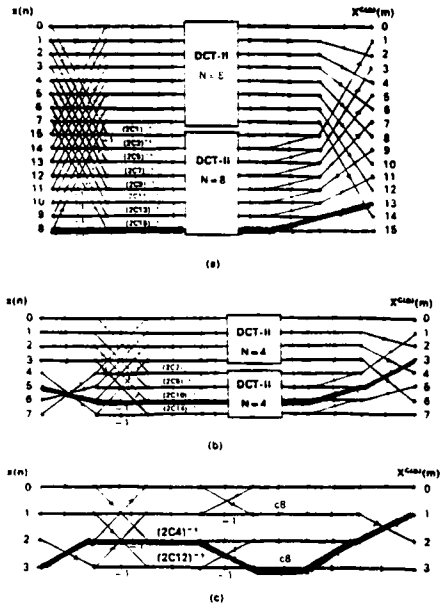

Figure 4.14: The critical path for SFG 2 for the DCT.

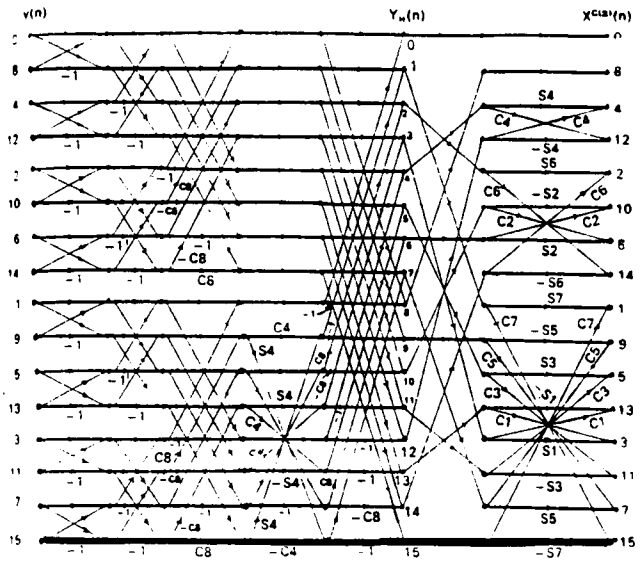

Figure 4.15: The critical path for SFG 3 for the DCT.

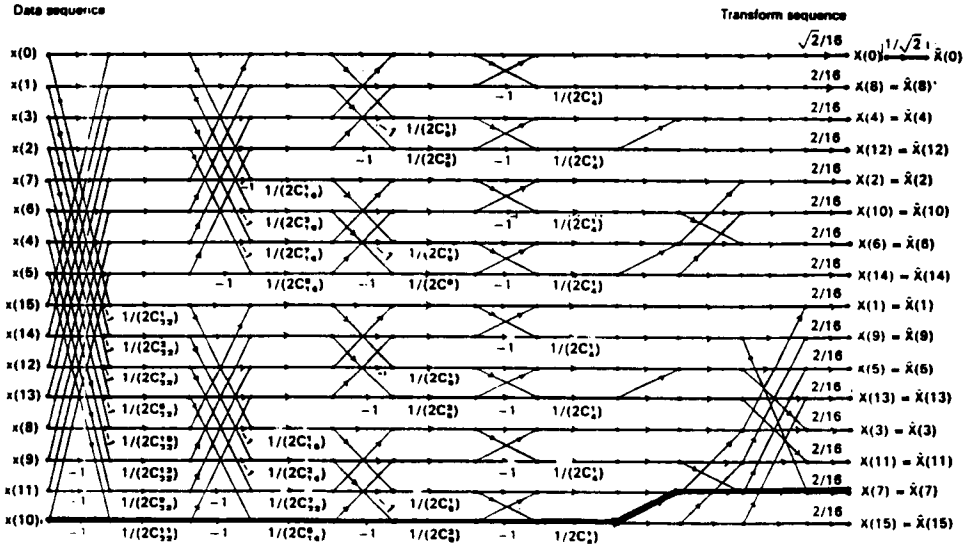

Figure 4.16: The critical path for SFG 4 for the DCT.

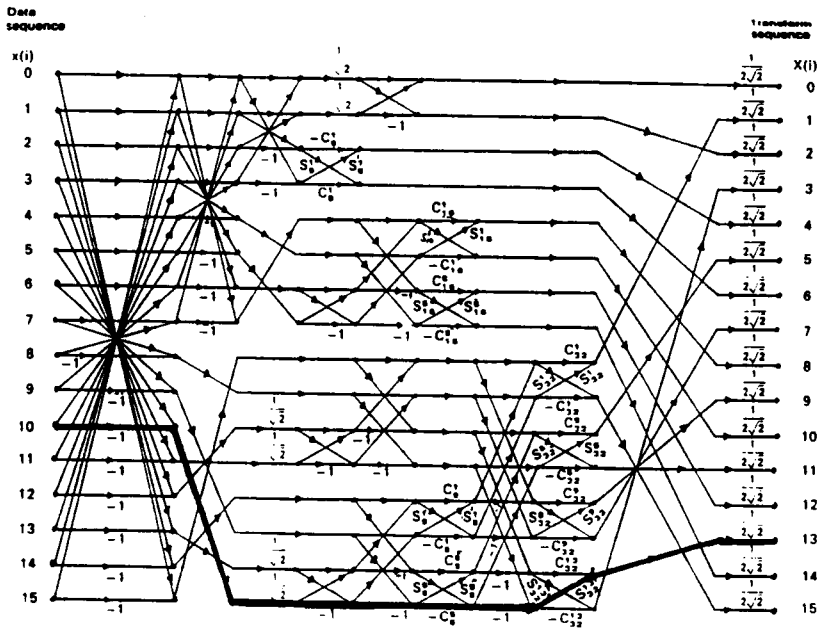

Figure 4.17: The critical path for SFG 5 for the DCT.

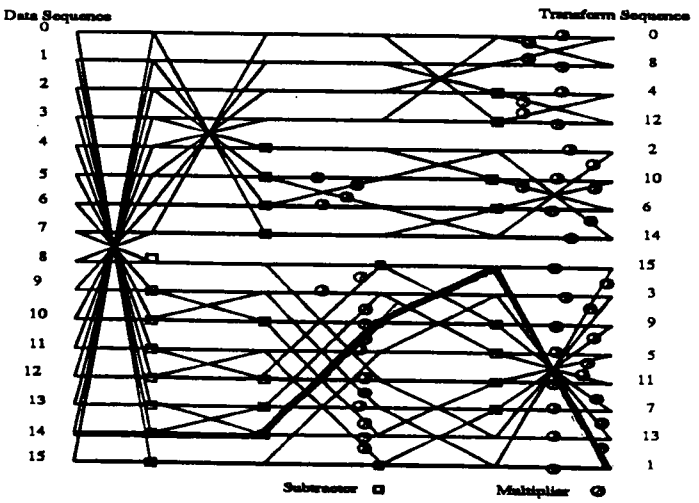

Figure 4.18: The critical path for SFG 6 for the DCT.

Table 4.10: Calculating the cost function for SFG 1 to compute the DCT.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	52	63	16	–
Critical path	3	6	3	–
Direct Cost	168	56.7	16	241.18
Time Cost	78	96	54	228.00

Table 4.11: Calculating the cost function for SFG 2 to compute the DCT.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	32	49	32	–
Critical path	4	3	4	–
Direct Cost	103.68	44.1	32	179.78
Time Cost	104	48	72	224.0

Table 4.12: Calculating the cost function for SFG 3 to compute the DCT.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	48	68	26	–
Critical path	3	3	3	–
Direct Cost	155.52	61	26	242.72
Time Cost	78	48	54	180.00

Table 4.13: Calculating the cost function for SFG 4 to compute the DCT.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	48	42	32	–
Critical path	4	2	4	–
Direct Cost	155.52	37.8	32	225.32
Time Cost	104	32	72	208.00

Table 4.14: Calculating the cost function for SFG 5 to compute the DCT.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	56	46	28	–
Critical path	4	2.0	4.0	–
Direct Cost	181.44	41.4	28	250.84
Time Cost	104	32	72	208.00

Table 4.15: The cost function for SFG 6 to compute the DCT by Justand et al.

units	Multipliers	Adders	Subtracters	Subtotal
Total number	48	47	25	–
Critical path	3	2	3	–
Direct Cost	155.52	42.3	25	222.82
Time Cost	52	32	54	164.00

Table 4.16: Summary of the cost functions for the SFGs to calculate the DCT.

SFG	Direct Cost	Time Cost	Total
1	241.18	228	469.18
2	1789.78	224	403.78
3	242.72	180	422.72
4	225.32	208	433.32
5	250.84	208	458.84
6	222.82	164	386.82

4.3 Tradeoffs in Using Different Elements for the Same SFG

A third design example is used to demonstrate the strength of the proposed methodology which is more global than other approaches that are reported [20,71]. This is the example treated by Jain *et al.* [17,20,71]. who used the SFG depicted in Figure 4.19. In their example, Jain *et al.* introduced a set of adders (add) and a set of multipliers ($mult$) with different sizes and time delays, as is depicted in Table 4.17.

The goal is to find one adder and one multiplier ($add, mult$) that will allow the above SFG to fit in a 50,000 mil^2 area. For the non-pipelined approach, their predictor [20] estimated the total number of multiplexers and registers. This data is taken directly from their paper. The cost function was calculated for the given SFG using all possible combinations of adders and multipliers. Only the time delay and estimated silicon area were used for the cost function and the calculations were performed with $w_a = w_t = 1$. The cost functions were sorted based on the direct cost (area in this case) and the data appear in Table 4.18.

Two combinations provide a total area of less than 50,000 mil^2: ($add3, mult3$) and ($add2, mult3$). However, the total cost of the second combination is less than that of the first, which leads to the same conclusion as that of Jain *el al.* [17,20]. Their work shows that the set ($add2, mult3$) is the best combination that satisfies the given design requirements. If the area constraint

is removed, the combination $(add1, mult2)$ will provide the design with the least cost function if only area and time delay are concerned.

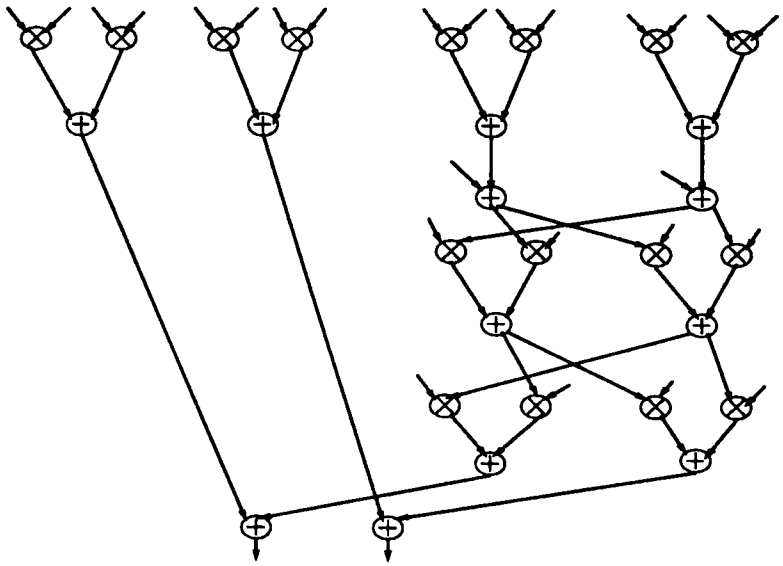

Figure 4.19: The SFG used by Jain *et al.* to demonstrate their work.

Table 4.17: Data of the elements used to implement the SFG depicted in Figure 4.19.

Function	Area ($10^6 mils$)	Delay ($nsec$)
add1	42	340
add2	28.8	530
add3	12	1510
mult1	490	375
mult2	98	2950
mult3	71	7370

Table 4.18: The cost of the SFG shown in Figure 4.19 using different combinations of adders and multipliers. Cost is sorted based on direct cost.

$(add, mult)$	Direct Cost (10^6 mil^2)	Time Cost (10 $nsec$)	Total Cost
3,3	45.38	296.6	341.98
2,3	48.74	247.6	296.34
1,3	51.38	238.1	289.48
3,2	56.18	164.0	220.18
2,2	59.54	115.0	174.54
1,2	62.18	105.5	167.68
3,1	212.98	89.75	299.73
2,1	216.34	37.75	254.09
1,1	218.98	28.25	247.23

In this chapter, three examples were presented, each showing an aspect of the SET. In the first example, the direct implementation of the FFT and the DADFT were compared and designed fully to verify the result obtained by the cost function. The second example dealt with the DCT. Various SFGs were introduced and the most suitable one for VLSI implementation was selected. The last example dealt with the case where a single SFG is available and different building blocks can be used. A combination of an adder and a multiplier were selected to give the best design within the given constraints. This last example showed that the proposed methodology is more global than other reported work.

CHAPTER 5

Summary, Conclusions, and Directions for Future Work

This dissertation has presented a new design methodology that treats the design problem as an optimization problem of a multi–dimensional design space. The various work related to ASIC design have been introduced. The four major groups: tutorials and global considerations, complete CAD tools systems, scheduling and allocation, and mapping to specific architectures were described. Each group was defined and the relevant papers were introduced. It was clear that not enough work has been done to guide the designer in selecting the most suitable algorithm on the highest level possible. Also, other tools that were used in this dissertation were introduced. A cost function that relates the various dimensions of the design space was defined. This cost function has several parts: direct, time, and throughput parts. It is defined such that a multi–level approach for the design can be used. A discussion was given on how to use Amdahl's law and the cost function defined in order to select the portion of the design that needs to be enhanced such that the return on investment is maximized.

Several examples were introduced to demonstrate the strength of the proposed methodology. These examples included a comparison between a direct implementation of the FFT and one using distributed arithmetic to calculate

the DFT. It was shown that using distributed arithmetic leads to more savings in both area and time delay. A second example demonstrated the use of this methodology to select the best SFG that depicts a method to implement a DCT. The third example was dedicated to selecting the most suitable elements to build a given SFG. This example was also used to show that the work introduced here is more global than other works described in the literature.

This methodology can be extended in several ways. The first extension would be to incorporate this methodology with a general CAD tool system. After representing the algorithm as an SFG, the methodology will evaluate a mapping and will select, from an existing database, the building blocks that will produce the design with the least cost. A comprehensive database would then be built. This database should include various building blocks that perform the same function and have different properties. Also, the coordinates of each building block with the design space would be included. Predicted values for the blocks that do not exist could be added as the designer wishes. Such a database will act as a pool of information to serve a group of designers.

Another extension would be to include the different effects of fabrication and packaging technologies on the design. New packaging techniques, such as Multi-Chip Module (MCM), provide the designer with great improvements. However, these improvements should be considered at the highest level of the design process and the design should be directed towards a specific fabrication from the highest level possible.

Since SFGs are a basic component of the work, the ability to manipulate the SFG automatically and map portions of the design in a parallel fashion or a serial fashion and then evaluate the cost function would provide great strength to this work.

An immediate research area would be to use fuzzy logic to develop a fuzzy algorithm that evaluates the cost function. The weights would be replaced by membership functions that describe how important each element is in the cost function to the designer.

The area of VLSI is growing rapidly. The growth rate has not decreased in the last few years. This motivates more work to develop software that aids the designer in achieving a better design and best utilizes the available resources.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] P. Losleben, "Computer Aided Design for VLSI," in *Very Large Scale Integration Fundamentals and Applications*, (D. F. Barbe, ed.), Springer–Verlag, 1980.

[2] K. Ueda, R. Kasai, and T. Sudo, "Layout Strategy, Standarization and CAD tools," in *Layout Desgin and Verification*, (T. Ohtsuki, ed.), North–Holland, 1986.

[3] E. Swartzlander, "VLSI Architecture," in *Very Large Scale Integration VLSI Fundamentals and Applications*, (D. F. Barbe, ed.), Springer–Verlag, 1980.

[4] R. Linderman, P. Chau, W. Ku, and P. Reusens, "CUSP: A 2–μm CMOS Digital Signal Processor," *IEEE Journal of Solid–State Circuits*, vol. sc–20, pp. 761–769, June 1985.

[5] J. Carlac'H and A. L. Sicre, "An ASIC Solution to ADCT Image Decoding For Telecom Services," in *VLSI Signal Processing III*, (R. Brodersen and H. Moscovitz, eds.), IEEE, 1988.

[6] J. Ward, P. Barton, J. Roberts, and B. Stanier, "Figures of Merit for VLSI Implementations of Digital Signal Processing Algorithms," in *IEE Proceedings*, pp. 64–70, February 1984.

[7] C. C. Yu, L. J. Wu, and Y. S. Wu, "Constant Capacity: An Information Theoretic Approach to VLSI/DSP Architecture," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 2528–2531, 1989.

[8] M. McFarland, A. Parker, and R. Camposano, "Tutorial on High–Level Synthesis," in *25th ACM/IEEE Design Automation Conference*, pp. 330–336, 1988.

[9] K. Keutzer, "Three Competing Design Methodologies for ASICs: Architectural Synthesis, Logic Synthesis and Module Generation," in *26th ACM/IEEE Design Automation Conference*, pp. 308–313, 1989.

[10] A. L. Sangiovanni–Vincentelli, "An Overview of Synthesis Systems," in *IEEE Custom Integrated Circuits Conference*, pp. 221–225, 1985.

[11] T. Tanaka, T. Kobayashi, and O. Karatsu, "HARP: Fortran to Silicon," *IEEE Transactions on Computer–Aided Design*, vol. 8–20, pp. 649–660, June 1989.

[12] T. Tsuda and Y. Mochida, "A Practical Study on the Architecture of VLSI DSP," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 2179–2182, 1986.

[13] C. Wu and P. Cappello, "Application–Specific CAD of VLSI Second–Order Sections," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36–5, pp. 813–825, May 1988.

[14] S. Smith and R. Morgan, "Generic ASIC Architecture and Synthesis Scheme for DSP," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 2413–2416, 1989.

[15] D. C. Ashby, R. A. McConnel, and C. C. Yu, "An Integrated Enviroment for DSP Systems Design," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 2524–2527, 1989.

[16] G. Borriello and E. Detjens, "High–Level Synthesis: Current Status and Future Directions," in *25th ACM/IEEE Design Automation Conference*, pp. 477–482, 1988.

[17] R. Jain, K. Kucukcakar, M. Mlinar, and A. Parker, "Experience with the ADAM Synthesis System," in *26th ACM/IEEE Design Automation Conference*, pp. 56–61, 1989.

[18] B. Haroun and M. Elmasry, "SPAID: An Architectural Synthesis Tool for DSP Custom Applications," *IEEE Journal of Solid–State Circuits*, vol. 24–2, pp. 426–435, April 1989.

[19] G. Goossens, J. Vandewalle, and H. D. Man, "Loop Optimization in Register–Transfer Scheduling for DSP systems," in *26th ACM/IEEE Design Automation Conference*, pp. 826–831, 1989.

[20] R. Jain, M. Mlinar, and A. Parker, "Area–Time Model for Synthesis of Non–Pipelines Designs," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 48–51, 1988.

[21] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI.* Reading, MA: Addison Wesley, 1990.

[22] Y. S. Wu and L. J. Wu, "Signal Flow Architecture," *Proceeding of COMPCON fall 84*, September 1984.

[23] S. Devadas and A. Newton, "Algorithms for Hardware Allocation in Data Path Synthesis," *IEEE Transactions on Computer–Aided Design*, vol. 8–7, pp. 768–781, July 1989.

[24] M. Potkonjak and J. Rabaey, "A Scheduling and Resource Allocation Algorithm For Hierarchical Signal Flow Graphs," in *26th ACM/IEEE Design Automation Conference*, pp. 7–12, 1989.

[25] A. Parker, J. Pizarro, and M. Mlinar, "MAHA: A Program for Datapath Synthesis," in *23rd ACM/IEEE Design Automation Conference*, pp. 461–466, 1986.

[26] F. Kurdahi and A. Parker, "PLEST: A Program for Area Estimation of VLSI Integrated Circuits," in *23rd ACM/IEEE Design Automation Conference*, pp. 467–473, 1986.

[27] N. Park and A. Parker, "SEHWA: A Program for Synthesis of Pipelines," in *23rd ACM/IEEE Design Automation Conference*, pp. 454–460, 1986.

[28] E. D. Lagnese and D. E. Thomas, "Architectural Partitioning for System Level Design," in *26th ACM/IEEE Design Automation Conference*, pp. 62–67, 1989.

[29] P. Paulin, "Scheduling and Binding Algorithms for High–Level Synthesis," in *26th ACM/IEEE Design Automation Conference*, pp. 1–6, 1989.

[30] P. Paulin and J. Knight, "Forced–Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, vol. 8–6, pp. 661–679, June 1989.

[31] H. Cai and J. J. A. Hegge, "Comparison of Floorplanning Algorithms for Full Custom ICs," in *IEEE Custom Integrated Circuits Conference*, pp. 7.2.1–7.2.4, 1988.

[32] A. Urbanski, P. Guebels, S. Shen, and J. Taeymans, "In Search for Better VLSI Design Method," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 416–419, 1985.

[33] D. Moldovan and A. Varma, "Design of Algorithmically–Specialized VLSI Devices," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 88–91, 1983.

[34] D. Thomas, E. Dirkes, and R. Walker, "The System Architect's Workbench," in *25th ACM/IEEE Design Automation Conference*, pp. 337–343, 1988.

[35] T. Lanfear, M. Fernando, and L. Wu, "Simulation of Signal Flow Graphs for Signal Processing Systems," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 1637–1640, 1985.

[36] W. Birmingham, A. Gupta, and D. Siewiorek, "The MICON System for Computer Design," in *26th ACM/IEEE Design Automation Conference*, pp. 135–135, 1989.

[37] K. Konstantinides, R. Kaneshiro, and J. Tani, "Scheduling and Task Allocation for Parallel Digital Signal Processing Architectures," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 2536–2539, 1989.

[38] M. Ercegovac and T. Lang, "On–Line Arithmetic: A Design Methodology and Applications in DSP," in *VLSI Signal Processing III*, (R. Brodersen and H. Moscovitz, eds.), IEEE Press, 1988.

[39] T. Chu, C. Leung, and T. Wanuga, "A Design Methodology for Concurrent VLSI Systems," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 407–410, 1985.

[40] T. Kuroda, H. Suzuki, H. Akiba, T. Aoki, T. Shigematsu, and K. Kawagai, "Unified Design Methodology and Device Architecture for Multi–Generation ASIC Applications," in *IEEE Custom Integrated Circuits Conference*, pp. 25.7.1–25.7.4, 1988.

[41] E. Lee, E. Goei, H. Heine, W. Ho, S. Bhattacharyya, J. Bier, and E. Guntvedt, "GABRIEL: A Design Environment for Programmable DSP's," in *26th ACM/IEEE Design Automation Conference*, pp. 141–146, 1989.

[42] K. Iwasaki, N. Yamaguchi, T. Shimura, Y. Hagiwara, T. Funabashi, and K. Minorikawa, "Design Methodology for Reconfigurable Module–Structured VLSI," in *IEEE Custom Integrated Circuits Conference*, pp. 464–467, 1985.

[43] J. L. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA.: Morgan and Kaufmann, 1990.

[44] M. McFarland, A. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," in *Proceedings of the IEEE*, pp. 301–318, 1990.

[45] D. Gajski and D. Thomas, "Introduction to Silicon Compilation," in *Silicon Compilation*, (D. Gajski, ed.), Reading, MA: Addison-Wesley, 1988.

[46] J. Allen, "Performance–Directed Synthesis of VLSI," in *Proceedings of the IEEE*, pp. 336–355, 1990.

[47] H. DeMan, F. Catthoor, G. Goossens, J. Vanhoof, J. V. Meerbergen, S. Note, and J. Huisken, "Architecture–Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms," in *Proceedings of the IEEE*, pp. 319–335, 1990.

[48] J. Rabaey, H. DeMan, J. Vanhoof, G. Goossens, and F. Catthoor, "CATHEDRAL II: A Synthesis System for Multiprocessor DSP Systems," in *Silicon Compilation*, (D. Gajski, ed.), Reading, MA: Addison-Wesley, 1988.

[49] B. Shung, R. Jain, K. Rimey, E. Wand, M. Srivastava, B. Richards, E. Lettang, K. Azim, L. Thon, P. Hilfinger, J. Rabaey, and R. Brodersen, "An Integrated CAD System for Algorithm–Specific IC Design," *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 447–463, April 1991.

[50] T. J. Kowalski, "The VLSI Design Automation Assistant: An Architecture Compiler," in *Silicon Compilation*, (D. Gajski, ed.), Reading, MA: Addison-Wesley, 1988.

[51] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated Annealing," *Science*, vol. 220–4598, pp. 671–680, May 1983.

[52] R. A. Rutenbar, "Simulated Annealing Algorithms: An Overview," *IEEE Circuits and Devices Magazine*, vol. 5–1, pp. 19–26, January 1989.

[53] P. H. Winston, *Artificial Intelligence.* Reading, MA.: Addison–Wesley, 1984.

[54] D. Bondurant, R. Cox, G. Deming, and D. Wick, "FFT Arithmetic Element Built on VLSI High Performance Gate Array," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 1477–1480, 1985.

[55] P. Duhamel and H. H'Mida, "New 2^n DCT Algorithm Suitable for VLSI Implementation," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 1805–1808, 1987.

[56] J. Moder and C. Philips, *Project Management with CPM and PERT.* Holland: Van Nostrand Reinhold., second ed., 1970.

[57] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing.* Englewood Cliff, NJ: Prentice Hall, 1975.

[58] S. Zohar, "Outline of a Fast Hardware Implementation of Winograd's DFT Algorithm," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 796–799, 1980.

[59] G. Allen, P. Denyer, and D. Renshaw, "A Bit Serial Linear Array DFT," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 41A.1.1–41A.1.4, 1984.

[60] C. Huang and F. Taylor, "High Speed DFT's Using Residue Numbers," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 238–242, 1980.

[61] Y. Zhang and Y. Yao, "Fast Implementation of Revursive DFT's," in *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, pp. 1071–1074, 1989.

[62] D. DeFatta, J. Lucas, and W. Hodgkiss, *Digital Signal Processing: A System Design Approach.* New York, NY.: John Wiley and Sons, 1988.

[63] A. Oppenheim and R. Schafer, *Digital Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1975.

[64] M. Takla, D. Bouldin, and D. Koch, "Fast Implementation of the Distributed Arithmetic Fourier Transform," in *Proceedings of the ASIC Conference*, pp. P3–7.1–P3–7.4, 1991.

[65] J. Cooley and J. Tukey, "An Algorithm for The Machine Calculation of Complex Fourier Series," in *Digital Filters and The Fast Fourier Transform*, (B. Liu, ed.), Halsted Press, 1975.

[66] N. Ahmed, T. Natrajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, pp. 90–93, January 1974.

[67] K. R. Rao and P. Yip, *Discrete Cosine Transform.* Academic Press, Inc., 1990.

[68] A. Netravali and B. Haskell, *Digital Pictures, Representations and Compression.* Plenum Press, New York, 1988.

[69] T. Rzeszewski and R. Pawelski, "Efficient Transmission of Digital Component Video," *SMPTE Journal*, pp. 889–898, September 1986.

[70] F. Justand, N. Demassieux, G. Concordel, J. Guichard, and E. Cassimatis, "A Single Chip Video Rate 16 × 16 Discrete Fourier Transform ," in *Proceedings of the International Conference on Accoustic, Speech and Signal Procsessing*, pp. 15.8.1–15.8.4, 1988.

[71] R. Jain, A. Parker, and N. Park, "Predicting Area–Time Tradeoffs for Pipelined Design," in *24th ACM/IEEE Design Automation Conference*, pp. 35–41, 1987.

VITA

Mourad B. Takla was born August 12, 1963, the son of Bushra S. Takla and Bedour B. Fahmy of Cairo, Egypt. He is married to Mary M. Takla. Mr. Takla graduated from Patriarcal School in 1980 and attended the Faculty of Engineering at Cairo University, Egypt. From 1980 until 1985 he was a recipient of the government fellowship for distinct students. In 1985 he graduated with an honor degree from Cairo University. After graduation, he joined the consulting office of Omar Seif El–Dien and Sons as a computer programmer.

In September 1986, he began his study toward a Master of Science in Electrical and Computer Engineering, at the University of Tennessee, Knoxville. He was a teaching and a research assistant since September 1986, the research being concentrated in the areas of VLSI, DSP system design, and Computer Vision . He received his Master of Science degree in December 1988 and Doctor of Philosophy degree in May 1992 both in Electrical and Computer Engineering at the University of Tennessee at Knoxville.