HIGH-LEVEL AREA-DELAY PREDICTION WITH APPLICATION TO BEHAVIORAL SYNTHESIS

BY

RAJIV JAIN

Technical Report CENG 89-23

July 1989

HIGH-LEVEL AREA-DELAY PREDICTION WITH APPLICATION TO BEHAVIORAL SYNTHESIS

by

Rajiv Jain

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Engineering)

July 1989

Dedication

To my parents and grandparents.

Acknowledgments

I take this opportunity to express my thanks to several people who have made this thesis possible. I convey my thanks to my advisor Prof. Alice Parker for her help and guidance in the making of this thesis. I would also like to thank Profs. Dean Jacobs and Sarma Sastry for being on my dissertation and guidance committees, and Profs. Aristides Requicha and V. K. Prasanna Kumar for being on my guidance committee.

Thanks are due to Profs. Melvin Breuer, Fadi Kurdahi, Nohbyung Park, C. S. Raghavendra, and J. Yee for their ready support.

I would like to thank my colleagues Esther Brotoatmodjo, Sally Hayati, Kayhan Küçükçakar, Dick Leubben, Mitch Mlinar, Shiv Prakash, Jagan Raghvendran, and Jorge Seidel for making the years at USC roll by unnoticeably. Also, I would like to thank Marion Rice for proof-reading my thesis and for making it presentable.

Two things which were very important to my research: computer games and cartoons. Even as I write this acknowledgment, I am playing xconq, backgammon and sopwith; they have kept me company on many occasions through the wee hours of the morning.

The person who really wanted to see me get a Ph. D. was my father. He, my mother and my wife have been a constant source of encouragement.

This research was supported by the Semiconductor Research Corporation (Contracts 86-01-075 and 88-DP-075), and the Departments of Air Force, Army and Navy (Contract N00039-87-C-0194). I would like to thank these organizations for the support.

Contents

List of Figures

List of Tables

Abstract

The automatic mapping of a behavioral description of a digital system into a register-transfer level design is known as high-level synthesis. High-level synthesis is computationally very expensive because a designer has to synthesize several designs and search the design space in order to find a good and acceptable implementation. One method of comparing two implementations is comparing their area-delay characteristics. The goal of this thesis is to accurately predict the lower-bound area-delay tradeoff curve of final implementations without synthesizing the designs. This helps the designer to narrow down the search in the design space and to find a satisfactory design quickly. The prediction mechanism developed here for pipelined and non-pipelined design styles is based on a lower bound area-delay theory.

The versatility of the area-delay model is demonstrated in this thesis by applying it to a variety of problems in high-level synthesis, such as pipelined and non-pipelined design style selection, module selection for pipelined designs and the evaluation of several data flow graph transformations including hierarchical decomposition. Experimental results using pipelined and non-pipelined synthesis tools support the theory.

This research is a part of the University of Southern California's ADAM Advanced Design AutoMation project.

Chapter 1

Introduction

"Begin at the beginning," the King said gravely,
"and go on till you come to the end: then stop."
—Lewis Carroll, *Alice in Wonderland*

Increasing VLSI design complexity and competitive market deadlines have led designers to search for advanced design tools to help produce correct designs quickly. The objective of automating the process of the design of digital systems is to be able to map onto silicon a design which meets the input specifications. The design should also satisfy any design requirements, called constraints, laid down by the designer. Automation may not produce the best possible designs (indeed, it is difficult, if not impossible, for human designers to do so). However it can produce acceptable and correct results quickly.

High-level synthesis is the mapping of a behavioral description of an algorithm onto a register-transfer level (RTL) design. Although the past few years has seen a proliferation of high-level design tools in research environments, none of these tools has gained general acceptance in industry. This thesis addresses some aspects of high-level synthesis. In the remainder of this chapter we present a brief introduction to high-level synthesis and a description of the research problem, followed by a global overview of the proposed solution technique. We conclude with a survey of the existing literature, and the thesis organization.

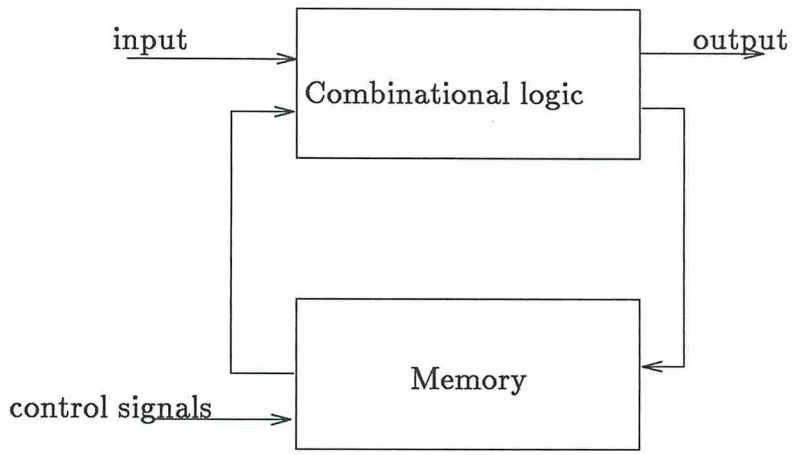


Figure 1.1: The RTL Data Path Model

1.1 Data Path Synthesis

High-level synthesis consists of two major tasks, data path synthesis and control synthesis. The output of a data path synthesis system is an RTL data path which meets the specified constraints and achieves a design goal. [1.1] An RTL data path consists of operators and registers interconnected via multiplexers, busses and wires. The RTL data path model is shown in Figure 1.1. Control synthesis refers to the automatic generation of a controller which provides the control signals for sequencing of events in the data path. This thesis focuses on the data path synthesis aspects. See [MPC88] for a tutorial on high-level synthesis.

The input to a typical data path synthesis system includes

1. a data flow graph or a control/data flow graph representing the behavior of the algorithm or a behavioral description of the algorithm in a hardware descriptive language,

2. a library of modules which implement the operations of the data flow graph,

[1.1] Design constraints specified for the high-level synthesis must be budgeted a priori for the control and data path. In this thesis we assume that such budgeting has been performed and henceforth design constraints are assumed to be applicable to data path synthesis alone.

3. designer specified constraints, and

4. goals for the design.

Following is a brief description of the inputs to a data path synthesis system.

A data flow graph is a directed graph with operations represented by nodes and values by arcs. For example, Figure 1.2 is the data flow graph of an AR filter element (referred to as AR filter in this thesis) [Kun84]. This graph represents an algorithm. The arcs impose an ordering on the execution of the nodes of the data flow graph for the correct execution of the algorithm [1.2]. If a hardware description language such as ISPS [Bar81] is used as an input, then the source code is translated into an internal data and/or control flow graph format [GBK85] [KP85] [McF78] [Sno78] [TN83] [Tri87].

A design library consists of module types which implement operations. For example, a ripple-carry adder is a module type which performs addition. A given operation may be implemented by more than one module type in the design library. For example, a 4-bit carry look-ahead adder and a ripple carry adder can both implement an addition operation. The library may also contain module parameters, such as area of the modules, which are used by the synthesis programs. Table 1.1 is an example design library. This library was generated by estimating actual areas using PLEST [KP86] which is a standard cell area prediction tool.

A data path is characterized by one or more of its parameters. The parameters most commonly considered are area and delay; however, other parameters such as power consumption or pin count may also be considered. The designer may choose to constrain the chip area of the data path design; for example, the chip area should not exceed 50,000 mil^2. Furthermore, the designer may require the data path synthesis tools to produce a design which meets this

[1.2] In Section 2.1.1 we will show how to convert an algorithm with loops and conditional branches into a directed acyclic graph.

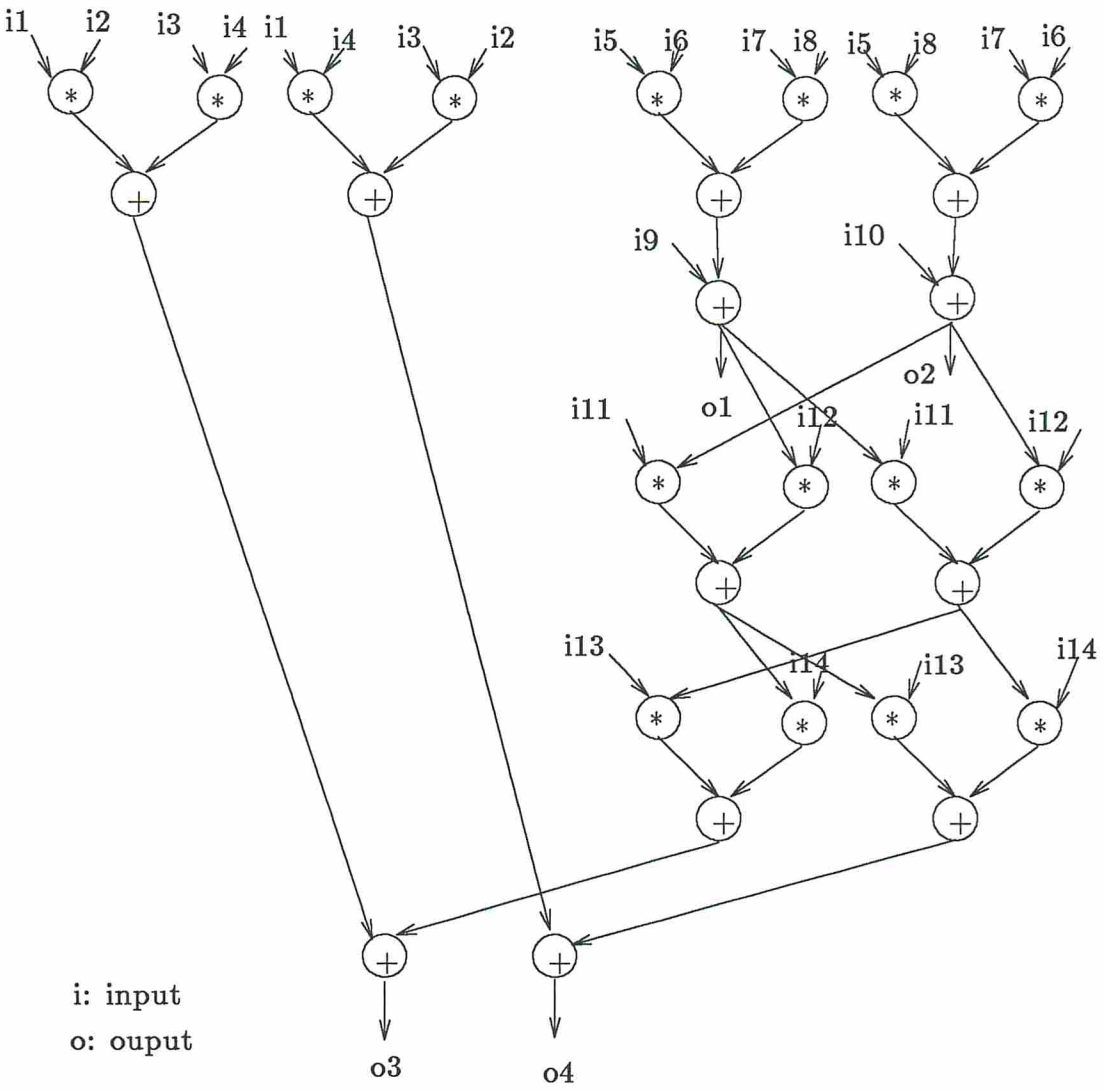


Figure 1.2: Data Flow Graph of AR Filter

Module Name	Operation	Bit Width	Area (A) mil^2	Delay (T) nS	$A \times T$ $10^3 mil^2 nS$
a1	addition	16	4200	340	1428
a2		16	2880	530	1526
a3		16	1200	1510	1812
s1	subtraction	16	4200	340	1428
s2		16	2880	530	1526
s3		16	1200	1510	1812
m1	multiplication	16	49000	375	18375
m2		16	9800	2950	28910
m3		16	7100	7370	52327
a4	addition	8	2000	225	450
m4	multiplication	8	13800	250	3450
d	distribute	*	0	0	-
j	join	*	0	0	-
sp	split	*	0	0	-
m	merge	*	0	0	-
a	assignment †	*	0	0	-
sr	shift right	per bit	72	5	.36
sl	shift left	per bit	72	5	.36
r	register	per bit	32	5	.160
mux	2:1 multiplexer	per bit	18	4	.072

* for all bit sizes

† area included in register area

Table 1.1: Design Library

design constraint and optimize some design goal. An example goal is to *minimize delay* while satisfying the design constraint of 50,000 mil^2.

Starting with the specified inputs, a data path synthesis system performs the following functions (not necessarily in the order given) to produce a "good" RTL data path meeting the specified design constraints:

1. module selection,

2. scheduling and partitioning,

3. module allocation,

4. module binding, and

5. register and multiplexer allocation and binding.

Module selection is the task of selecting the module types from the design library to implement the operations which occur in the data flow graph. The output of the module selection task is a module set which contains the module types. The selection must meet the constraints and attempt to optimize the design goal.

The assignment of each node in the data flow graph to the clock cycle in which it must be executed is called *scheduling.* Scheduling must preserve the data dependency requirements between nodes while trying to satisfy design constraints. Scheduling depends on various factors such as design constraints, module selection, module allocation, conditional branching, and resynchronization (rate of pipe flushing in pipelined designs). The interaction among these various tasks makes the problem of scheduling difficult. The general scheduling problem is known to be a NP-Complete problem [GJ79] and several heuristics exist to perform scheduling which produce good, though not necessarily optimal results [DLS81] [GBK85] [Gir87] [HPK87] [McF83] [NT86] [PG87] [Par85] [PPM86] [TS86].

Module allocation is the task of computing the number of each module type required for a given implementation. The various module parameters, design constraints and scheduling must be considered for optimal module allocation.

Operation module binding is assigning modules to data flow operation nodes. In executing two addition operations in the same time step using two adders, the decision as to which addition operation is performed in which adder module is module binding. The task of optimal module binding is dependent upon scheduling, module allocation, interconnect costs and delays.

The task of *register and multiplexer allocation* is to assign data values to (from) registers and route them through multiplexers or busses from (to) the modules. Several programs which perform module binding and register and multiplexer allocation with different allocation philosophies exist [BG87] [KP] [McF81] [PK] [TS86].

Several of the synthesis functions described above are known to be computationally expensive problems. For a globally optimal solution several factors such as design time, pipeline complexity, cost, available resources, performance constraints, and concurrent operations must be taken into account. Furthermore, the interaction of the synthesis functions with each other makes data path optimization even harder. For example, in pipeline designs the shortest schedule does not guarantee the smallest delay, since module conflicts between consecutive tasks may reduce the initiation rate [PP88]. Also, satisfying the design constraints complicates data path synthesis even more. Finally, all the five synthesis steps need to be performed concurrently for a globally optimal design. The computational complexity of the combined five tasks is intractable and is thought to be NP-hard. Consequently, practical data path synthesis systems use heuristics to generate non-optimal but "acceptably good" designs.

For a given input (behavioral description), a data path synthesizer can produce a variety of candidate RTL designs. However, not all designs are of

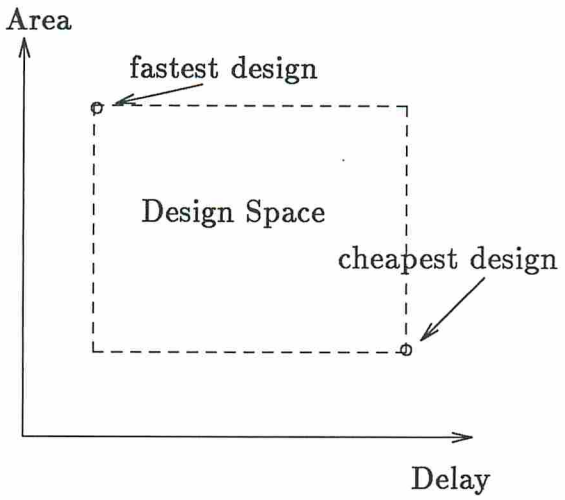


Figure 1.3: An Example Design Space

equal quality. [1.3] Designs can be characterized by any measurable quantity such as area, delay, power or design time. An example characterization of an RTL design is the area-delay characteristic of the design. In this case, all designs lie in a *design space*, which is a 2-dimensional plane with area and delay as its two axes. [1.4] A *design point* represents a design in this space. In a 2-dimensional design space with area and delay as the axes, the cheapest (least area) and the fastest designs delimit the design space boundary for a given data flow graph and a given module set. Joining all non-inferior design points produced by the data path synthesizer gives an area-delay tradeoff curve for the input. Figure 1.3 shows the optimal design space bounded by the fastest and the cheapest designs. A plot of the area versus delay of the non-inferior design points gives an area-delay tradeoff curve for the designs which implement the given behavior. Figure 1.4 shows the area-delay characteristics of the several designs synthesized by Sehwa, a pipeline synthesis program, from the data flow graph shown in Figure 1.2.

[1.3] A design implementation is inferior when there exists at least one other implementation which performs better in one or more figures of merit, all other figures of merit being at least equal.

[1.4] The design space is not restricted to area and delay alone. A dimension could be any measurable quantity which is important to the designer.

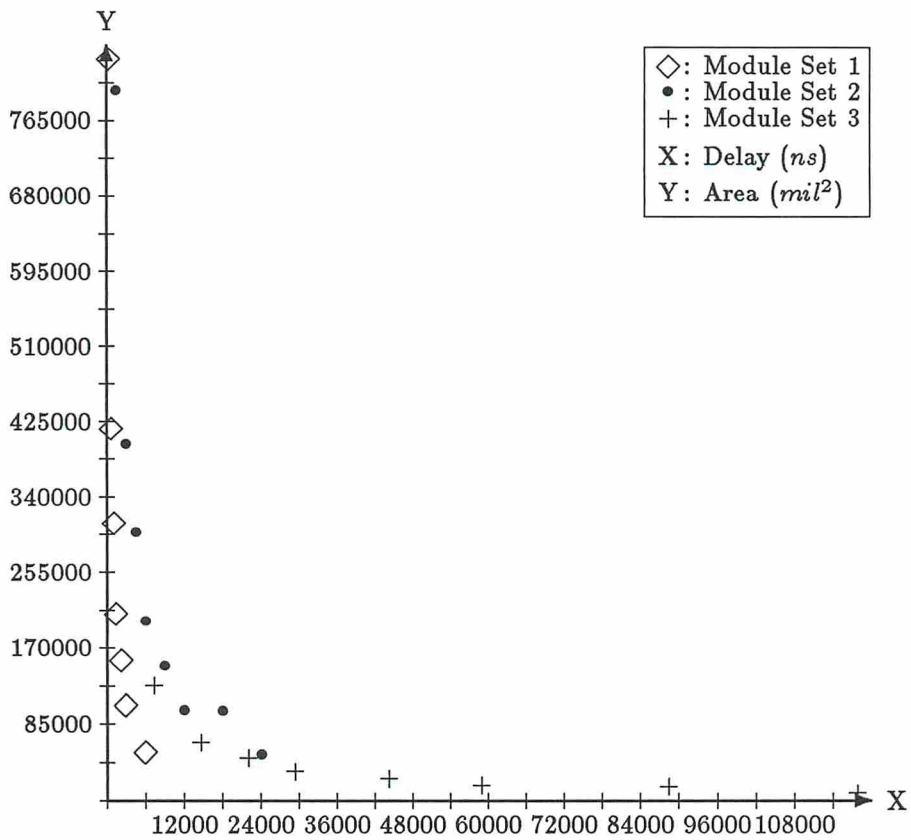


Figure 1.4: Designs Produced by Sehwa Using Several Module Sets

1.1.1 High-Level Design Decisions

A data path synthesis tool accepts an input behavioral description and produces an RTL design. In order to generate area-delay efficient designs, however, designers make high-level design decisions before synthesis [ASU86] [MR85] [SJB*88] [Sno78] [Tri87] [Wal88]. These high-level design decisions alter the input description without changing the behavior; i.e. a transformation is applied to the input description which generates another functionally equivalent behavioral description. For example, dead code elimination is a high-level design decision which eliminates redundancy and produces a more efficient description. Other examples of high-level design decisions are tree height reduction, loop unwinding and loop winding [Gir87], choice of operation bit-width [HJ88] [RD87] [YJHN87], hierarchical decomposition of operations into sub-operations, design style selection [Tho77], choice of arithmetic (for example, 2's complement vs. signed magnitude) [SJB*88], and choice of a number system.

High-level design decisions may impact the final RTL implementation in various ways. While some design decisions may lead to better implementations, others may have a negative effect on design quality. Furthermore, some decisions may produce a behavioral description which does not satisfy design constraints. No existing analytical method predicts the effects of these high-level decisions on the final design without performing the synthesis process.

Making correct high-level design decisions is important. An erroneous decision may result in a poor design despite lower-level efforts. An example: decomposing operations into sub-operations may be valuable until a certain level is reached as it introduces more scope for parallelism and sharing of resources. Beyond that level the cost and delay of the steering logic may outweigh the benefits obtained. Similarly, the decision to implement an input behavior with a pipelined or non-pipelined design style is made before synthesis begins, and is instrumental in meeting performance constraints. For example, a non-pipelined RTL design may not match the specified throughput requirements while a pipelined design style could.

1.2 Functional Pipelining

Pipelining is a desirable method for designing circuits with large throughput. In pipelining, each unit computation task (e.g. a microinstruction) is partitioned into a sequence of subtasks, each of which is executed during a single clock cycle. Every clock cycle has the same time period. Initiations of consecutive tasks occur at some fixed or variable interval, called the initiation interval which is the number of clock cycles between initiations of two successive tasks. In this fashion, execution of subtasks of consecutive tasks may overlap in time on different parts of the pipeline circuit. The pipeline strategy used in this thesis is actually more complex than conventional pipelining. Unlike conventional pipelining, in our case a module can be shared amongst different stages of the pipeline. Within a stage, some modules might be shared differently from others. An adder might be shared by stages 1 and 3 of a pipe, while a multiplier is shared by stages 1 and 4. Thus, there is no physical *stage* which corresponds to the logical grouping of operations in a time step. This is called *functional pipelining.* For further details see [PP88].

1.3 The Need for Area-Delay Prediction

Let us assume that high-level design decisions have been made and the designer now has a behavioral description for which a data path is to be synthesized. The main difficulty which the designer encounters at this stage is that of selecting the design which best meets goals and constraints from the numerous possible designs. In order to arrive at a satisfactory design, the designer usually iterates on the design process by trading off the amount of parallelism and resources, varying the clock cycle, selecting different module sets and module allocations, and generating different schedules. The number of iterations required to synthesize every possibility is prohibitively large. An area-delay predictor which can predict the area-delay parameters of the final design space without actually synthesizing the designs can significantly reduce the number of trial

passes and the consequently design time. Thus, this predictor can be used by automatic synthesis programs or by designers to narrow the search space and arrive at a satisfactory design quickly.

In this thesis we develop lower-bound area-delay predictors. Given a behavioral description and a module set, this predictor produces a tradeoff curve. All design points lying on this curve are *module-optimal* [1.5]. Furthermore, there is no design point between the lower-bound curve and the axes. All design points lie on or above the curve and hence the attribute "lower-bound". In this thesis area-delay refers to lower-bound area-delay, unless otherwise stated.

A lower-bound area-delay predictor provides an independent performance measure for comparing different synthesis systems. In the past, most synthesis systems were compared using examples. However, there has been no absolute standard by which to judge performance of these programs; exhaustive search to find optimal designs is impractical. Using this basic prediction tool, synthesis systems can quantify the quality of the designs they produce against a known lower bound.

The area-delay predictor can also be used by the designer in making high-level design decisions thereby producing behavioral descriptions which have efficient RTL implementations. The benefits of high-level decisions are obvious for some transformations, like elimination of dead code and expression compaction, and are not so obvious for many other design decisions. It is computationally very expensive to perform synthesis to obtain actual design characteristics [Par85]. Every high-level design decision creates a different (functionally equivalent) behavioral description for the synthesis process. In order to make wise high-level decisions we must be able to predict the impact of the changes in the input algorithm on cost, speed, and other characteristics of the resultant RTL design. An accurate and fast predictor which could predict area-delay characteristics of a final RTL design without actually synthesizing one would be helpful in making these high-level design decisions. The designer could then perform

[1.5] A module optimal design is one in which every module is utilized every clock cycle. It is defined in Section 2.1.3.

a transformation on the behavioral description and observe the changes of this transformation on the design without actually performing the synthesis process. With this capability the decision to accept or reject the transformation is made easily.

These area-delay models are not limited to high-level synthesis alone. They can be used in a variety of architectural level decisions such as selecting appropriate algorithms or evaluating the performance of a chip, and computer architects can use these tools in making design style selections.

To summarize, in this thesis we develop an area-delay predictor and demonstrate its usefulness by using it to solve three problems, namely, module selection, design style selection and evaluation of algorithm transformations.

1.3.1 The ADAM Data Path Synthesis System

The tools developed in this thesis are a part of the ADAM synthesis system being developed at the University of Southern California.

The ADAM data path synthesis system consists of two major subsystems; namely, the program tools which synthesize RTL designs from a behavioral description (Figure 1.5) and the prediction tools (Figure 1.6) which guide the designer in exploring the design space for a good design. Three inputs are required by the data path synthesis programs: a data flow graph representing the behavioral description, a design library, and a set of design constraints. The data flow graph may contain loops and conditional branches. The synthesis system accepts an absolute value of area or delay as a design constraint. If an area constraint is given on the design, then delay is minimized within the limits imposed by the area requirements; if a delay constraint is given, then, the area is minimized as much as possible for the given delay. The prediction and synthesis subsystems are used by the designer in conjunction with each other to arrive at a satisfactory design. Further details on design space exploration using the two ADAM subsystems are available in [JKMP89].

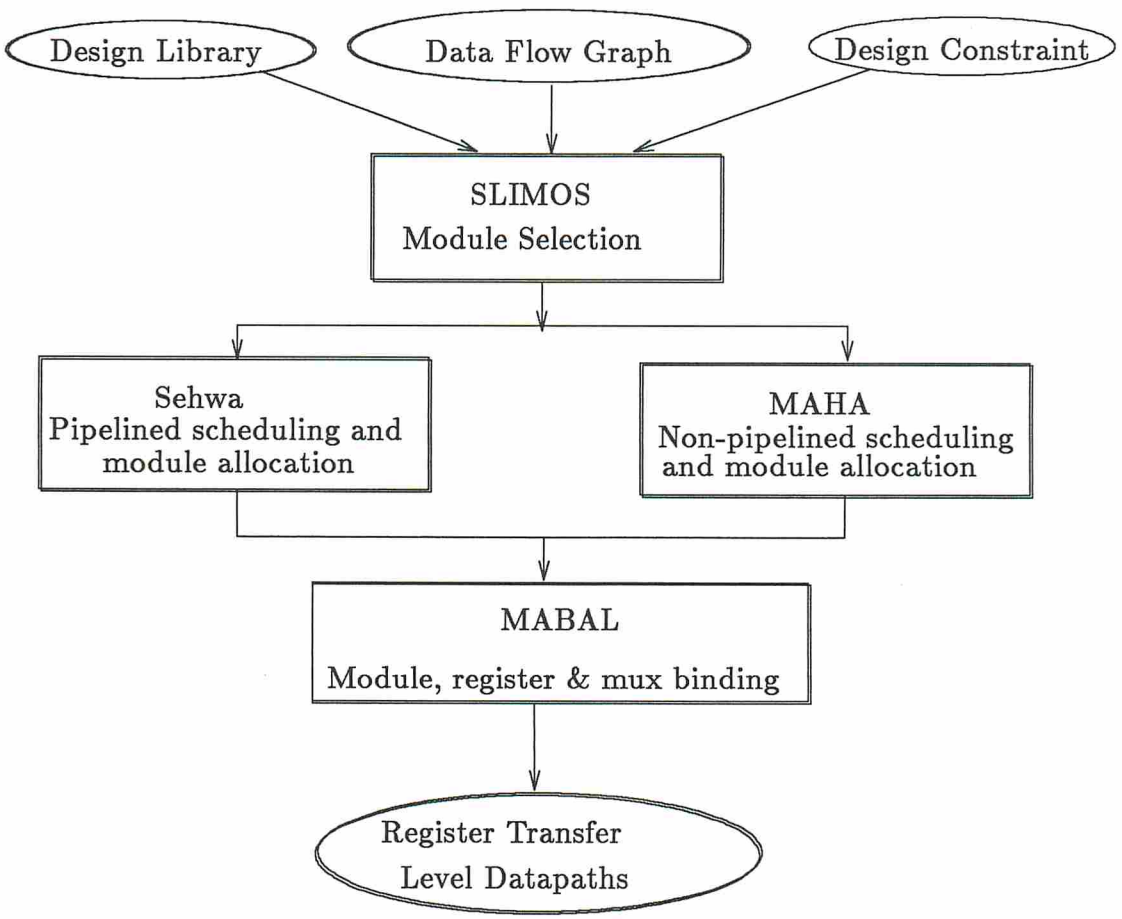


Figure 1.5: Data Path Synthesis in the ADAM System

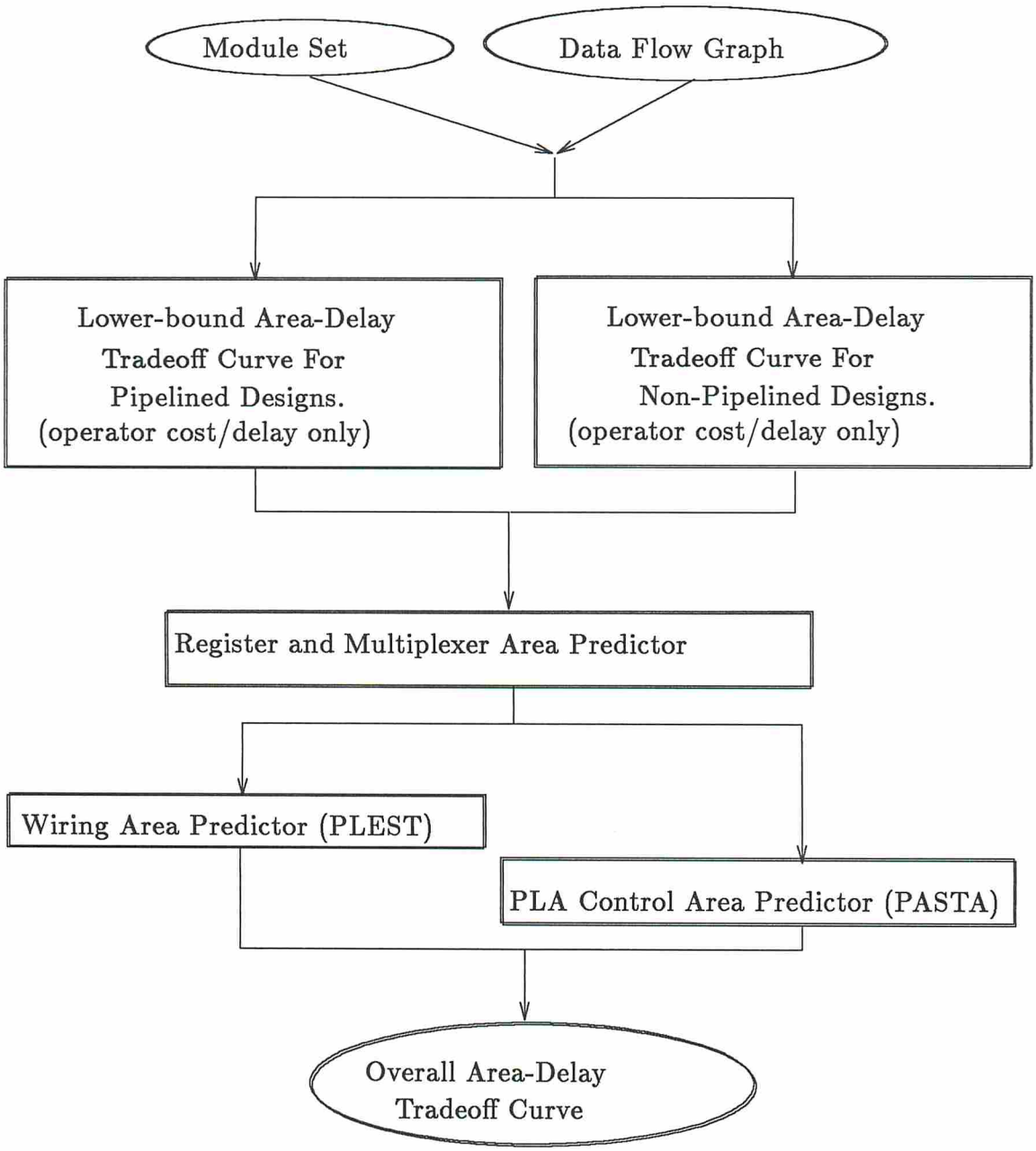


Figure 1.6: Area-Delay Predictors in the ADAM Synthesis System

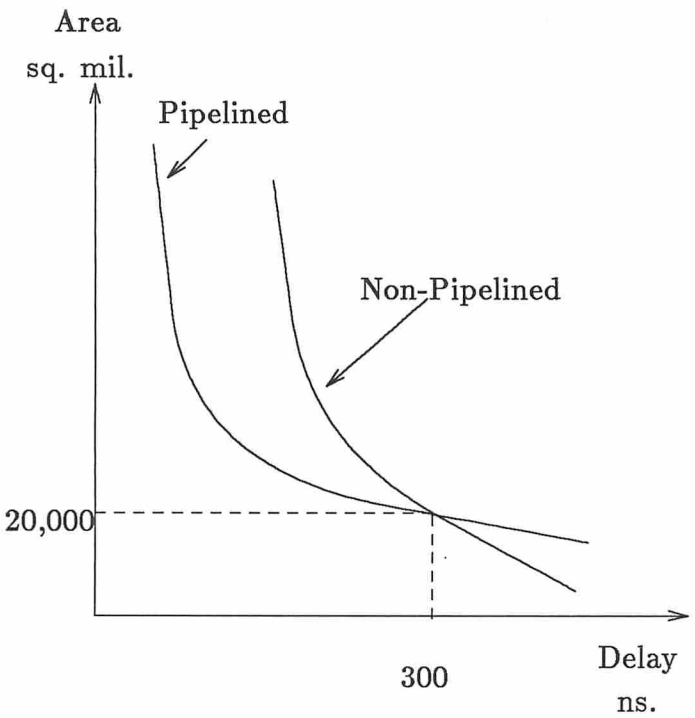


Figure 1.7: Area-Delay Curves For Pipelined and Non-Pipelined Design Styles

design style should be chosen for designs constrained to be less than 20,000 mil^2; otherwise the pipelined design style should be selected.

The area-delay model for pipelined design style predicts the lower-bound area and the lower-bound initiation delay of the pipeline. The initiation delay of the pipeline is the delay between two successive data inputs to the pipeline. The area-delay model for non-pipelined design style predicts the lower-bound area (same as for the pipelined design style) and lower-bound circuit delay of the design. Circuit delay is the time required by the design to process one set of input data, and is different from the initiation delay. Since the delay measures for the two design styles are different a common platform for comparison between the two design styles is first developed, and then design style selection is performed.

1.4.3 Data Flow Graph Transformations

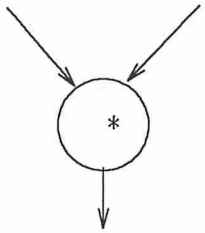
Yet another application of the area-delay predictors is in predicting the impact of data flow graph transformations. The problem of evaluating data flow graph transformation is as follows. *What is the impact of the data flow graph transformation on the design characteristics of the RTL design?*

We propose the following methodology for analyzing the impact data flow graph transformations on the area-delay characteristics of the final design. The method is based on the lower-bound area-delay model. Using these predictors one will be able to examine the effect of each transformation on the design without performing synthesis. We can predict the direction that the design curve will move in the design space and quantify the movement. Although the technique is quite general and can be used to evaluate various transformations, we will restrict discussion to transformations such as tree height reduction and change in node count. In particular we will show the effect of hierarchical decomposition on each design style. The evaluation is restricted to area and delay characteristics.

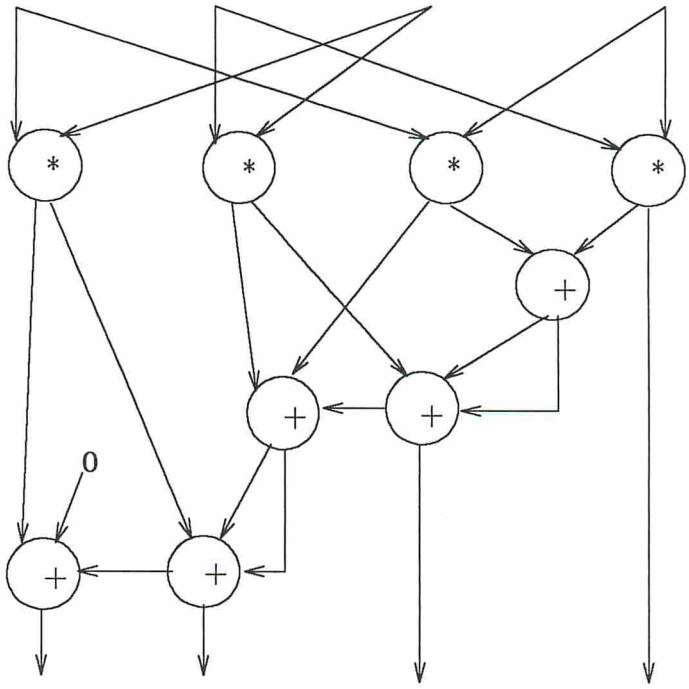
Hierarchical Decomposition: Operation Decomposition is an important design technique and has several uses. Decomposition may lead to more opportunities for better resource sharing and a higher degree of pipelining. Furthermore, decomposing a slow operation into smaller and faster operations may increase the throughput of a pipeline. Other reasons for using hierarchy such as abstracting operations, or to defer implementation details till later in the design process, are not discussed in this thesis. As a subproblem of the data flow graph transformation analysis we analyze the problem of hierarchical decomposition of operations into sub-operations. Figure 1.8 describes decomposition of a 16-bit multiplication node into 8-bit operations.

1.4.4 Module Selection

Area-delay predictors can also be used to solve the module selection problem. During the process of synthesizing an RTL design, a designer or an automated synthesis tool needs to select appropriate module types from the design



(a) 16-bit Multiplication



(b) 16-bit Multiplication Using 8-bit Multipliers

Figure 1.8: Example of Operation Decomposition

library to be used for implementation. The input to module selection consists of (i) a behavioral description; (ii) a design library, and (iii) a cost or delay constraint. Using these inputs, module selection system chooses appropriate module types from the library for implementation. In this thesis we solve the module selection problem for the pipelined design style only.

A *module set* is a subset of the design library, and satisfies the following conditions: (i) a module set contains modules needed to perform operations in the given data flow graph, and (ii) a module set is constrained in that a given operation may be implemented by only one module[1.7].

Table 1.1 shows a sample design library which is used in the experiments in this thesis. A module set which can be selected from this library for the data flow graph of Figure 1.2 is (a_1, m_1). (a_1, a_2, m_1) is an invalid module set since it contains two adder modules. Likewise, module set (m_1) is invalid because it does not contain an adder module.

The module selection problem is combinatorial in nature. For example, assume that a data path requires an adder module and a multiplier module, and the design library contains p_1 different types of adders and p_2 different types of multipliers. An exhaustive search for an appropriate module set would result in $p_1 \times p_2$ different combinations of module sets. Of these, some module sets will lead to inferior designs. Using an area-delay predictor for pipelined designs we have shown that a maximum of $p_1 + p_2$ module sets are sufficient to cover the optimal surface of the pipelined design space, reducing thereby the number of choices considerably. Also, these $p_1 + p_2$ module sets can be easily identified by applying the area-delay theory. Further, the designer (or the automated synthesis tool) can select the module set which best meets his design goals prior to performing design synthesis.

The design space for the data flow graph of Figure 1.2 using three different module sets is shown in Figure 1.4. The optimal surface of the design space is bounded by the fastest and the cheapest design points generated for

[1.7] This is not really a constraint as we shall see in Section 3.3.2.

the library. There are three design curves; one for each module set. Every design point on a curve corresponds to an actual design produced by Sehwa with different values of initiation interval. The design space covered by Sehwa for design exploration for a good design is 834,400 mil^2 to 8,300 mil^2 in area and 375 ns to 117,920 ns in time. If prior selection had not been made and only one module set (say module set 2) was used for searching the design space, Sehwa would have covered a much smaller design space. The module selection program described in this thesis explores the larger design space and selects an optimal module set meeting design constraints.

Module selection also impacts the scheduling of a data flow graph. Suppose a scheduler generates an optimal schedule using module set ms_1 for a data flow graph. After the scheduling has completed let us replace module set ms_1, with module set ms_2. This new module set ms_2 may make the schedule (which was optimal for module set ms_1) non-optimal. Thus, selecting module sets prior to scheduling is important. By using the module selection program prior to scheduling we ensure the best possible scheduling of the data flow graph.

1.4.5 Summary of Proposed Research

To summarize, we propose solutions to the following problems:

1. Lower-bound area-delay prediction for pipelined designs (Chapter 2).

2. Module selection for pipelined designs (Chapter 3).

3. Lower-bound area-delay prediction for non-pipelined designs (Chapter 4).

4. Design style selection (Chapter 5).

5. Evaluation of data flow graph transformations on pipelined and non-pipelined design styles (Chapter 6).

1.5 Literature Survey

Related research as reported in the literature will be divided into four basic areas: (i) area-delay prediction, (ii) design style selection, (iii) data flow graph transformations, and (iv) module selection.

1.5.1 Area-Delay Prediction

Although many synthesis systems exist (see [MPC88] for a partial list of existing synthesis systems), none of the published systems uses an area-delay prediction model as a front end to prune inferior design space. This missing feature forces the designer to partially or completely generate a design before quality can be assessed.

Early research on area-delay prediction for pipelined designs was published by Davio, et al. in 1983 [DDT83]. One major assumption made by Davio was that all nodes of the input data flow graph were identical, i.e. they had the same delay, and area, and performed the same function. This assumption simplified the problem since clock cycle time was readily determined. With the model we present in this thesis we can solve problems in which data flow graphs contain nodes of one or more operation types.

Kurdahi [Kur87] has empirically derived the area-delay tradeoff curves of non-pipelined adders and multipliers. He used PLEST [KP86] to estimate the total area (including wiring and functional area) of adder modules and found the shape of the curves to be $AT^{.825}$, $AT^{.841}$ and $AT^{.983}$ for 16-bit, 8-bit and 4-bit adders respectively. He also derived upper-bound register and multiplexer areas [Kur87] and proved that an upper-bound register count is given by the maximum cut of the data flow graph which can be computed by any *max-flow, min-cut* algorithm (like Dinic's algorithm [Eve79]).

Recently, McFarland reported several area-delay curves generated by BUD [McF87] for non-pipelined designs. Experimental curves depicted the design space considering only module area-delay. Then register, multiplexer and bus

area-delay were added, and finally, wiring area-delay was added. The curve shown in Figure 3e in [McF87] considers the operator area and delay alone. The shape of this curve is the same as that predicted in this thesis.

1.5.2 Design Style Selection

A significant work in the area of high-level design issues is by Thomas [Tho77]. In this work three design styles were compared. The three design styles were microprocessor based designs, designs using TTL components and bus-architecture designs. This work is very general, and mainly experimental, and several important issues are studied. Although [Tho77] does provide the results of using different design styles, design style selection is not done prior to synthesis, using the input behavior. Thomas uses a scheduled input description and then estimates the cost of synthesizing the description in different design styles. The final decision of the selection procedure rests on producing one design for each design style and choosing the best one. In this thesis, we start with an unscheduled input description and compare the entire design space to make the final design style selection. Thomas identifies pipelining, and bit-slice microprocessor design styles as well.

The choice of the non-pipelined or pipelined architecture design style is also discussed in the Systems Architect Workbench developed at CMU [TDW*88]. This system provides the designer with the capability of synthesizing microprocessor based, pipelined or non-pipelined architectures. The design style decision is made by the designer on the basis of the designs produced at the workbench. In order to compare several different designs, the whole process is iterated. A further restriction is that as the designer has to manually partition the input description into different stages for pipelined designs [Wal88].

Three application specific silicon compilers CATHEDRAL I, CATHEDRAL II [DRSC86] and the CATHEDRAL III [RD87] have been built at the IMEC, Leuven. These silicon compilers are specific to digital signal processing applications. CATHEDRAL I produces hardwired bit-serial architectures aimed

at medium sample rates. CATHEDRAL II aims at producing a customized microcoded architecture with low sample rates, and CATHEDRAL III produces a fast hardwired bit-sliced architecture which aims at medium to high sample rates. For high sample rates a new CATHEDRAL IV has been recently announced which generates hardwired bit-sliced architectures. The decision to use one of the CATHEDRALs lies with the designer. These compilers have the usual disadvantages in that the final decision of the selection procedure rests on producing one design for each design style and choosing between design styles. The CATHEDRAL compilers are, however among the first synthesis tools which make several high-level arithmetic tradeoffs, like word length selection.

Flamel [Tri87] makes several high-level tradeoffs like tree height reduction. The program does not synthesize pipelined architectures or address the problem of design style selection.

1.5.3 Data Flow Graph Transformations

There are several research articles which suggest numerous data flow graph transformations [ASU86] [Sno78] [Tri85] [Wal88]. Many of these transformations are taken from compiler optimization theory. None of these articles evaluate the impact of the transformations on the final design because none of them have a model which captures the area-delay characteristics of the final design. In this thesis we have evaluated the impact of some of the data flow graph transformations described in these articles.

Hierarchy for high-level synthesis does not show up in the literature except in circuit design [BTF83]. Palladio is a program which performs incremental refinement of function, with periodic validation by simulations. It decomposes components at one level into sub-components at that level or a lower level. The components may be functional (i.e. representing behavior) or structural. Refinement of a component continues till the component can be laid down in silicon.

1.5.4 Module Selection

In the past, selection of module styles or types has been viewed as a function to be performed after data path scheduling and allocation had been completed [Lei81] [McF81] [PK87] [TL83] [TS86]. The synthesis programs determined how many of each operator were required, but the specific operator implementation (e.g. carry-look-ahead vs. ripple-carry adder) was not decided until scheduling and operator allocation was complete. This procedure did not restrict the design space severely, since most synthesis programs assumed that each operation take one time step. This assumption was made regardless of the function being performed or how it was implemented. Thus, scheduling of operations could proceed independently of module selection.

In the last several years, however, data path synthesis programs have removed the simplifying assumption that each operation takes a single time step (see [BG87] [Gir87] [McF86] [PP88] [PPM86]). Today, state-of-the-art programs schedule multiple operations chained together into single time steps, and achieve designs which are faster and more area efficient. In order to perform this more complex scheduling, some information about actual module delays is required. In addition, many synthesis programs use even more sophisticated cost measures during data path synthesis, such as chip area, which requires knowledge of the actual cell area and interconnect costs.

In order to provide this additional information to the scheduler, module selection must be done before scheduling. BUD [McF86] was one of the earliest synthesis programs to accomplish this. BUD uses a straightforward heuristic which selects the module with minimum $area \times delay^n$ product, where *area* (*delay*) is the area (delay) of each module and n is a variable set by the designer.

Leive [Lei81] has proposed a solution to the general module selection problem for non-pipelined designs. After performing module allocation and scheduling, Leive performs module binding. Each module is evaluated for the *goodness* of implementing node k of the data flow graph. Finally the module with maximum *goodness* is selected to implement that node. The value of *good-*

ness is a function of weighted area, delay and power of the node; these weights are designer defined. As we shall see in Section 3.4, optimizing the module style for every node does not necessarily lead to a globally optimal result for pipelined designs.

Foo and Kobayashi [FK86] have adopted an approach similar to [Lei81]. They use a rule based system to solve the problem. Again, local optimization of individual modules is performed rather than optimizing the overall design.

In [HP83], scheduling, allocation, module binding and module selection problems are solved concurrently for non-pipelined designs. This approach employs a mixed integer-linear programming (MILP) technique for the solution of non-pipelined designs. Constraint equations are derived from the data flow graph and modules in the library. The basic drawback of this technique is that the solution entails enormous computer runtime and is not practical for realistic examples. Also, the module selection problem by itself is not independently solved, thus forcing the designer to solve the whole problem of data path synthesis. Optimal results are obtained, however.

In the remainder of the existing literature, the problem of module selection has been simplified. The programs use either a fixed predetermined module set [Gir84], or they specify that only ALU's can implement the operation [DN87] [SLP88] [TS86]. The cited research is restricted to non-pipelined designs. We believe that our work is the first effort in formalizing and proposing a solution to the module selection problem for pipelined designs.

Girczyc and Knight [Gir84] start with a fixed library of cells which implement every operation type. In their library, there is exactly one type of module which can perform a given operation, so, the problem of module selection does not arise.

In [TS86], Tseng and Siewiorek have tried a different approach to the module selection problem. In their paper, sets of operation which are *compatible* (compatibility is determined using the clique-partitioning algorithm) are formed. Then, all operations in a set are implemented by the same module. The selection of a module from a library of candidates is not involved.

MAHA [PPM86] is another non-pipelined synthesis program in which if modules have not been preselected, the parameters of all the modules which implement the same operation type are averaged. That is, if the library contains three different adders, then the average delay and the average cost of the three modules is taken as the module parameters for the adder module.

In HAL [PK87] the module selection problem has been avoided by assuming that every module has a delay equal to some integer multiple of the designer specified clock cycle. This implies that the module set is constrained to contain modules having a fixed delay.

1.6 Thesis Organization

The thesis is organized as follows. In Chapter 2 the lower bound area-delay theory for pipelining is developed. The theory predicts lower bound area-delay curve can be produced for the input behavior and a given module set. Results of some experiments which were conducted to verify the theory are also given. Using the lower bound area-delay theory, the model and solution to the module selection problem for pipelined designs is given in Chapter 3. The chapter concludes with the experimental verification of the model and the theory. We derive the lower bound area-delay model for non-pipelined designs in Chapter 4. Experimental verification using a non-pipelined synthesis tool supports our model. A solution to the design style selection problem is given in Chapter 5. In Chapter 6 we present our analysis of the impact of the data flow graph transformations on pipelined and non-pipelined design styles. In particular we will detail the analysis for hierarchical decomposition. Results from several experiments are also given. Conclusions and future research directions are described in Chapter 7.

Appendix A summarizes the notation used in the thesis.

Chapter 2

Lower Bound Area-Delay Tradeoff Curve for Pipelined Designs

2.1 Introduction

In this chapter we derive a lower-bound area-delay model for pipelined designs. The model accepts a data flow graph and a module set. It produces an area-delay tradeoff curve of the module-optimal designs. A tighter lower-bound area-delay tradeoff curve can be generated from the model using an algorithmic procedure. After stating the preprocessing steps, the assumptions and the optimality criterion, the model is developed in two stages: first the best possible clock cycle for module-optimal designs is computed and then the lower-bound area-delay tradeoff curve is derived. This chapter concludes with the results of some experiments conducted to verify the model. Having developed the model for a fixed module set, the effect of different module sets on the area-delay curves of the design can be examined to select the best module set (Chapter 3).

The area-delay model predicts the design points in the design space for module-optimal designs. The lower-bound area-delay curve is estimated to be $AT_p = k$ where A is the total functional area of the design, $T_p = l \times c_p$ is the

initiation delay of the design (i.e. delay between two successive data inputs to the design), l is the initiation interval of the design, c_p is the clock cycle of the design, and k is a constant [2.1].

2.1.1 Preparing the Input Data

In this section we list the preprocessing operations that may have to be performed on the input data flow graph before the area-delay curve predictor can be used. These preprocessing steps do not alter the area-delay characteristics, but map the input data flow graph to a form acceptable to the predictor. We note that these steps prepare a general input specification so that it is acceptable to the ADAM synthesis system.

2.1.1.1 Representing Loops

The input description to our prediction tool is a data flow graph which represents an algorithm. It is a directed acyclic graph with conditional branches (represented by *dist* and *join* node pairs). In general, the algorithm itself has loops. All loop arcs are removed prior to synthesis or area-delay prediction. Loop breaking methods have been suggested in [BCM*88] [MP88a] [Par85] [Pau88] [Tri87]. Here we briefly outline the ideas behind these methods. Loops with determinate iteration counts can be unrolled easily by duplicating the loop as many times as the iteration count [Par85] [Tri87]. Loops with unknown iteration count, or with conditional iteration counts are broken using one of several existing methods.

One method of breaking loops is described in [Pau88]. In [Pau88], each loop has a timing constraint which specifies the maximum delay of the loop. The loop present in the algorithm is extracted from the data flow graph and scheduled independently, and a node with a delay equal to the timing constraint is substituted in the original data flow graph. Our model can handle this method of

[2.1] Explanation of notation is provided in Appendix A.

breaking loops. Similar techniques are used in [Gir84] and [BCM*88] to generate a directed acyclic data flow graph.

Another method of breaking loops is shown in [MP88b]. With this method, a loop is broken such that the critical path of the loop is inserted in-line into the data flow graph. Yet another method is to move the inner loops of the algorithm to the outer body of the algorithm.

The current version of CSTEP [Nes87, page 61] does not allow loops in its input description. Several synthesis systems do not discuss the issue of loops or describe how they are handled ([McF86] [PG87] [TS86]).

The results from different synthesis systems which use different loop breaking strategies demonstrate the independence of the prediction tool to the way a loop is broken.

2.1.1.2 Handling Multi-Cycle Operations

The ADAM synthesis programs assume that an operation executes within one clock cycle. That is, a single operation cannot be scheduled into two or more time steps. However, the design library may contain multi-cycle or pipelined modules such as multi-cycle or pipelined multipliers. In this case we decompose the operation which occurs in the data flow graph into sub-operations so that each sub-operation is executable within one clock cycle. For example, any operation *foo* which can only be executed over two clock cycles can be decomposed into *foo1* and *foo2* each of which is executable in one clock cycle. Thus, in the predictive model we can assume that multi-cycle operations have already been decomposed.

If the delay of some module types is such that several operations implemented with those module types can be performed sequentially in the same clock cycle, then these operations are assumed to be chained (scheduled in the same time-step).

2.1.1.3 Handling Arbitrary Module Sets

Prior to area-delay prediction, an operator type (module type) must be selected for every operation type in the data flow graph. This must also occur prior to scheduling since adequate scheduling cannot be performed unless operator delays are known. The ADAM synthesis programs assume that a module set has been selected which contains exactly one module type for a given operation type. If, however, operations of the same type occurring in several places are to be implemented using different module types which perform the same operation (say, addition), the instances of the operation type are replaced by two or more distinct operation types (say addition1 and addition2), each corresponding to a separate module type and performing the same operation.

If, in the data flow graph, several operation types are to be implemented using a single module type (for example, add, and subtract are to be implemented using an ALU module type), then all these operation types are replaced by the single operation type which corresponds to the module type.

The prediction technique can be used with different input module sets and the best module set can then be selected for implementation. For example, if the design library contained two different types of adders and three different types of multipliers, then area-delay curves using all possible (six) combinations of adders and multipliers could be predicted and the combination which yielded the best design could be selected as the best module set. Thus, area-delay curves using any desired module set could be predicted. Obviously, more clever techniques than exhaustive generation of all possibilities must be used for practical designs.

2.1.2 Assumptions

The area-delay model is limited by the following assumptions. These assumptions have allowed us to develop a mathematical model for the area-delay tradeoff curve.

Resynchronization (flushing the pipeline) does not occur. The two main reasons for resynchronization are exception conditions and conditional branches. Several applications including signal processing applications seldom have conditional branches, and hence resynchronization due to conditional branching is rare. Furthermore, as the data is normalized to meet the pipeline's arithmetic precision capabilities, exception conditions due to arithmetic errors are reduced. (See [Kun84] for examples.) This assumption does not affect our results as resynchronization serves only to increase the delay for the same design area and such designs lie above our predicted lower bounds.

Static scheduling of pipelines is assumed. Considering the merits of static scheduling and dynamic scheduling outlined in [Par85], this assumption is reasonable.

2.1.3 Definitions

Before explaining how to compute this curve, we provide the following definitions of utilization and module-optimality. The underlying concept of module-optimality has been taken from [Gir84] and [PP88].

Definition 2.1 *The number of clock cycles between two successive sets of data entering the pipeline is called the* **initiation interval.**

Definition 2.2 *The utilization of each module type* $0 \leq i \leq m-1$ *is defined as*

$$u_i = \frac{n_i}{l \times o_i} \tag{2.1.1}$$

where n_i *is the effective number of nodes of operation type* i *in the data flow graph,* l *is the initiation interval of the pipelined design,* o_i *is the number of modules of type* i *used in the implementation and* m *is the number of different types of operations in the data flow graph.*

For an operation type, utilization of one is *module-optimal.*

Definition 2.3 *If* $u_i = 1$ *for all operation types,* $0 \leq i \leq m-1$, *then the implementation is a* **module-optimal design.**

For example consider Figure 1.2 where $n_{addition} = 12$ and $n_{multiplication} = 16$. If this data flow graph is implemented with four adder modules and five multiplier modules and initiation interval is four ($l = 4$), then $u_{addition} = \frac{12}{4\times4} = 0.75$ and $u_{multiplication} = \frac{16}{5\times4} = 0.8$. If the implementation with initiation interval equal to four uses three adders and four multipliers then $u_{addition} = u_{multiplication} = 1$ and the design is module-optimal. Of course, in practical designs, it is often not possible to utilize all the modules in every cycle, resulting in suboptimal designs.

For a data flow graph with conditional branches, the effective number of nodes in the data flow graph n_i is computed as the sum of

1. the number of unconditional nodes of type i, and

2. for every pair of outermost *dist* - *join* nodes, the maximum number of nodes of type i in that conditional branch. For more details refer to Lemma 4.5.3 of [Par85].

This sum represents the maximum number of the type i nodes to be executed during any single execution instance of the data flow graph.

The total area of the design is defined as $A = \sum_{i=0}^{m-1}(a_i \times o_i)$, where a_i is the area of module which implements operation type i. The delay between successive output (or inputs) is $T = l \times c_p$, where c_p is the clock cycle. Delay between two successive outputs of the pipeline is also a measure of the throughput of the pipeline.

2.2 Clock Cycle Prediction

Prior to predicting the AT_p curve itself, a lower bound on the value of the clock cycle which is used for the lower-bound predictions must be derived. Since this is a lower bound, latching delays are not included in clock cycle computations. The lower-bound on the clock cycle can be computed using only the module delays (d_i). The clock cycle computation is used to derive the lower-bound area-delay curve. The clock cycle is first derived and then substituted in

the final area-delay equation to give the complete model. The number of partitions the graph is divided into is enumerated from 1 to the maximum number of operations (n_i) of all types in order to plot the AT_p curve, and, like the clock cycle, does not need to be specified a priori.

Theorem 2.1 *For a module-optimal design, $c_p = maximum(d_i)$, where the maximum is taken over all modules used in the implementation.*

Proof: For a module-optimal design the utilization, $u_i = 1, 0 \leq i \leq m-1$ and Equation 2.1.1 reduces to

$$o_i = \frac{n_i}{l}$$

From the definition of total functional area A,

$$A = \sum_{i=0}^{m-1} (a_i \times o_i) = \sum_{i=0}^{m-1} (a_i \times \frac{n_i}{l})$$

where a_i is the area of module which implements operation type i.

For a given module set, the area of each module type a_i is constant. Likewise, the number of nodes of type i, n_i is fixed for a given data flow graph. Hence, *for a fixed value of initiation interval*, the total area of the module-optimal design is a constant. Consider two pipelined designs of the same data flow graph with the same initiation interval, and hence same area, but different clock cycles. The design with lower clock cycle time will have a lower AT_p^x for any $x > 0$ (as the initiation interval and area of the two designs is the same). From the assumption that every operation must complete execution within a clock cycle, the minimum value c_p can take is $maximum(d_i)$.

This argument holds good for every value of initiation interval, and for every value of initiation interval the minimum value of $c_p = maximum(d_i)$, we conclude that for lower-bound estimates, $c_p = maximum(d_i)$. ■

Associated with each module set there is exactly one clock cycle for module-optimal designs. It is interesting to see that the optimal value of clock cycle does not depend on the topology of the data flow graph, but only on the modules selected for implementation of the data flow graph. (This is not true if resynchronization is considered.)

2.3 Lower-Bound Area-Delay Model

Having derived the optimal value of the clock cycle, we now determine the lower-bound area-delay tradeoff curve for pipelined designs. Using Definition 2.2, the tradeoff curve is based on the following theorem.

Theorem 2.2 *Given a data flow graph, for module-optimal pipeline design when* $\forall_i u_i = 1$, $(AT_p)_{min} = constant$ *for all initiation intervals, where* $(AT_p)_{min}$ *is the lower-bound of all possible* AT_p*'s for that graph.*

Proof: For any module, $l \geq n_i/o_i$. For module-optimal designs, every module is utilized every clock cycle and

$$l \times o_i = n_i$$

Multiplying both sides by the area of the module a_i,

$$l \times a_i \times o_i = a_i \times n_i$$

Summing this over all m operations which occur in the data flow graph, and multiplying by the clock cycle,

$$c \times l \sum_{i=0}^{m-1} (a_i \times o_i) = c_p \sum_{i=0}^{m-1} (a_i \times n_i)$$

$$A \times T_p = c_p \sum_{i=0}^{m-1} (a_i \times n_i) \tag{2.3.2}$$

For a given data flow graph and a module set, the right hand side is a constant, and hence

$$(AT_p)_{min} = constant \tag{2.3.3}$$

■

Theorem 2.2 predicts an area-delay tradeoff curve for module-optimal designs with the shape $(AT_p)_{min} = constant$. All design points lying on this curve are module-optimal. However, for many data flow graphs, it is not possible for every non-inferior design to be module-optimal. That is, the point $(AT_p)_{min} =$

constant represents a design which is not possible for every initiation interval. A tighter lower bound $(AT_p)_{lb}$, which will be used later, can be obtained using the algorithm given in Figure 2.1. This algorithm is executed with initiation interval l varying from 1 to $maximum(n_i)$ to get all non-inferior design points. This tighter lower-bound design point $(AT_p)_{lb}$ is different from a module-optimal design point $(AT_p)_{min}$. Only when the equality $\lceil n_i/l \rceil = n_i/l$ holds do we have the case where a module-optimal design point is the same as the tighter lower-bound design. This difference is illustrated by an example: Consider a data flow graph with three addition operations. For an initiation interval of one, three adders are required for the design, and every adder will be utilized every clock cycle. This design is module-optimal and produces a lower-bound design point. For an initiation interval equal to two, two adders are required, of which one adder will lie idle for one clock cycle. This design point is not module-optimal. However, it is a lower-bound point because we cannot achieve a better design for an initiation interval equal to two.

A design with $(AT_p)_{min}$ or $(AT_p)_{lb}$ may not exist. In fact, the best possible design might not have the minimum clock cycle. No design, however, will have a point closer to the origin than the curve computed by Figure 2.1 under the above assumptions. If resynchronization is considered, the actual curves move even farther from the origin.

2.3.1 Complexity Analysis

In this subsection, we derive the run time complexity [AHU74] of the prediction algorithm given in Figure 2.1. For every value of initiation interval l, Steps 1, 2 and 3 take $O(m)$ time to execute. The value of initiation interval itself can vary from one to $maximum(n_i)$. Thus, the runtime complexity of generating the complete lower-bound area-delay tradeoff curve is

$$O(m \times maximum(n_i)) \tag{2.3.4}$$

```
procedure estimate_lower_bound(l)
begin
1.      for 0 ≤ i < m calculate o_i = ⌈n_i/l⌉ ;
2.      for 0 ≤ i < m calculate A = A + o_i × a_i;
3.      c_p = maximum(d_i);
4.      T_p = c_p × l;
        print A × T_p ;
end;
```

Figure 2.1: Algorithm for Predicting Lower-Bound AT_p Curve

Thus, in $O(m \times maximum(n_i))$ computations we can predict the entire area-delay tradeoff curve for pipelined designs. Compare this to the number of computations required to actually schedule all possible designs, which in the case of Sehwa is $O(Q^5\ log(Q))$, where Q is the total number of nodes in the data flow graph (irrespective of mutual exclusivity). For a data flow graph Sehwa produces $O(Q^2)$ possible clock cycles and for each clock cycle it produces a scheduled data flow graph in $O(Q^2\ log(Q))$, giving the overall complexity of Sehwa to be $O(Q^4\ log(Q))$ without exhaustive search. Further, in order to produce the entire area-delay tradeoff curve, Sehwa would have to perform $O(Q^5\ log(Q))$ computations.

For a data flow graph with equal numbers of nodes of different types, $m \times maximum(n_i)$ gives the total number of mutually exclusive nodes M. For this special case, the worst-case runtime complexity of the predictive algorithm reduces to $O(M)$.

2.4 Experiments and Results

Several experiments using a pipelined synthesis tool Sehwa [PP88] were conducted to verify the lower-bound area-delay curve. Sehwa generates near-

optimal designs and hence such a comparison is possible. Sehwa concurrently performs scheduling and module allocation. Sehwa takes into consideration conditional branching within the data flow graph and resynchronization due to resource conflicts and data dependencies. The scheduling is a static scheduling which takes into account all possible combinations of conditional branches which can occur. First, Sehwa produces the fastest and the cheapest designs to fix the design space boundary. Sehwa then requests the user for a speed or cost constraint and generates several solutions meeting the constraint. The user then changes the constraints and iterates. Finally, the user can perform exhaustive search in a small part of the design space to tune the design. By altering the design constraints Sehwa can synthesize a number of designs for the same input description.

The experiments were conducted with several module sets and different data flow graphs. In this thesis we give results produced by Sehwa and the estimation procedure for four data flow graphs and the module set (a_1, m_1, s_1) (Table 1.1). To represent computing in signal processing applications we have selected the AR filter (Figure 1.2), and an elliptical wave (EW) filter (Figure 2.4) [2.2]. In addition we present an example data flow graph containing several conditional branches (Figure 2.2) and finally a data flow graph which was generated by using a random number generator (Figure 2.3). The results of the experiments with Sehwa and the predictor are shown in Figures 2.5 through 2.8.

As seen from Figures 2.5 through 2.8, the results produced by Sehwa are the same as those produced by the lower-bound area-delay algorithm. The experiments show that Sehwa produces optimal results in several cases and that the prediction model is accurate. The results of the remaining experiments using other data flow graphs and module sets were identical.

[2.2] The elliptical wave filter is one of the 1988 High-Level Synthesis Workshop benchmarks.

2.5 Summary

In this chapter we have derived the lower-bound area-delay curve for pipelined designs. This model accepts a data flow graph and a module set and produces the curve consisting of operator optimal designs which can be generated by any pipelined synthesis tool. The theory has been verified by Sehwa, a pipelined synthesis tool.

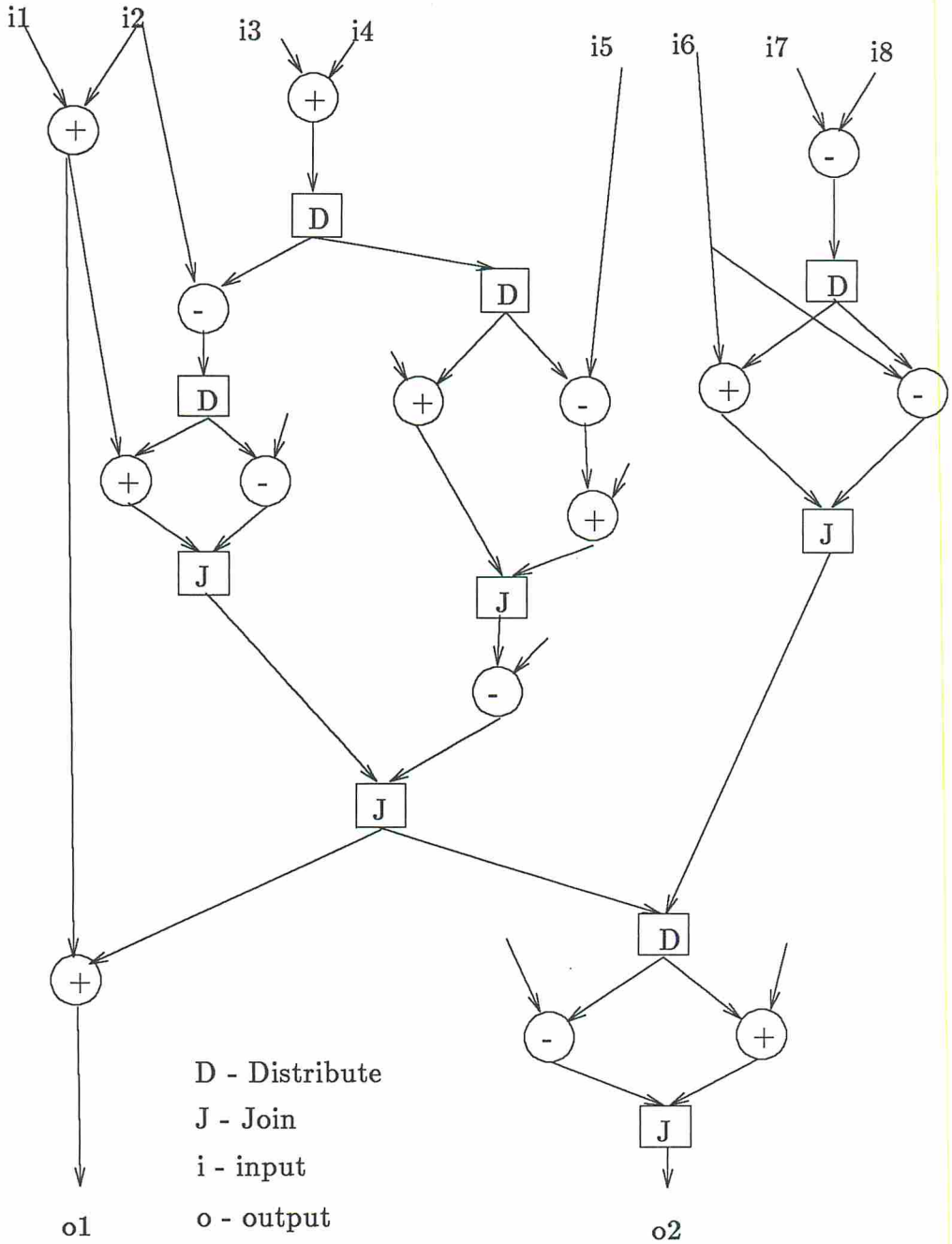


Figure 2.2: Data Flow Graph with Conditional Branches

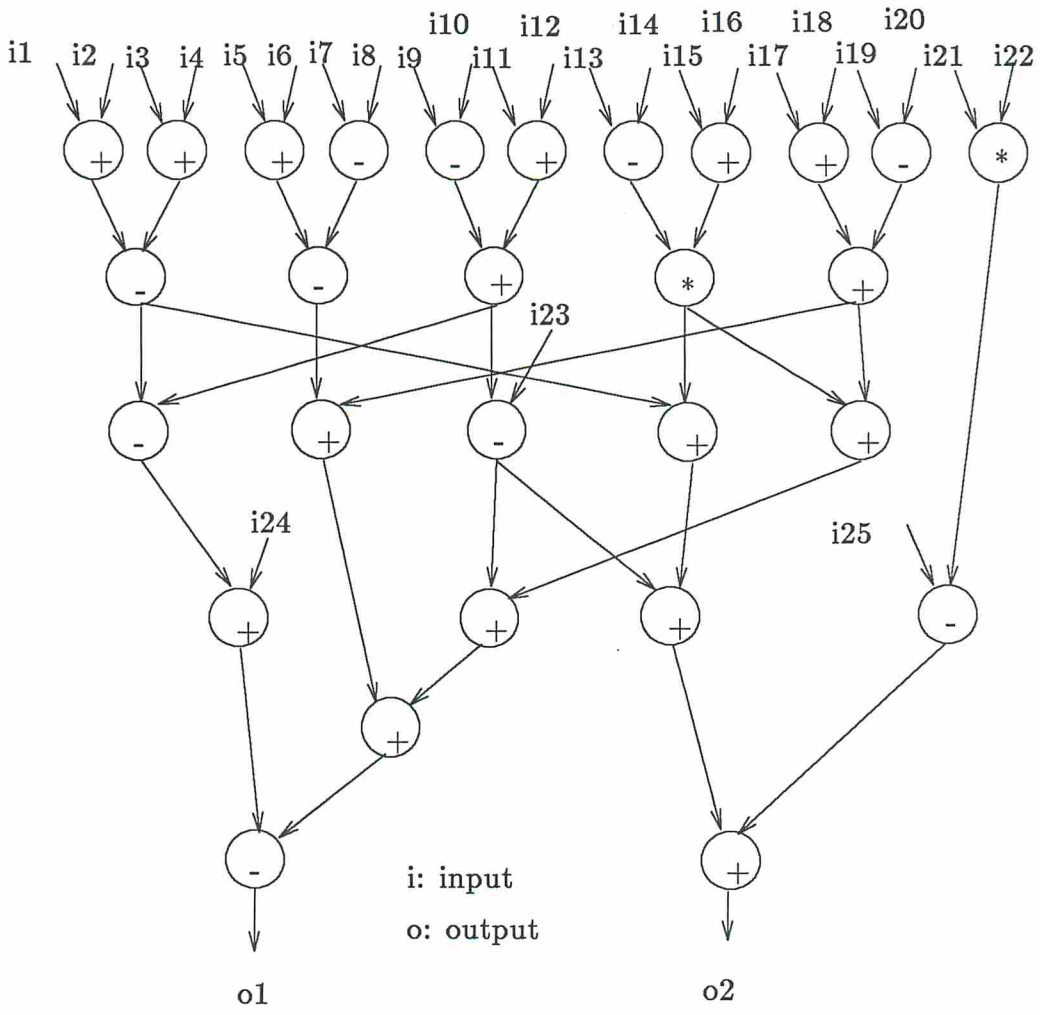


Figure 2.3: Random Data Flow Graph

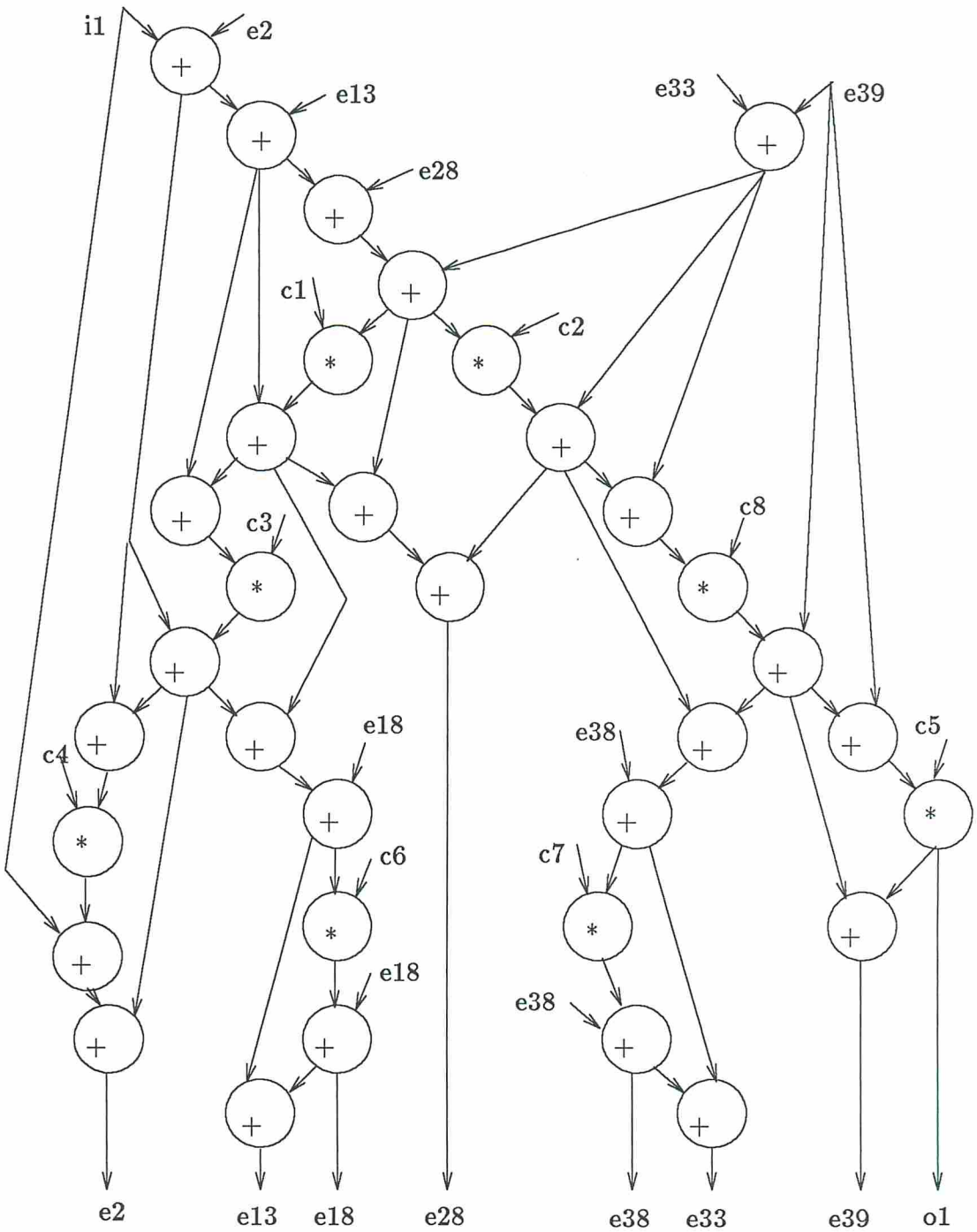


Figure 2.4: Elliptical Wave Filter

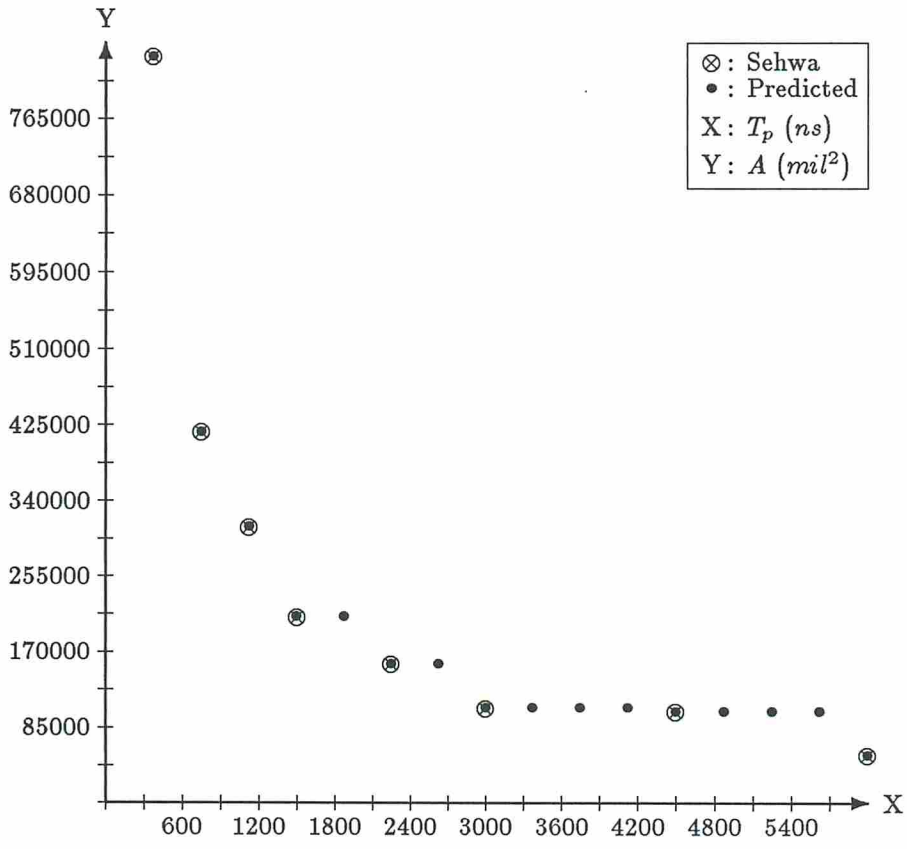


Figure 2.5: Area-Delay Curves for AR Data Flow Graph

The predictors used in the ADAM system are shown in Figure 1.6. Using a data flow graph specification of behavior and a chosen module set, the predictors generate area-delay tradeoff curves. First, the predictors examine the operator area and delay to produce an area-delay tradeoff curve. Next, the register and multiplexer area predictor [MP88a] uses the behavioral description, the predicted number of operator resources of each type and the number of time steps and computes the average multiplexer and register count. The register and multiplexer quantities are then multiplied by their respective areas and added to the operator area-delay tradeoff curve. Next, the wiring area is added to the area-delay tradeoff curve. Given the average wire length, total cell width, and number of nets, and assuming a standard cell placement, PLEST [KP87] predicts the data path wiring area. In parallel, the control area is predicted and added to the area-delay curve. PASTA [MP88c] is a control area predictor which accepts as input the number of time steps, information about loops and conditionals, design style, and the register count to predict folded and unfolded PLA controller area.

1.4 Problem Specification and Approach

In this section, we formally state the problems of area-delay prediction, design style selection, computing the impact of data flow graph transformations and module selection.

1.4.1 Area-Delay Prediction

The problem of lower-bound area-delay prediction for high-level synthesis is the following: *Given a behavioral description and a module set* [1.6] *from the design library, predict the area and performance of all module-optimal designs.*

[1.6] A module set is a set of modules selected from a design library which are used in the implementation of the behavioral description. A module set consists of exactly one module type for every operation type.

The total area consumption is a function of many parameters.

$$A_{total} = f(A_{op}, A_{reg}, A_{mux}, A_{wiring}, A_{pads}, A_{controller}, A_{unused})$$

where A_{total} is the total chip area, A_{op} is the operator area, A_{reg} is the register area, A_{mux} is the multiplexer area, A_{wiring} is the wiring area, A_{pads} is the area of the input-output pads, $A_{controller}$ is the area of the controller and A_{unused} is the unused area on the chip. We have assumed that this function can be broken into its components as follows:

$$\begin{aligned} A_{total} &= f(A_{op}, A_{reg}, A_{mux}, A_{wiring}, A_{pads}, A_{controller}, A_{unused}) \\ &= f(A_{op}) + f(A_{reg}) + f(A_{mux}) + f(A_{wiring}) + f(A_{pads}) + f(A_{controller}) \\ &\quad + f(A_{unused}) \end{aligned}$$

The assumption of breaking the problem into its individual component has enabled us to study each sub-component in isolation in order to solve the complete problem. This thesis deals with only one aspect of the area-delay prediction problem, namely computing A_{op}. Area-delay predictors for register, multiplexer, wiring (standard cell layout), and controller (PLA controller) have been developed in [KP86] [Kur87], [MP88a], [MP88c] under the following dependency assumptions:

$$\begin{aligned} A_{reg} &= f(data\ flow\ graph, op) \\ A_{mux} &= f(data\ flow\ graph, op, reg) \\ A_{controller} &= f(reg,\ mux,\ states,\ product\ terms) \\ A_{wiring} &= f(average\ wire\ length,\ number\ of\ nets) \end{aligned}$$

Similarly, the delay of the design is assumed to be a function of several individual delays.

$$T_{total} = g(T_{op}, T_{latch}, T_{mux}, T_{wire}, T_{unused})$$

where T_{op} is the operator delay, T_{latch} is the latching delay, T_{mux} is the multiplexer delay, T_{wire} is the wiring delay, and T_{unused} is dead time. In this thesis we only

solve for T_{op}. From now on,unless otherwise specified, we refer to A_{op} and T_{op} as area and delay.

Chapters 2 and 4 discuss the lower-bound area-delay tradeoff curves for pipelined and non-pipelined designs respectively.

1.4.2 Design Style Selection

Having outlined the features of an area-delay predictor for pipelined and non-pipelined design styles, we demonstrate its application to design style selection. The problem of design style selection is briefly stated as follows. *Given a behavioral description, a module set, and a design constraint, select the design style which satisfies the design constraint and optimizes the design goal best.*

There are several design styles from which a choice of implementation can be made [Tho77]. These design styles can be broadly categorized as non-pipelined, pipelined, or parallel (in which the hardware is replicated several times to meet the input data rate). Additional design styles can be formed by combining these three styles. For example, in the parallel design style, where every hardware element is replicated several times to achieve the desired performance, each hardware element may itself be pipelined. Hardware replication is seldom the preferred style because of the large cost involved. In this thesis we assume that a selection is to be made between pipelined and non-pipelined design styles only. The selection technique presented in this thesis, however, is general enough to be applied to the parallel design style as well.

We propose to solve the design style selection problem as follows. Given a behavioral description and a module set, we use the lower-bound area-delay predictors to generate the lower-bound area-delay tradeoff curve for pipelined and non-pipelined design styles. Taking the area-delay curves for the two design styles we select the one that most nearly fits the design constraints. For example, Figure 1.7 shows lower-bound area-delay tradeoff curves for the pipelined and non-pipelined design styles. The prediction tool suggests that the non-pipelined

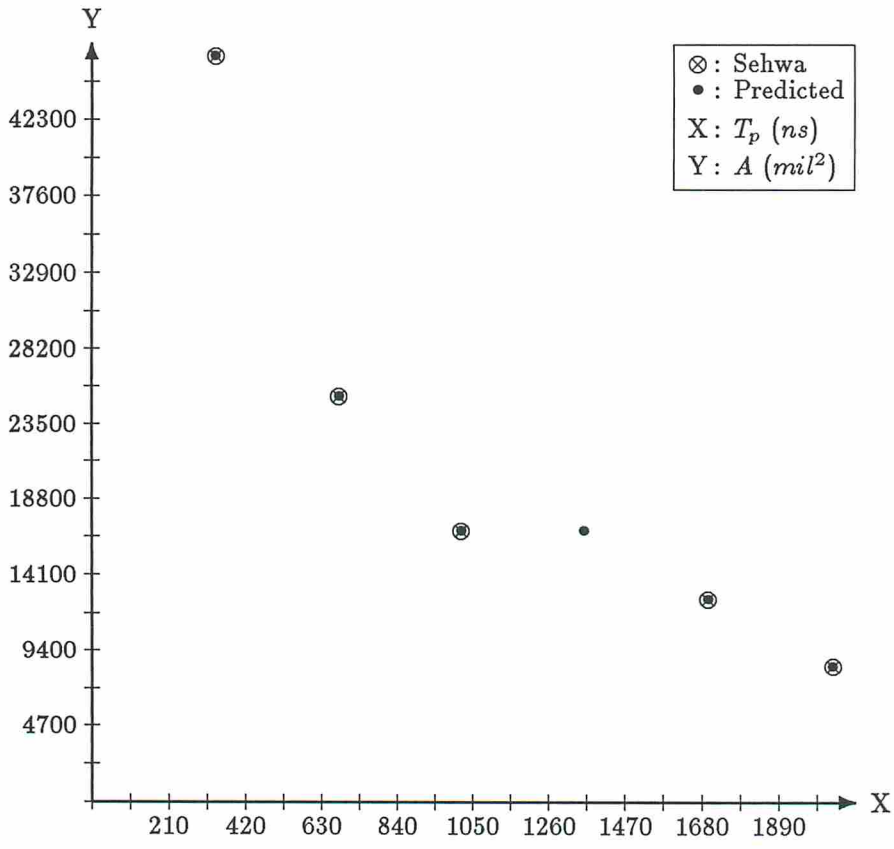


Figure 2.6: Area-Delay Curves for Conditional Data Flow Graph

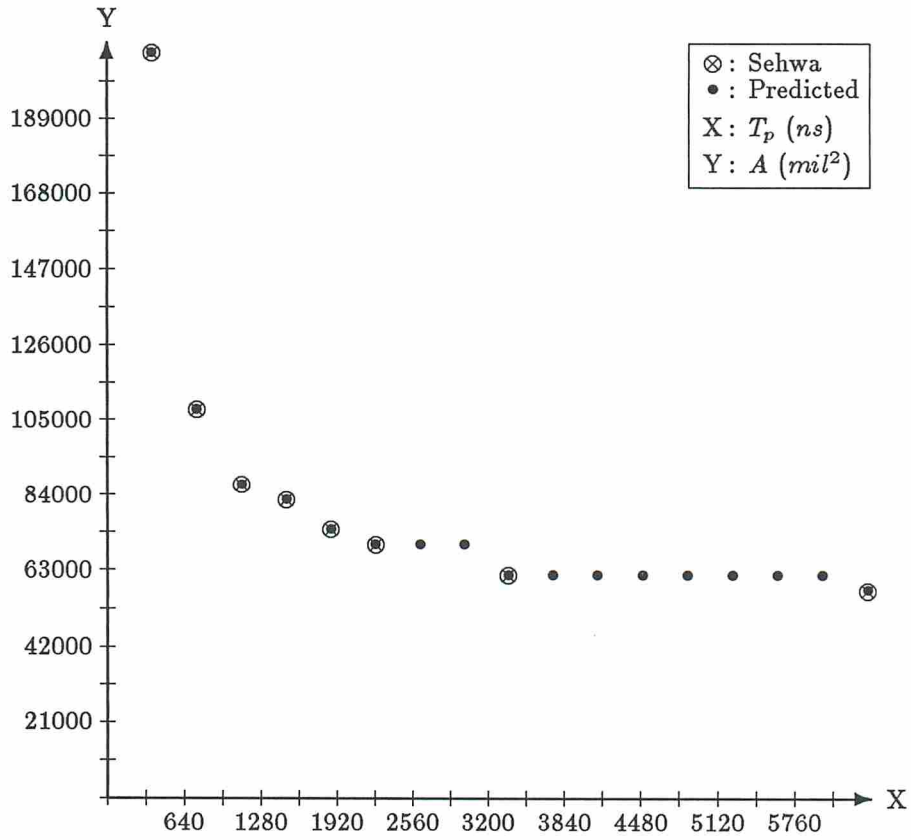


Figure 2.7: Area-Delay Curves for Random Data Flow Graph

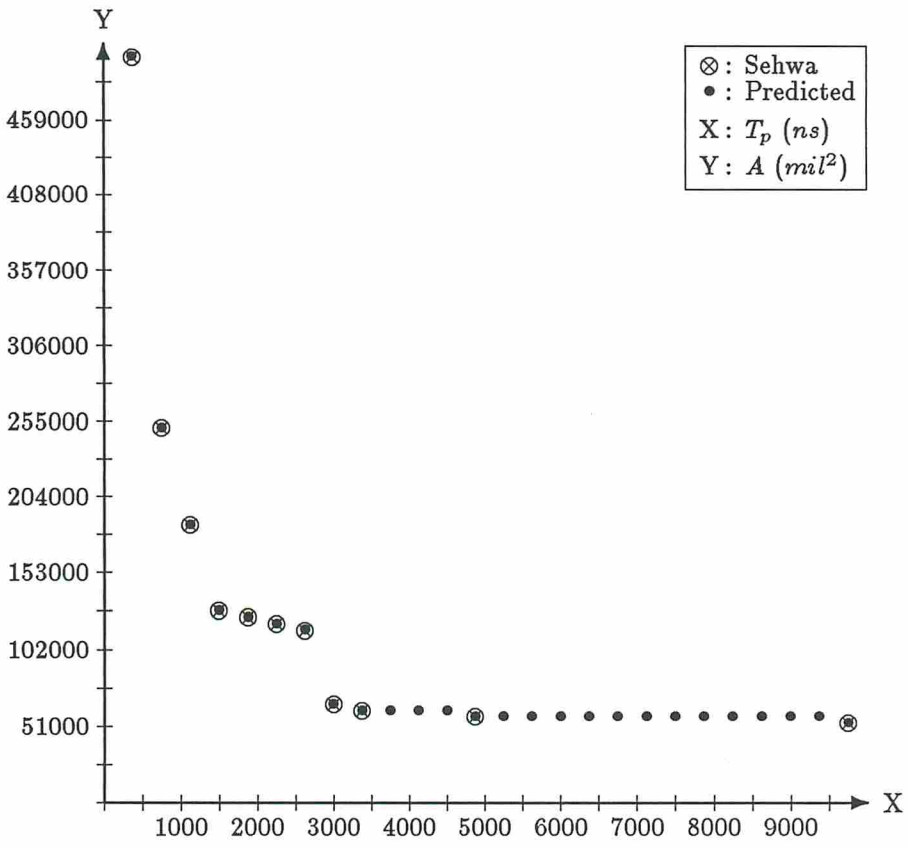


Figure 2.8: Area-Delay Curves for EW Filter

Chapter 3

Module Selection for Pipelined Designs

3.1 Introduction

While developing the lower-bound area-delay model for pipelined designs, the module set was assumed to be predetermined. Using this model the effect of different module sets on the area-delay tradeoff curve will be studied and the analysis will be used to solve the module selection problem. In Sections 2.1.1 and 2.1.2 the input data processing steps required for module selection and the assumptions were given. The problem complexity is discussed in Section 3.1.1. An overview of the solution technique is described in Section 3.2. In Section 3.3, an algorithm based on this model which generates several module sets will be presented. A technique for selecting an appropriate module set given an area or initiation delay design constraint is described in Section 3.3.3. Several experiments using SLIMOS [3.1], a program based on the theory developed in this chapter, were conducted. The results of these experiments are summarized in Section 3.4. A method for selecting the best module set for a generalized objective function is discussed in Section 3.5. The effects of resynchronization on the module selection problem are illustrated with several examples in Section

[3.1] Selection from LIbrary of MOdule Sets

3.6. A brief comment on the module generation problem is made in Section 3.7. Suppose that the design library consists of design curves for each module type instead of discrete components. Given such a design library, how should one perform module selection? This problem is solved for initiation delay constraints in Section 3.8. In Section 3.9 we discuss the effect of register and multiplexer area on module selection. An analysis of the runtime complexity of the algorithms developed in this chapter is given in Section 3.10.

3.1.1 Problem Complexity

In the *general module selection* problem for an implementation of a data flow graph, different module types can be used for each instance of the same operation type. For example, a ripple carry adder may be used to implement an addition node and a carry look ahead adder may be used to implement another addition node. There are $p_i^{q_i}$ different possibilities for each operation type, where q_i is the total number of nodes of operation type i in the data flow graph, and p_i is the number of different types of modules in the design library which can perform operation i. Considering m different types of operations there are an exponential number of unique module sets :

$$\prod_{i=0}^{m-1} p_i^{q_i} \tag{3.1.1}$$

In the *restricted module selection* problem all operations in the data flow graph of the same type are implemented using the same module type. For example, if a ripple-carry adder is used to implement an *addition* operation in the data flow graph, then all *addition* nodes in the data flow graph are implemented using ripple-carry adders. In this case, there are

$$\prod_{i=0}^{m-1} p_i \tag{3.1.2}$$

unique module sets. This restriction is not serious since prior to module selection the user can force selection of different module types for the same operation type by creating two or more operation subtypes for the operation nodes to be

differentiated. (See Section 2.1.1 for further details.) In Section 3.3.2 we will show that if resynchronization does not exist, the optimal solution to the general problem is identical to the restricted problem.

The restricted module selection problem solved in this chapter generates a maximum of $\sum_{i=0}^{m-1} p_i$ module sets which meet the following two goals:

1. the design space explored by considering all $\prod_{i=0}^{m-1} p_i$ module sets spans the same design surfaces as that spanned by the generated $\sum_{i=0}^{m-1} p_i$ (or fewer) module sets, and

2. if a design point meeting some constraint can be implemented by the ungenerated module sets, then the generated module sets can also implement a design point which will not only satisfy the same constraint but has lower area-delay product.

3.2 Theory of Module Selection

For a given data flow graph and a fixed module set a lower-bound area-delay curve can be generated as described in Figure 2.1. This curve represents design points with different initiation intervals and spans a part of the design space. A different area-delay curve for the same data flow graph can be generated by using a different module set. Thus, by using different module sets for a data flow graph, larger design spaces can be spanned and a good design can be selected from a larger set of candidate designs. Figure 3.1a shows an example design space spanned by four module sets for the same data flow graph. Each rectangle shows the design space spanned by the lower-bound area-delay curve lying within it. Design spaces spanned by different module sets may overlap partially or completely. The combined design space explored by all four module sets is shown in Figure 3.1b. From Figure 3.1b it is observed that the contribution of module set a in the design space exploration is redundant and can be eliminated. A design constraint which can be satisfied by module set a can also be satisfied better by other module sets. If the design space spanned by a module set A can

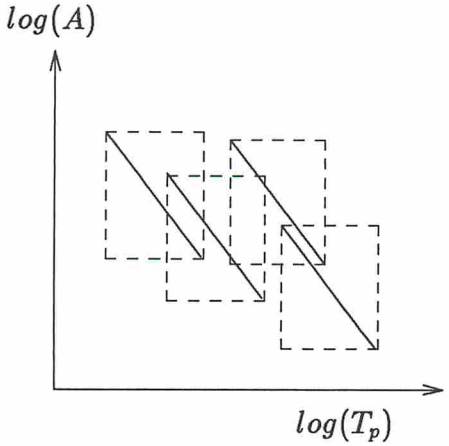
be spanned by other module set(s) and the design points generated by module set A are inferior, then module set A can be eliminated from the list of possible candidates for the solution.

Associated with every module set there is a best possible value of clock cycle given by Theorem 2.1. Several module sets may have the same best value of clock cycle. All module sets with the same associated clock cycle and some identical initiation interval will overlap in the design space as shown in Figure 3.2. Of the module sets with the same associated clock cycle, the module set which satisfies the objective value (to be defined later in this section) best is generated. The module set on the top in Figure 3.2 can be eliminated because, for the same design constraint (area or initiation delay) the rejected module set produces designs inferior to the other module sets.

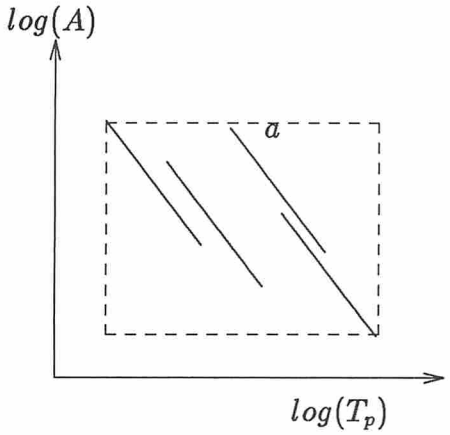
Consider Figure 3.1c which shows design curves for three module sets and an area constraint of 200 mil^2. This design constraint cannot be met by any design point generated by module set b and module set b can also be eliminated from the list of possible candidates.

In the above discussion we have assumed that all design points generated with a module set are module-optimal. In reality this is not true and local search has to be made to select the best choice.

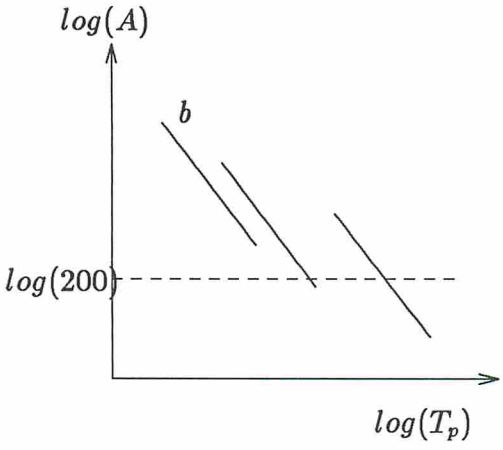
We begin the discussion on the theory of module selection problem with the aid of an example. Consider a data flow graph with two types of operations, addition and multiplication. Let us generate a design for each of the following module sets (1) multiplier and ripple-carry adder ($d_{multiplier} > d_{ripple-carry}$), and (2) multiplier and carry look-ahead adder ($d_{look-ahead} < d_{ripple-carry}$ and $a_{look-ahead} > a_{ripple-carry}$). We know that the best clock cycle for pipelined design is given by $c_p = maximum(d_i)$. For design (1) $c_p = d_{multiplier}$, and for design (2) also $c_p = d_{multiplier}$. However, the area of the first design will be smaller than the area of the second design as the area of the ripple-carry adder is less than the area of the carry look-ahead adder. Thus, without sacrificing performance,



(a) Design Space Spanned by Four Module Sets



(b) Total Design Space Spanned by all Four Module Sets



(c) Module Set Meeting A Design Constraint

Figure 3.1: Design Space Exploration Using Several Module Sets

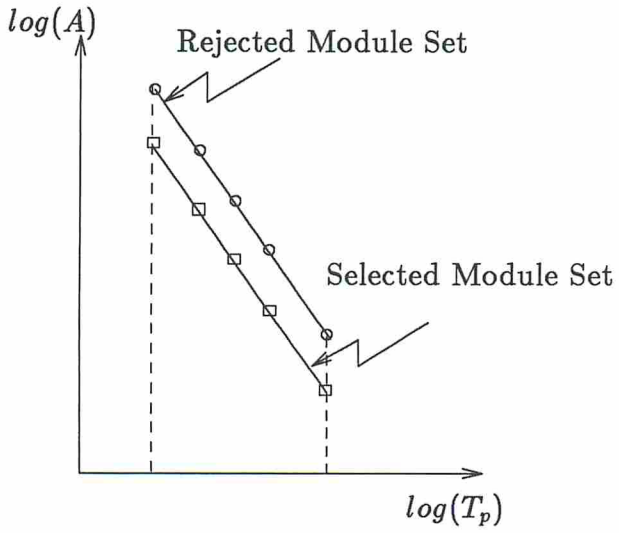


Figure 3.2: Module Set Generation

the first design is more area-efficient. This idea has been used in developing the solution to the module selection problem.

Definition 3.1 *The objective functions are defined as follows:*

1. *for objective function "minimize initiation delay " the objective function is minimize* ($1 \times \text{maximum}(d_i)$),
2. *for objective function "minimize area" the objective function is minimize* $\left(\sum_{i=0}^{m-1} \lceil n_i/l \rceil a_i\right)$, *and*
3. *for objective function "minimize area-delay product" the objective function is*

$$minimize \left[maximum(d_i) \sum_{i=0}^{m-1} (a_i \times n_i) \right]$$

(this is the best area-delay product for the module set).

The following theorem helps in developing the solution to the module selection problem.

Theorem 3.1 *For a fixed clock cycle, the optimal module set (ms_{opt}) with objective function of minimizing the area-delay product of the design consists of modules with identical delay equal to the value of the clock cycle.*

Proof: Pipelined design is characterized by

$$AT_p = c_p \sum_{i=0}^{m-1} (a_i \times n_i) \tag{3.2.3}$$

To minimize Equation 3.2.3 for any given clock cycle, we have to select modules such that $\sum_{i=0}^{m-1}(a_i \times n_i)$ is minimized without changing the clock cycle. We observe that the number of operations of any type is constant for the data flow graph. Thus, the only variable is the area of the modules. The cheapest module permitted by the clock cycle requirement (delay of a module should not exceed the clock cycle) will minimize Equation 3.2.3. We know that a faster module is bigger than a slower module of the same type. Hence, the module with minimum area and which still meets the clock cycle requirements is the module with delay equal to the clock cycle value. ■

In other words for a fixed clock cycle, the best module set is one in which all modules have the same delay equal to the clock cycle, assuming again, no modules extend over more than one clock cycle. Since the library may not contain modules of every type with delay identical to the clock cycle, the ms_{opt} module set may not exist. In such cases module set (ms_{lb}) consisting of modules with minimum area and delay less than or equal to the clock cycle will minimize the area-delay product.

Also, if no two modules of the same type have identical delay, then the module set which minimizes the area-delay product is unique.

Corollary 3.2.1 *For a given data flow graph and any fixed clock cycle, the design which consists of all modules having delay equal to the clock cycle (i.e. ms_{opt}) and which satisfies the design constraint will be the one with*

1. *minimum area-delay product which meets an area or initiation delay constraint,*

2. *minimum initiation delay which meets an area constraint, or*

3. *minimum area which meets an initiation delay constraint.*

Proof: We know that for pipelined designs AT_p is a constant and from Theorem 3.1 when the delay of all modules is equal to the clock cycle the value of this constant is minimum. This proves the first property. We know that for module set ms_{opt}, $AT_p = constant$ is minimum. Given an area constraint a, and substituting the constraint value we get $T_p = constant/a$. Since *constant* is minimum over all possible constants for different module sets, T_p is minimum for the given area constraint. This proves the second property. If the area-delay product is minimal, then for an initiation delay constraint satisfied by this design, area is minimized for that clock cycle. This proves the third property. ■

The clock cycle for a module-optimal pipelined design with a fixed module set is (Theorem 2.1)

$$c_p = maximum(d_i) \tag{3.2.4}$$

The number of different values of delays in the library (and consequently the number of different clock cycles) is upper bounded by $\sum_{i=0}^{m-1} p_i$. Thus, the minimum number of module sets is $\sum_{i=0}^{m-1} p_i$ since for a given clock cycle, exactly one unique module set ms_{lb} exists (proved in Section 3.3.1). This is a subset of the total number of possible module sets ($\prod_{i=0}^{m-1} p_i$). The set of possible values of clock cycles can be reduced by deleting all potential clock cycle values which are smaller than the smallest possible delay of any operation type. For example, the clock cycle value 340 can be deleted from the set of possible values of clock cycles when using the adder, subtractor and multiplier modules given in Table 1.1, as 375 is the smallest clock cycle possible.

3.2.1 SLIMOS - A Module Selection Program

SLIMOS is a software package based on the theory of module selection which generates the best module set after three processing steps. In the first step a set of candidate module sets is generated. If there are no constraints,

the module set which minimizes the objective function is chosen, and SLIMOS terminates. Otherwise processing step two is started. In step two SLIMOS selects the module sets which meet the design constraint and discards the remaining module sets.

In the third step, SLIMOS optimizes the objective function by performing local search over the module sets selected by step two. For an area constraint, SLIMOS prompts the user to choose an objective function of initiation delay or area-delay product, and finds the module set which meets the area constraint and optimizes the objective function. The two objective functions may lead to different results. Let us illustrate this by an example. Figure 3.3 shows the lower-bound area-delay curves for a hypothetical design generated using two module sets a and b. Let us assume that the user wants to minimize initiation delay satisfying the given area constraint (shown by a dashed horizontal line). On comparing the actual designs which can be predicted for the two module sets, we observe that a design produced using module set b has a lower initiation delay than the designs produced using module set a (satisfying the area constraint). Thus, module set b is a better choice than module set a for the given area constraint and the optimizing function of minimizing initiation delay. If the optimization function requires minimizing the area-delay product, then module set a will be selected. This "fine tuning" of module selection is achieved through local searching.

SLIMOS also also provides the designer with a choice of two objective functions for initiation delay constraint: minimizing the area-delay product or minimizing area. The two objective functions may lead to different results. Figure 3.4 shows an initiation delay constraint on the design. According to the objective function of minimizing the area-delay product module set 1 will be selected for implementation. For an optimizing criteria of minimizing area, however, module set 3 will be selected for implementation. This situation may arise when the area of the design is at a premium and SLIMOS provides the user with both the options of minimizing either AT_p or area.

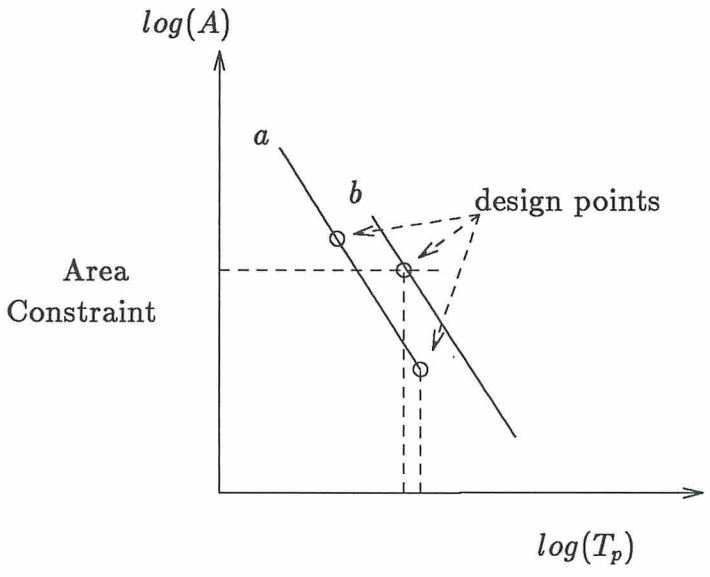


Figure 3.3: Module Selection with an Area Constraint

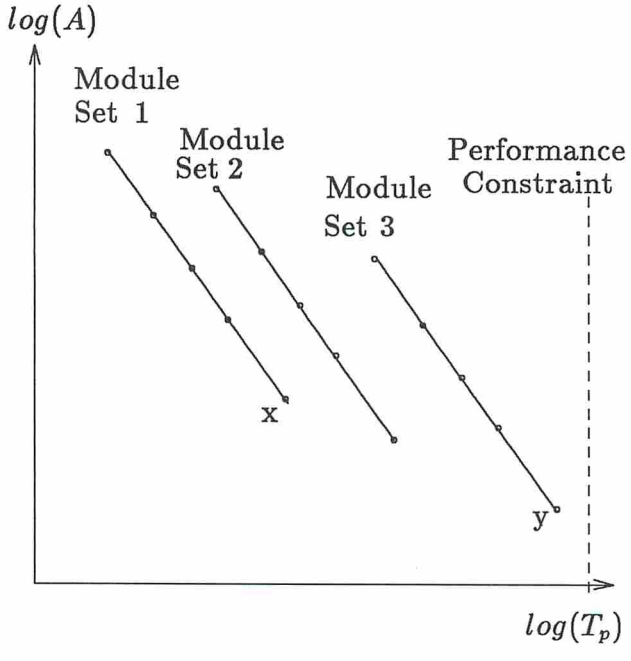


Figure 3.4: Module Selection with an Initiation Delay Constraint

3.3 Module Selection Algorithms

Module selection consists of three major steps and each step consists of many stages.

3.3.1 Algorithm One: Candidate Module Set Generation

Algorithm 1 of the solution consists of module set generation and rank assignment. As mentioned earlier there are a total of $\prod_{i=0}^{m-1} p_i$ module sets which can be generated from the design library. However, some of these module sets can be eliminated. The module generation procedure outlined in this section eliminates some of these redundant module sets and generates a maximum of $\sum_{i=0}^{m-1} p_i$ module sets. Let X be the set of possible candidate module sets.

The set of unique delays of all modules in the design library forms the set of possible values of clock cycles and is used in Step 1 of Algorithm 1. Algorithm 1 is given in Figure 3.5 and it generates a module set for every possible clock cycle which minimizes AT_p. This is achieved easily by ensuring the delay of every chosen module type does not exceed the selected clock cycle and has minimum area. The pruning of the search space for a selected clock cycle is done in Step 3 of Algorithm 1.

Theorem 3.2 *Module sets in the set X generated by Algorithm 1 satisfy the following properties:*

1. *the optimal design space explored by considering all possible $\prod_{i=0}^{m-1} p_i$ module sets is the same as that explored by the generated $\sum_{i=0}^{m-1} p_i$ (or less) module sets,*
2. *if a design meeting some constraint can be implemented by the un-generated module sets, then the generated module sets can implement a design which not only satisfies the same constraint but has lower area-delay product, and*
3. *generated module sets in X optimize the given objective function.*

1. For every possible value of clock cycle {
2. For each operation type {
3. Choose the module with minimum area and delay less than or equal to the clock cycle.
   If no module implementing an operation type has a delay less than the selected clock cycle then this clock cycle is rejected.
   }
4. Add this module set to set X.
   }

Figure 3.5: Algorithm 1 - Module Set Generation

Proof: Associated with every module set there is a minimum value of clock cycle given by Equation 3.2.4. Several module sets may have the same minimum clock cycle. In this situation one module set is selected and the rest are discarded.

Let G be the set of all module sets which have $c_p = y$ as their associated clock cycle. Given a design constraint and the objective function set G contains exactly one module set which minimizes the objective function. Let the module set generated by Algorithm 1 with a clock cycle of y be $g \in G$. We will now show that the optimal design surface covered by the module set g is the same as that covered by all the module sets in G. The optimal design surface is the design space bounded by the fastest and the cheapest design which can be produced by G. Figure 3.2 shows an example design space covered by two module sets with the same clock cycle.

For the fastest design, initiation interval $l = 1$ and $T_p = ly = y$ for all module sets in G. Since the clock cycle for all module sets in set G is identical and $g \in G$, g can generate the fastest design with an initiation delay of y. Equation 3.2.3 can be written as $AT_p = y \sum_{i=0}^{m-1}(a_i \times n_i)$. As y and n_i are constants for G and the given data flow graph respectively, the module set g can minimize this

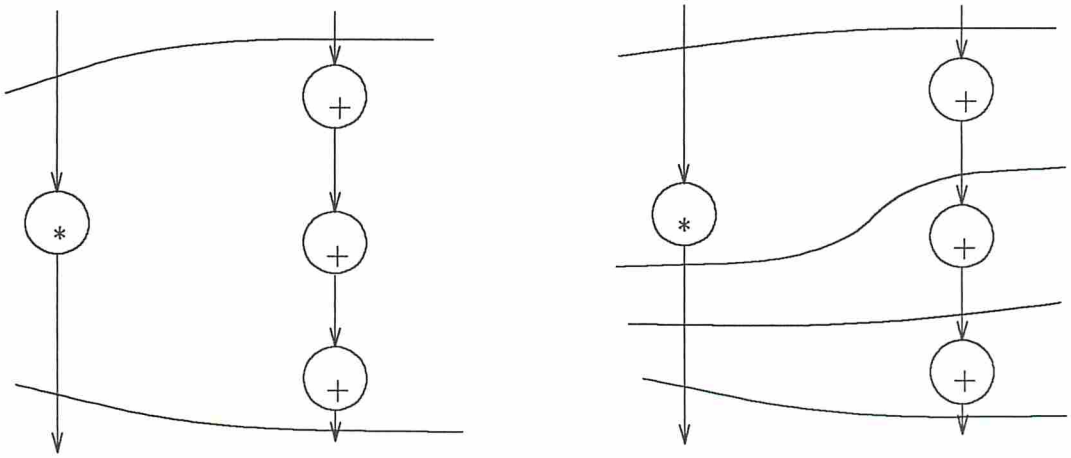
expression only if g contains the smallest modules meeting the clock cycle. This is ensured in Step 3 of Algorithm 1. As g contains the smallest modules, the smallest design can be generated by g. Thus, g can generate the fastest and the smallest design while spanning as much design space as all module sets belonging to G.

The module set generated for a clock cycle covers the optimal design space for that clock cycle value, and all unselected module sets with the same associated clock cycle value are covered. The set of all possible module delays make up the set of all possible clock cycles, and Step 1 of Algorithm 1 ensures that for every possible clock cycle value a module set is generated. Hence, all ungenerated module sets are covered by a generated module set. This proves the first property.

As the selected module set g can generate the fastest and the smallest design amongst all module sets in G, any design constraint which can be met by a module set in G can also be met by g. Further, g produces the most area-delay efficient design in G. This proves the second property.

The objective function for an area constraint can be initiation delay or area-delay product. As the module sets in set X generate the most area-delay efficient curve the objective function of minimizing area-delay is easily satisfied. Further, since the area-delay product is minimized, for any fixed area the module sets in X can also produce a design with minimum initiation delay and conversely for any fixed minimum delay the module sets in X can produce a design with minimum area. This proves the third property. ■

Algorithm 1 discards expensive modules in favor of cheap ones excepting the ones which dominate the value of the clock cycle and subsequently affect the scheduling of the data flow graph. Consider a data flow graph with two schedules given in Figure 3.6: Figure 3.6a is a schedule produced by pipeline synthesis when presented with a non-optimal module set and a tight performance constraint; Figure 3.6b is a schedule produced using our optimal module set selection. In Figure 3.6a, let the delay of the multiplier be greater than three times the delay of adder. Clock cycle time is equal to the delay of the multiplier, and the initiation



(a) Latency = 1
Number of Partitions = 1

(b) Latency = 1
Number of Partitions = 3

Figure 3.6: Example for Algorithm 1

interval of this design is one. The area and the throughput of the pipe, assuming no resynchronization and the pipe is kept full, is $A = a_{multiplier} + 3a_{adder}$ and $T_p = d_{multiplier}$ respectively.

Figure 3.6b shows another schedule of the same data flow graph. In this design, the add modules are slower than the add modules of Figure 3.6a and also smaller (*adder* is the module used for the addition operation in the design given by Figure 3.6a and *adder'* is the module used for the addition operation in the design of Figure 3.6b). Note that the initiation interval of the design is still one. The area and the delay of this design, with the same assumptions as above, is $a_{multiplier} + 3a'_{adder}$ and $d_{multiplier}$ respectively. Further, $a'_{adder} \leq a_{adder}$. Thus, without affecting the initiation delay[3.2], the area of Figure 3.6b is less than or equal to the area of the design in Figure 3.6a. Algorithm 1 selects a module set such that designs, as in Figure 3.6b, may have a large number of partitions.

[3.2] With resynchronization, Figure 3.6a would be a lower initiation delay design, since flushing the pipe would take less time.

We can map the module set generation step to a 0-1 linear programming problem. However, since the solution time for 0-1 linear programming problems is large, we developed an algorithm (see Algorithm 1) which guarantees optimal results in a much smaller time. The 0-1 linear programming formulation is as follows. We know that ar_{ij} (dl_{ij}) is the area (delay) of module type j which can implement operation type i. The lower-bound area-delay curve for pipelined design style is given by $A \times T_p = c_p \sum_{i=0}^{m-1}(a_i \times n_i)$. The number of modules in the design library of type i is p_i. Let $y_{ij} \in \{0,1\}$, $0 \leq i \leq m-1$, $0 \leq j \leq p_i - 1$, be the decision variables. ar_{ij}, dl_{ij}, and n_i are fixed by the design library and the behavioral description. Then, for any fixed value of clock cycle c_p we can formulate the module set generation as follows.

Objective function:

$$minimize \sum_{i=0}^{m-1} \sum_{j=0}^{p_i-1} c_p(ar_{ij} \times y_{ij} \times n_i)$$

Constraint equations:

$$\sum_{j=0}^{p_i-1} y_{ij} = 1, \quad 0 \leq i \leq m-1$$

$$\sum_{j=0}^{p_i-1} (y_{ij} \times dl_{ij}) \leq c_p, \quad 0 \leq i \leq m-1$$

Solving this formulation for every possible value of clock cycle will generate the required module sets.

3.3.2 The General Module Selection Problem

In the general module selection problem different nodes of the same operation type can be implemented using different modules. It will be shown that, without resynchronization, the general module selection model reduces to the restricted problem for the same design constraints and objective functions.

Suppose for a given data flow graph, we use Algorithm 1 and generate a module set which is optimal for restricted module selection. Additional optimization may be achieved by assigning different modules to nodes which have

the same operation types. Let us select a node for further optimization. This optimization is possible by picking a smaller module for implementation without changing the value of clock cycle or by placing more nodes in a single cycle by selecting faster modules. Since the module chosen for implementing this node by Algorithm 1 is already minimal in area (delay being less than or equal to the clock cycle), no further reduction in area is possible without changing the value of clock cycle. Hence for a particular value of clock cycle, area has already been minimized. Conversely, if we reduce delays so that more operations fit into a single cycle, we could decrease initiation delay only if resynchronization were considered[3.3]. Thus, Algorithm 1 also produces optimal module sets for the general module selection problem.

Theorem 3.3 *If there is no resynchronization, the module sets generated by Algorithm 1 for the restricted module selection problem satisfy the following properties for the general module selection problem as well:*

1. *the optimal design space explored by considering all the* $\prod_{i=0}^{m-1} p_i$ *module sets is the same as that explored by the generated* $\sum_{i=0}^{m-1} p_i$ *(or fewer) module sets,*
2. *if a design point meeting some constraint can be generated by the unselected module sets, then the selected module sets can also generate a design point which not only satisfies the same constraint but has lower area-delay product, and*
3. *generated module sets in* X *optimize the objective function.*

Proof: We provide the proof in two steps. First, the general module selection problem is modeled and then it is shown to reduce to the model for restricted module selection problem. Let ar_{ij} and dl_{ij} be the area and delay of the module type i which performs the function j (for example, a_{11} could be a ripple-carry type adder which performs addition operation). Also, let h_{ij} be the quantity of this module used in the final implementation (for all i, $o_i = \sum_{j=0}^{p_i-1} h_{ij}$). Then, for the

[3.3] A high resynchronization rate leads to the design of a pipeline with fewer stages [PP88].

general module selection $A = \sum_{i=0}^{m-1} \sum_{j=0}^{p_i-1} (ar_{ij} \times h_{ij})$, and $T_p = lc_p$. c_p depends upon the selected modules and is equal to the delay of the slowest module. Thus,

$$AT_p = lc_p \sum_{i=0}^{m-1} \sum_{j=0}^{p_i-1} (ar_{ij} \times h_{ij}) \tag{3.3.5}$$

Equation 3.3.5 is the model for the general module selection problem. For the best solution we select modules which minimize this expression for every value of clock cycle. Let clock cycle $c_p = y$ in Equation 3.3.5,

$$AT_p = ly \sum_{i=0}^{m-1} \sum_{j=0}^{p_i-1} (ar_{ij} \times h_{ij})$$

Consider any one operation type. The library contains several modules which can implement this operation type. Of these several modules, there are some modules with delay less than or equal to y. However, there will be exactly one which has a minimum area while satisfying the clock cycle requirement. Thus, for every operation type there is exactly one module which will be selected for implementation. Hence, ar_{ij} reduces to a_i, dl_{ij} reduces to d_i and h_{ij} reduces to o_i.

As the reduction holds for every possible clock cycle the theorem is proven. ■

3.3.3 Algorithm Two: Module Sets Meeting Design Constraint

At the second step of the module selection technique, location of module sets which satisfy design constraints is performed. Users may specify a *constraint* which is not met by the design points generated using a module set which minimizes the objective function. In this situation the user may have to settle for a module set with inferior objective function value which meets the constraint.

Figure 3.1c shows an example of selecting the best possible module set in X, assuming that each design point is module-optimal. Different curves correspond to different module sets and the design points on each curve are for

1. For every module set $j \in X$ compute $AL_j = \sum_{i=0}^{m-1} a_i$.
   /* This is the area of the cheapest design with this module set.
   (The right end design point of the AT_p curve) */
2. For every module set $j \in X$ compute $TU_j = maximum(d_i)$.
   /* This is the speed of the fastest design with this module set.
   (The left end design point of the AT_p curve) */
3. If $constraint\_type = cost$ then
4. Let S be the set of module sets $i \in X$ such that $AL_i \leq constraint$.
   /* $S \subseteq X$ */
5. If $constraint\_type = speed$ then
6. Let S be set of module sets $i \in X$ such that $constraint \leq TU_i$.
   /* $S \subseteq X$ */

Figure 3.7: Algorithm 2 - Selecting Module Sets Which Satisfy the Constraint

different initiation intervals. Suppose the user specifies an area constraint of 200 mil^2 (shown by a dotted horizontal line). Then, module set a cannot satisfy this constraint and is rejected. Only module set b can satisfy the constraint. In the second step, SLIMOS will collect all module sets which satisfy the design constraint and pass them to Algorithm 3 for processing. A similar approach is used when the user specifies an initiation delay constraint.

3.3.4 Algorithm Three: Optimizing the Objective Function

In the third and final processing step SLIMOS selects the best module set which optimizes the objective function from the set S generated by Algorithm 2. Local searching is performed in this step since actual predicted area-

delay tradeoff curves may cross and to handle the objective function optimization correctly.

An important observation from Figure 3.1c can be made for designs with cost constraints. Given the design library shown in Table 1.1, for module-optimal designs, when searching for a cheaper design, it is always better to sacrifice parallelism than to change the module set to a cheaper one. The area is reduced more as delays are increased by serializing the design rather than by substituting the cheaper modules. For Figure 3.1c, given a cost constraint, it is best to start on the curve generated by module set a. If the cost constraint is not satisfied, then one should serialize the design and stay on that curve until no further cost reduction is possible. If the cost constraint is still not met, then a jump to the next best curve, which is module set b, is made. The reason for the curves behaving as such under varying module sets is very simple. For the same initiation interval, if an optimal design point generated from a module set i has $(AT_p)_i$ less than $(AT_p)_j$ of an optimal design point generated from module set j, then all optimal design points generated from module set i will have lower AT_p than the optimal design points of module set j. Graphically, this is because area-delay curves for module-optimal designs theoretically have the same slope, and hence do not cross each other. Furthermore, if all module sets in S are sorted in order of their objective function values, then this list of sorted module sets gives the order in which the search for best possible module set must be made given a user constraint.

So far we have considered a design library (Table 1.1) where the module set which is used to produce the fastest and most expensive design has the minimum area-delay product (for a given data flow graph). In reality, we may have a design library where the module set which produces the fastest and most expensive design does not have the minimum area-delay product. For example, consider a data flow graph consisting of five addition nodes and two multiplication nodes, and a hypothetical design library given in Table 3.1. The area-delay curves for this data flow graph using the three module sets generated by Algo-

	Area (mil^2)	Delay (ns)
add_1	100	100
add_2	50	120
$mult_1$	1000	100
$mult_2$	500	150

Table 3.1: A Hypothetical Design Library

Module Set	Area-Delay Product $maximum(d_i) \sum_{i=0}^{m-1}(n_i \times a_i)$ (10^3 mil^2 ns)
$add_2, mult_2$	187.5
$add_1, mult_1$	250
$add_2, mult_1$	270

Table 3.2: Area-Delay Product of the Three Module Sets

rithm 1 are plotted in Figure 3.8 [3.4]. For a delay constraint of 150 ns, module set $(add_1, mult_1)$ is the optimal module set choice, and for larger delay constraint module set $(add_2, mult_2)$ is the optimal module set choice. That is, the search for the optimal module set for this design library is ordered by going from one module set to another rather than by sacrificing parallelism (as was done for the design library given in Table 1.1). The area-delay products of the designs using the three module sets generated by Algorithm 1 are given in Table 3.2.

Let us consider the situation in which all design points are not module-optimal. Consider a data flow graph with two multiplication operations and ten addition operations. If this data flow graph were pipelined with initiation interval of one, then there would be two multipliers in the implementation which would be kept busy every clock cycle. If the design were implemented with one multiplier and data initiation interval of two, then this module would still be

[3.4] In Figure 3.8, area-delay curves using module sets $(add_1, mult_1)$ and $(add_2, mult_1)$ intersect because of non-optimal design points.

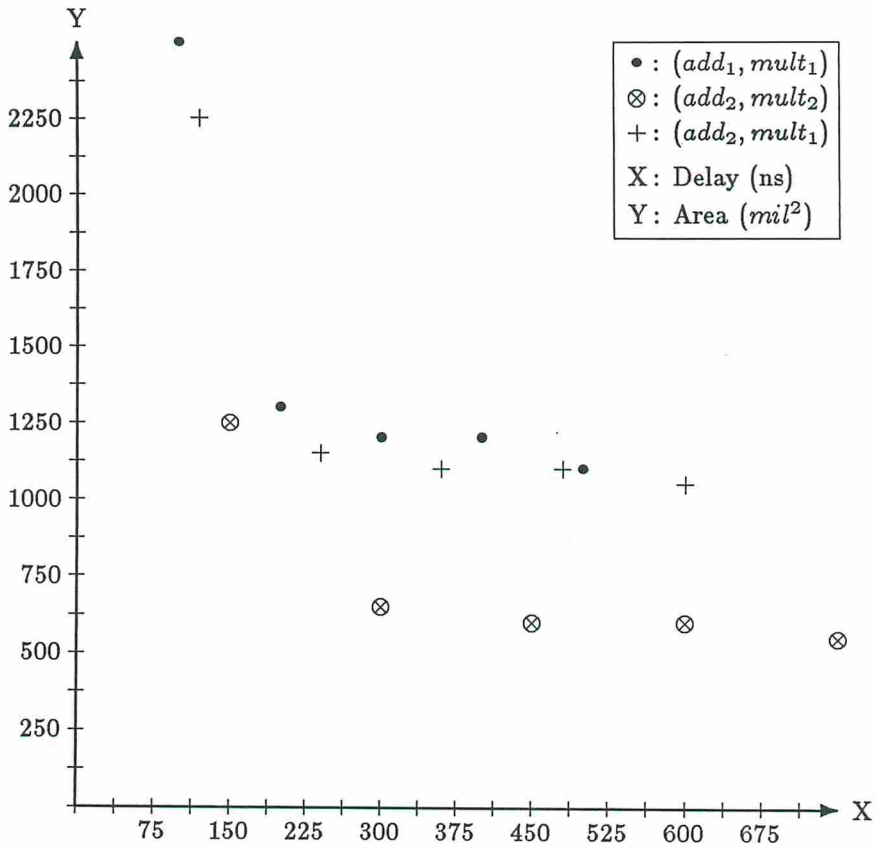


Figure 3.8: Area-Delay Curves for a Hypothetical Data Flow Graph

1. Sort set S in increasing order of the value of the objective function.
2. j = 1.
3. Generate all design points using module set j. Select the design which satisfies the constraint and optimizes the objective function.
4. If module set $j+1$ does not exist then go to Step 7.
5. Generate all design points using module set $j+1$. Select the design which satisfies the constraint and optimizes the objective function. If such a design does not exist then go to Step 7.
6. If the design selected in Step 5 has a better objective function value than the design selected in Step 3, then { $j <= j+1$. Go to Step 3.}
7. Module set j is selected.

Figure 3.9: Algorithm 3 - Performing Local Search to Select the Best Module Set

used every clock cycle. For initiation interval of $l > 2$ the implementation would require one multiplication module, and this module would lie idle for $l-2$ clock cycles. The implementation would no longer be module-optimal. In fact, of the total area of the design A, area of size $a_{multiplier}$ will be unused for $l-2$ clock cycles.

There may be some design points using module set j which are optimal and some which are not. In case the design points are not optimal, whether their AT_p is less than any of the module-optimal design points using module set $(j+1)$ or not has to be calculated. As such, a search for the design points generated by using module set $(j+1)$ may also have to be made. Similarly, a check has to be made for module set $(j+2)$ and so on. The local searching given in Algorithm 3 examines the adjacent module set $(j+1)$ only. However, it can be easily expanded to search over $(j+2)$ and others as well.

AR Lattice Filter	Conditional Data Flow Graph	Random Data Flow Graph
$(m_1, a_1)^*$		$(m_1, a_1, s_1)^*$
(m_1, a_2)	$(a_1, s_1)^*$	(m_1, a_2, s_2)
(m_2, a_3)	(a_2, s_2)	(m_2, a_3, s_3)
(m_3, a_3)	(a_3, s_3)	(m_1, a_3, s_3)
(m_1, a_3)		(m_3, a_3, s_3)

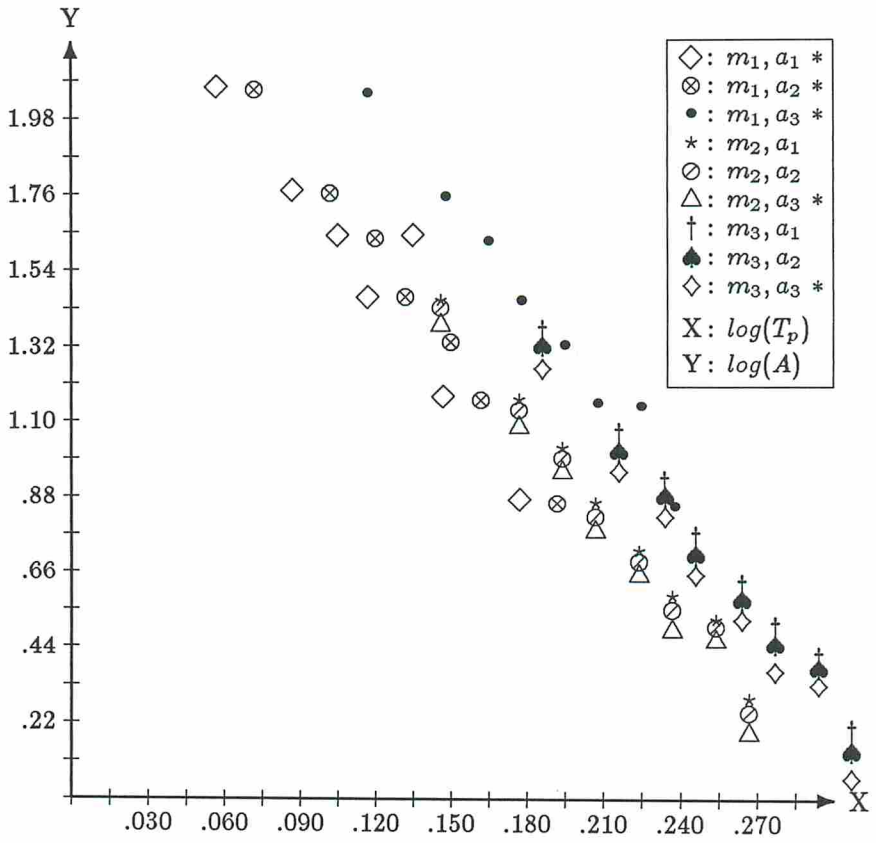
* Module set with minimum AT_p.

Table 3.3: Module Sets Generated Using Algorithms 1

3.4 Experiments and Results

Several experiments were conducted using SLIMOS to ensure that Algorithm 1 does indeed generate module sets satisfying the properties given in Section 3.2. The results were verified using Sehwa. The library of modules consisted of three add modules, three subtract modules, and three multiplication modules (Table 1.1). In this thesis we give results for the three data flow graphs given in Figures 1.2, 2.2 and 2.3. The AR filter data flow graph (Figure 1.2) consists of addition and multiplication operation types. As the library has three add modules and three multiplication modules, a total of nine various combinations of module sets can be formed for this example. Sehwa was executed using these module sets for the AR filter and its results shown in Figure 3.10 (plotted on a normalized log-log scale for better readability).

Algorithm 1 was executed using the AR filter data flow graph and the library. Algorithm 1 generated five module sets which are listed in Table 3.3 (the entries are sorted in increasing AT_p from top to bottom). Comparing the results in Figure 3.10 produced by Sehwa with those produced by Algorithm 1 in Table 3.3, it is seen that the five module sets produced by Algorithm 1 do encompass as much optimal surface in the design space as all nine module sets. The five selected module sets *cover* the four unselected module sets. For this example, five module sets is not the minimal number of module sets required to cover the



* indicates the module set selected by Algorithm 1

Figure 3.10: Area Time Curves For AR-Lattice Filter

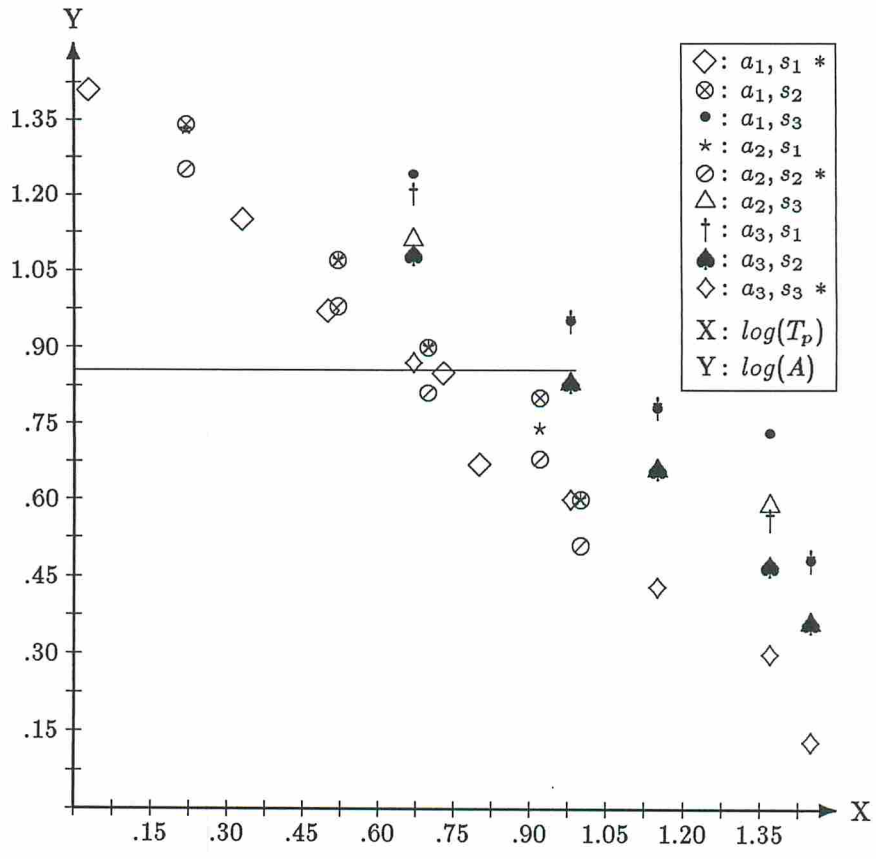
Data Flow Graph	Number of Operation Types	Total Number of Module Sets $\prod_{i=0}^{m-1} p_i$	Expected Number of Module Sets $\sum_{i=0}^{m-1} p_i$	Number of Module Sets Generated
AR Filter	2	9	6	5
Conditional	2	9	6	3
Random	3	27	9	5

Table 3.4: Summary of Results

optimal design space. Selected module set (m_1, a_3) is a redundant module set which is covered by the other four selected module sets.

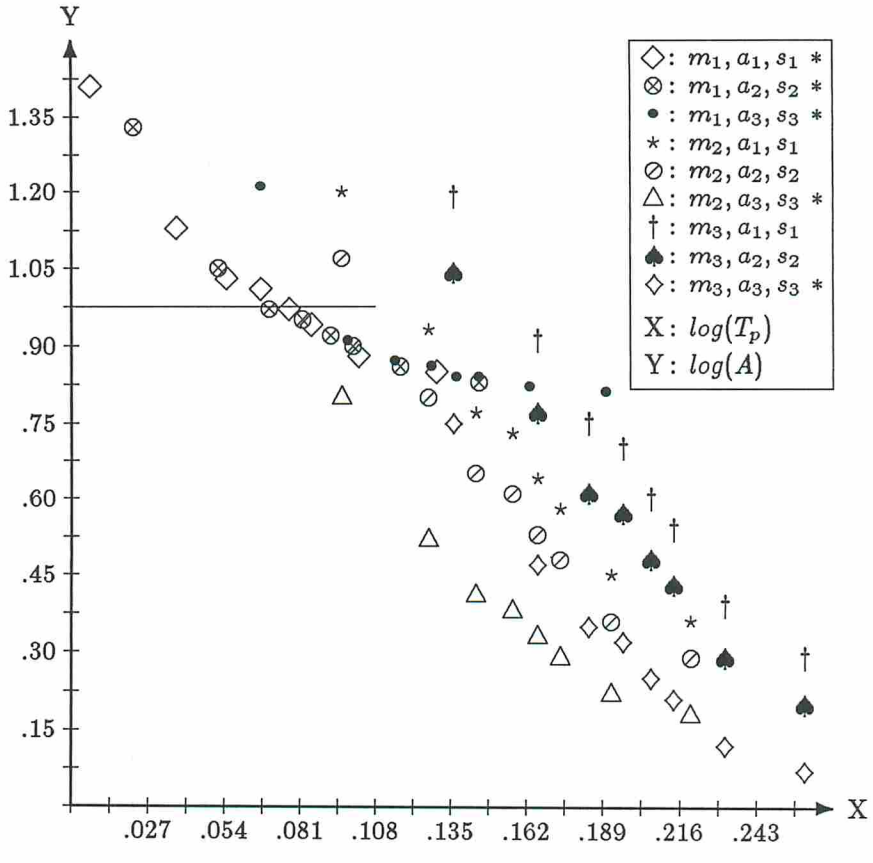
Similar results for other data flow graphs were obtained. Figures 3.11 and 3.12 show results when non-optimal designs were encountered and local search took place. In Figure 3.11, module set (a_2, s_2) was selected finally over the initial choice (a_1, s_1) when the area constraint of 85 and objective function of area-delay minimization was specified (shown in the figure by a horizontal line). In Figure 3.12 module set (m_1, a_2, s_2) was selected finally over the initial choice (m_1, a_1, s_1) when an area constraint of 98 and area-delay minimization was specified. Table 3.4 summarizes the results of Algorithm 1 of SLIMOS.

Figure 3.10 illustrates that selection of modules whose individual AT_p products are minimum does not result in global optimization. Let us assume that module m_1 does not exist. By selecting modules on the basis of optimization of individual modules, module set (m_2, a_1) would be chosen for the AR lattice filter example. Seeing the results in Figure 3.10, we observe that this module set is not selected. Instead, a better choice which would cover the same optimal design space with better AT_p is module set (m_2, a_3). Thus, optimizing individual nodes in the data flow graph does not necessarily lead to a globally optimal solution for pipelined design.



* indicates the module set selected by Algorithm 1

Figure 3.11: Area Time Curves For Conditional Dataflow Graph



* indicates the module set selected by Algorithm 1

Figure 3.12: Area Time Curves For Random Dataflow Graph

3.5 Satisfying A Generalized Objective Function

The theory presented in this chapter can be easily adapted with minor modification to Algorithm 3 for any objective function of the type $A^x T_p^y, x, y \geq 0$. Algorithm 4, which can find the best module set for an area or delay constraint and any $A^x T_p^y$ objective function, is given in Figure 3.13 and is used in lieu of Algorithm 3. The searching principle behind Algorithms 3 and 4 is the same. In Algorithm 3 search for the best possible module set is done in an ordered fashion, since the objective function value for area, delay or area-delay product is easy to compute. In this case (Algorithm 3), search for the best module set is done by generating the best design point satisfying the objective function using every module set in set S and then selecting the best module set [3.5]. However, for the general objective function where $A^x T_p^y$ the the search for the optimal module set cannot be ordered, and the optimal module set is selected by generating all design points using all module sets and selecting the module set which produces the best design point (Algorithm 4). The worst case for Algorithm 3 occurs when search for the best module set is done over all module sets in S and in this situation Algorithm 3 degenerates to Algorithm 4. However, in all the experiments we conducted with the objective function of area, delay or area-delay product, the ordered search technique of Algorithm 3 found the optimal module set quickly and Algorithm 4 was not required.

3.6 Effect of Resynchronization

A preliminary study of the effect of resynchronization on the restricted module selection problem is given here. The average initiation interval (measure of effective performance) for a pipeline with P stages and average resynchronization rate ρ, $0 \leq \rho \leq 1$, is given by [PP88]:

[3.5] We are still merely generating the area and delay characteristics of the designs, and not actually synthesizing the designs themselves.

1. For every module set $j \in S$ {
2. Generate all design points using module set j.
   Select the design which satisfies the constraint and the objective function.
   Add this design point to the set Y.
   }
3. Select the design point from set Y which best satisfies the objective function.
4. The module set which generated the selected design point in Step 3 is the desired module set.

Figure 3.13: Algorithm 4

$$T_{task} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \tag{3.6.6}$$

The overall equation for AT_{task} of the design, including the resynchronization is

$$AT_{task} = (\sum_{i=0}^{m-1} (a_i \times o_i)) \times (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \tag{3.6.7}$$

For $P \leq l$, $T_{task} = lc_p$ and resynchronization rate has no effect on the initiation delay of the pipeline. This can be intuitively explained with the help of an example. Suppose there is a pipeline with three stages. A new task is initiated every three clock cycles ($l = 3$). If the current task causes resynchronization, the next task has to wait for three clock cycles for the pipe to be flushed. But the new task would have to wait for three clock cycles even if there was no resynchronization as the initiation interval of the pipeline is three.

The effect of module selection on scheduling is now illustrated. Consider the two schedules given in Figure 3.6. The initiation interval of both designs is one. For the first schedule, the AT_{task} of the design computed by Equation 3.6.7 is independent of the resynchronization rate and is equal to

$$(a_{multiplier} + 3a_{adder}) \times (1 + \rho(1-1))c_p$$
$$= (a_{multiplier} + 3a_{adder}) \times c_p \quad (3.6.8)$$

For the design in Figure 3.6b, the AT_{task} is

$$(a_{multiplier} + a'_{adder}) \times (1 + \rho(3-1))c_p$$
$$= (a_{multiplier} + a'_{adder}) \times (1 + 2\rho)c_p \quad (3.6.9)$$

(*adder* is the module used for the addition operation of the design given in Figure 3.6a and *adder'* is the module for addition in Figure 3.6b.)

Consider some hypothetical module parameters, $a_{multiplier} = 10000$, $a_{adder} = 1000$, $a'_{adder} = 300$, $c_p = 300$ and $\rho = .25$. For these parameters, Expression 3.6.8 evaluates to 39×10^5 and Expression 3.6.9 evaluates to 46.35×10^5 clearly indicating that the design given in Figure 3.6a is the better of the two designs. Here we have an example where having a faster adder, versus the slower one which would have been chosen by Algorithm 1, leads to a better design when resynchronization is possible. The module parameters and the value of the average resynchronization alone are not sufficient to decide which is a good design. The number of stages of the pipeline P, and initiation interval of the design should be known a priori. However, P can only be known after the scheduling has been done. If resynchronization is considered, the module selection and scheduling problems become closely interrelated and must be solved concurrently for an overall optimal design. With high rates of resynchronization, it becomes important to chain more than one node into same time step and reduce the number of stages.

The theory used for the generation of module sets (Algorithm 1) works because without resynchronization the best possible clock cycle can be identified before synthesis. However, resynchronization affects the best possible clock cycles. Whereas, without resynchronization, Equation 3.2.4 gave an optimal value of clock cycle, this is no longer valid if resynchronization is present. In reality, different designs produced using the same module set have different clock cycles,

and the notion of a module set having an associated clock cycle is no longer valid. This is illustrated by a simple example.

Consider some hypothetical module parameters for the designs given in Figure 3.6. Let $a_{multiplier} = 10000$, $d_{multiplier} = 1000$, $a_{adder} = 1000$, $d_{adder} = 350$, $a'_{adder} = 300$ and $d'_{adder} = 800$. Also, let $\rho = .25$. Then $c_p = 1050$ for Figure 3.6a and $c_p = 1000$ for Figure 3.6b. From Expression 3.6.7, the AT_{task} for the schedule of Figure 3.6a is 13.65×10^6 and the AT_{task} for Figure 3.6b is 15.45×10^6 indicating that Figure 3.6a is a better schedule.

Thus, resynchronization adds another dimension to the synthesis problem of finding an optimal value of clock cycle. The set of all possible values of clock cycles with resynchronization is larger than the set of all possible values of clock cycles without resynchronization. Sehwa finds the optimal value of clock cycle for scheduling meeting the resynchronization requirements by performing exhaustive search over all possible values of clock cycles. For further details on selecting an optimal value of clock cycle for pipelined designs refer to [Par85].

The module selection theory developed in this thesis will hold good for resynchronization if all the following are true:

1. the design curves using all module sets move in the same direction in the design space when resynchronization is added,

2. if two curves with zero resynchronization did not intersect, then after adding resynchronization they do not intersect, and

3. if two curves with zero resynchronization intersect, then they intersect at the same design points after resynchronization is added.

It is difficult to predict the number of stages which the data flow graph is going to be partitioned into and the effect of resynchronization is not known until scheduling is actually performed. Under such conditions it is not clear if SLIMOS will continue to produce optimal results. Several experiments were conducted using Sehwa with varying resynchronization rates for the AR Filter

and the conditional data flow graphs. These results are given in Figures 3.14 through 3.21.

For the conditional data flow graph, the design curves using every module set satisfy the above three properties and SLIMOS produces correct results. For the AR Filter example it is observed that SLIMOS generates the correct result for 10 %, 20%, and 30 % resynchronization rate. For 40 % resynchronization (Figure 3.21), the design curves (m_2, a_2) and (m_2, a_3) intersect and SLIMOS does not produce the best module sets. The design points produced by using module set (m_2, a_2) are non-inferior and module set (m_2, a_2) cannot be excluded from the set of possible solution.

It is observed from the experiments that SLIMOS produces good, not necessarily optimal, results with resynchronization. For low resynchronization rates it has been shown experimentally to produce optimal results. However, as the resynchronization rate increases the quality of results can deteriorate.

3.7 Specifying Module Constraints for Module Generators

The theory developed for solving the problem of module selection can be used to specify constraints for module generation. The problem of module generation is: generate a module which can perform the given function, meet some design constraint and optimize some design goal. For example, a module generation program can be directed to generate a module which performs 4-bit addition (desired function), has 200 ns delay or less (design constraint), and has minimum area (goal). Given this information, the program will generate a module which satisfies these requirements. If the module generator cannot satisfy these requirements, then an error flag is set.

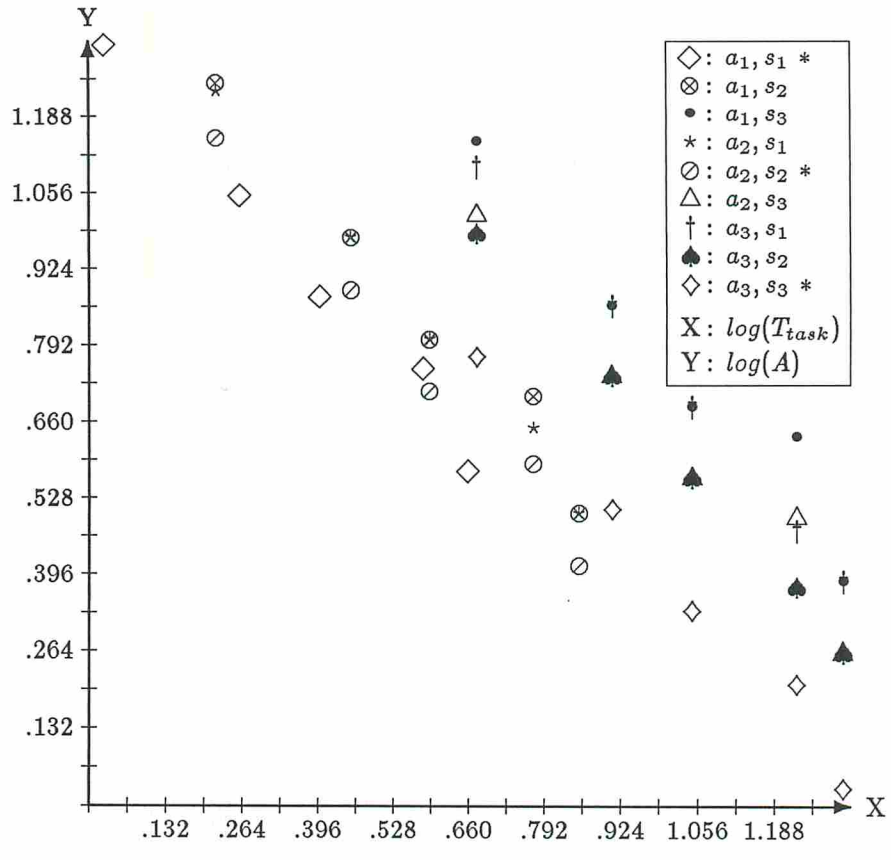


Figure 3.14: Conditional Example - 10 % Resynchronization

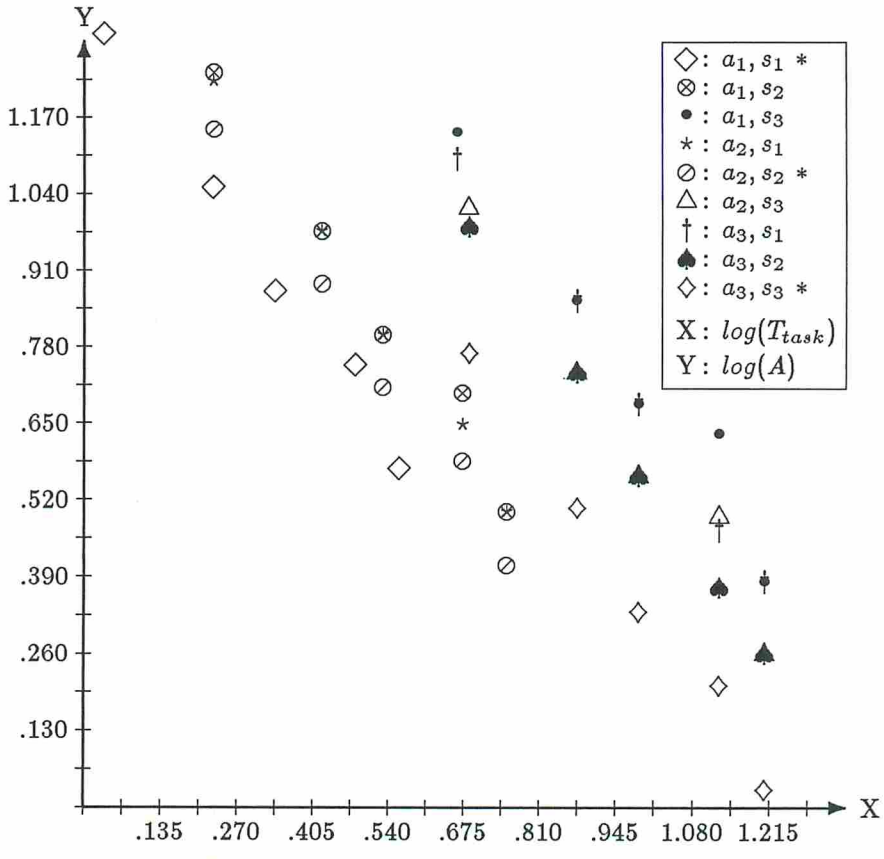


Figure 3.15: Conditional Example - 20 % Resynchronization

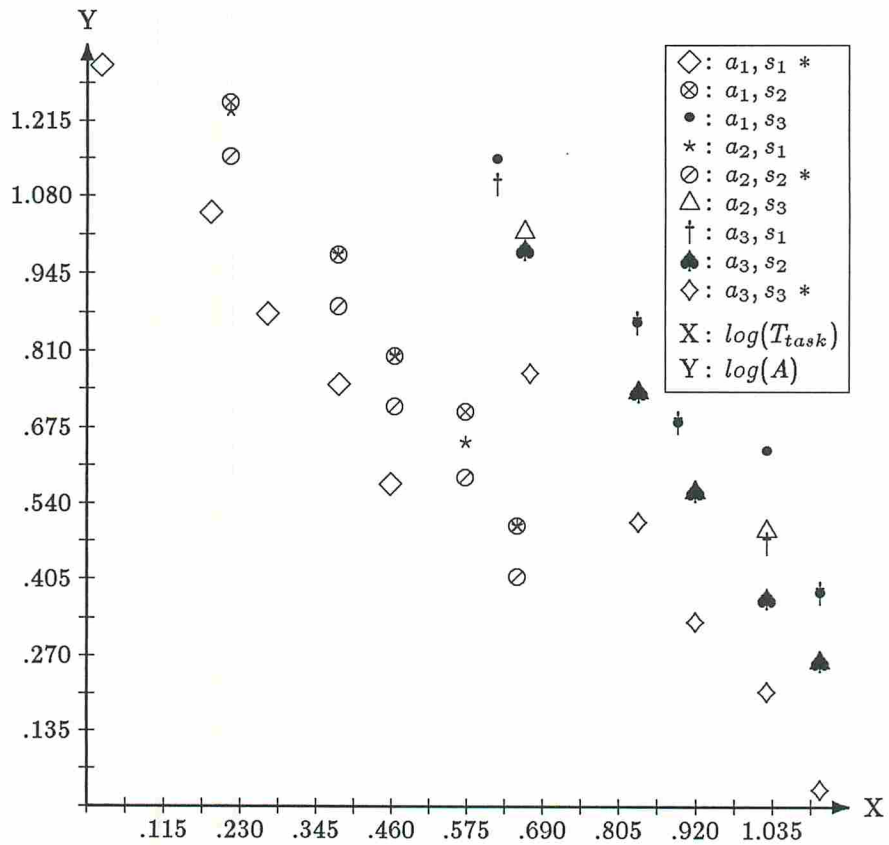


Figure 3.16: Conditional Example - 30 % Resynchronization

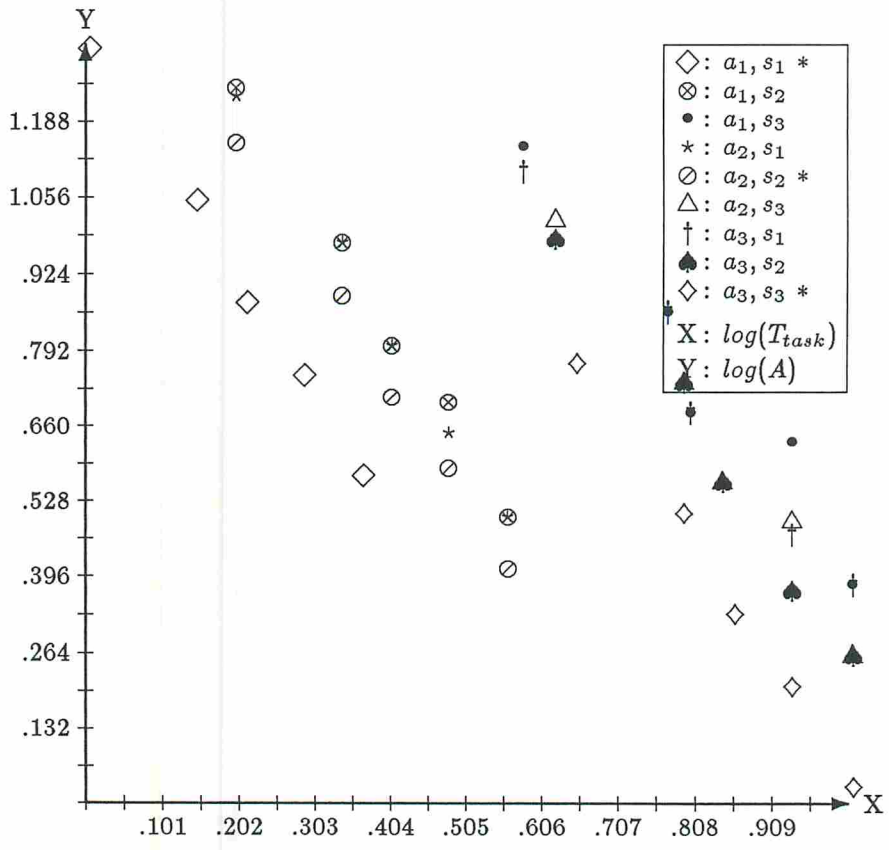


Figure 3.17: Conditional Example - 40 % Resynchronization

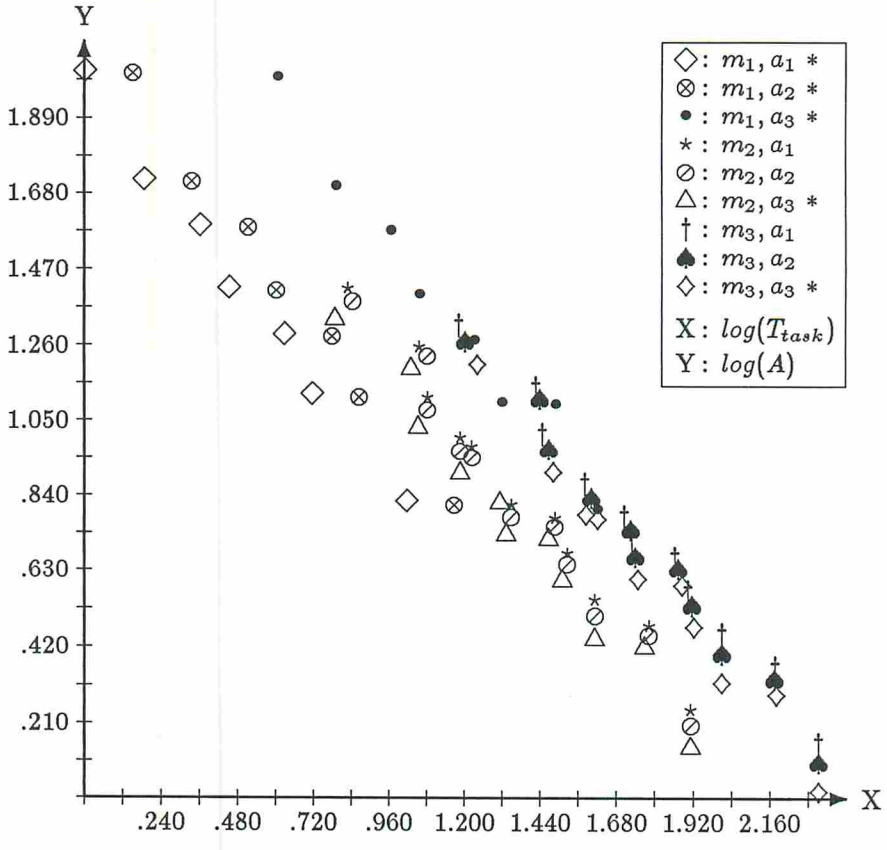


Figure 3.18: AR Filter - 10 % Resynchronization

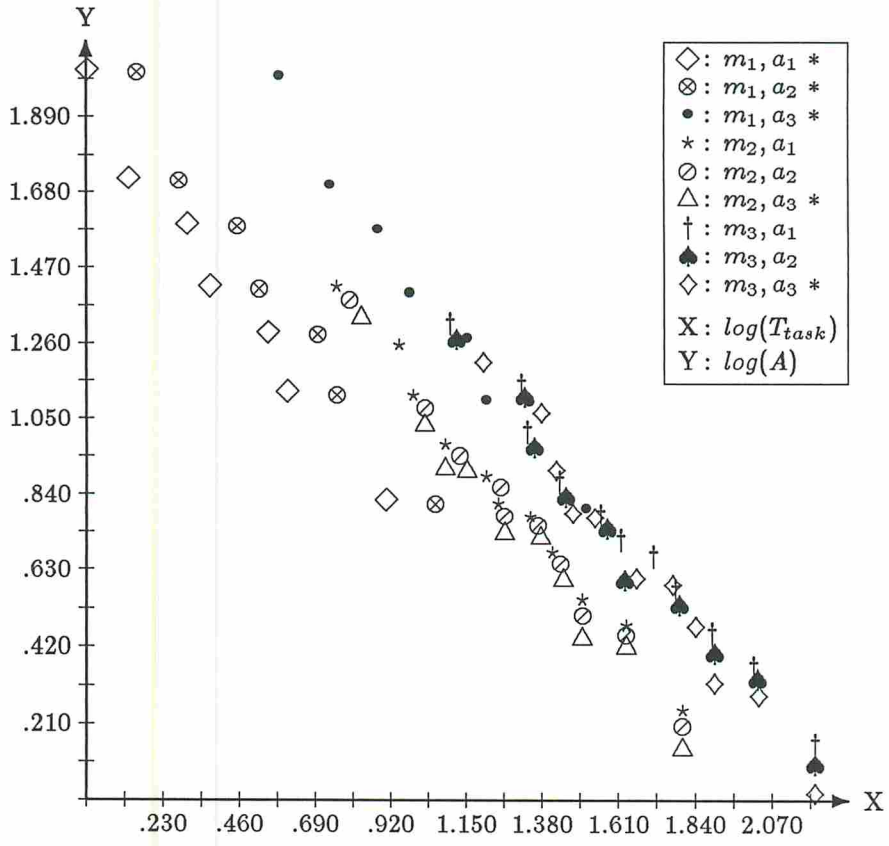


Figure 3.19: AR Filter - 20 % Resynchronization

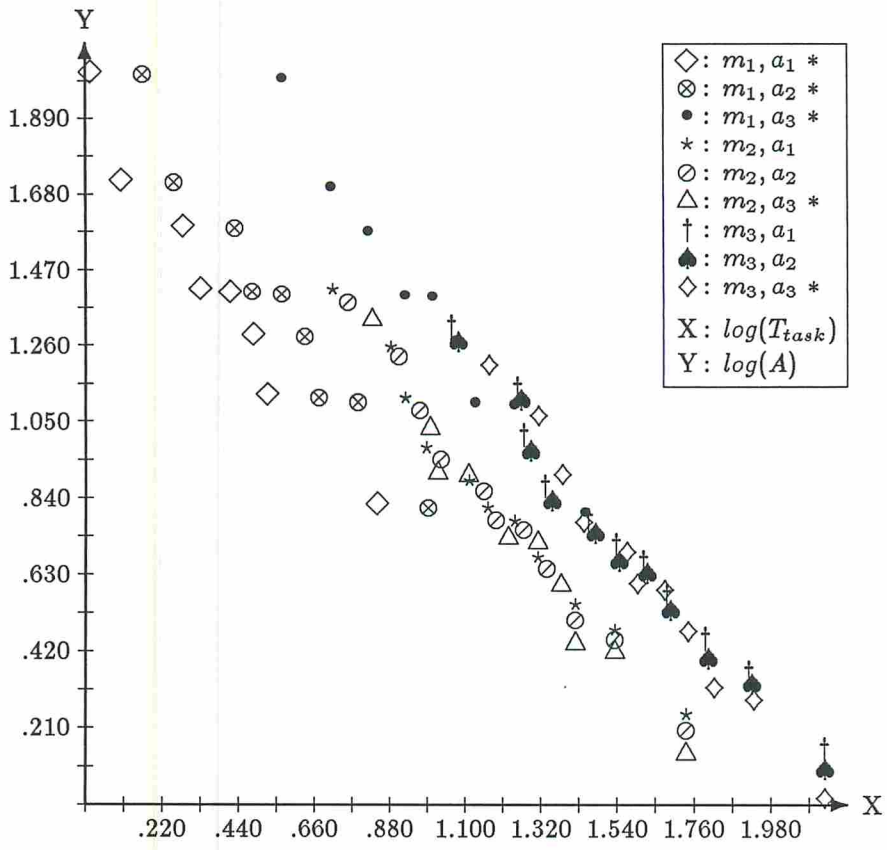


Figure 3.20: AR Filter - 30 % Resynchronization

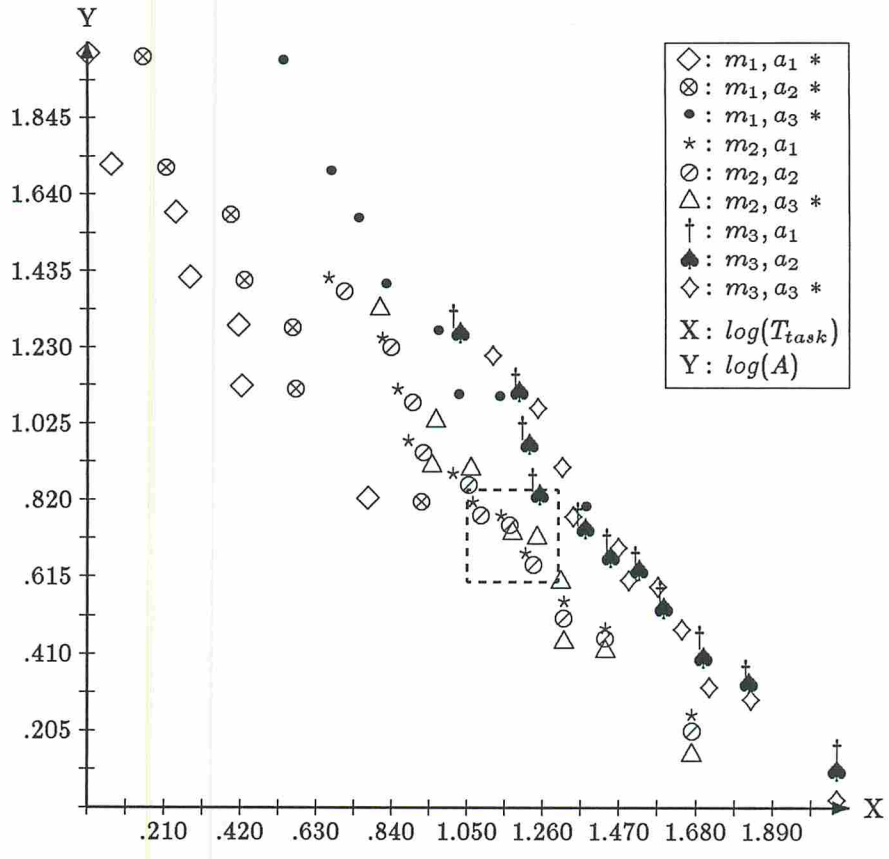


Figure 3.21: AR Filter - 40 % Resynchronization

Associated with every module set is a clock cycle for module-optimal designs. The value of clock cycle c_p is determined by the existing modules in the library. The module generator can be directed to generate a module for every operation type with delay constraint of c_p and optimizing function of minimizing area. The module generator will generate a module with a delay less than or equal to c_p and with minimum area; in doing so it may generate a module smaller than the existing modules. Such directives to the module generation help in exploring a larger design space.

For example, suppose the library contains only two adders, namely, the 16-bit ripple carry adder with a delay of 50 ns and the very fast 16-bit carry look-ahead adder with a delay of 10 ns. Given a data flow graph, the module selection program generates the best possible module set with some value of $c_p = 30$ ns. This module set contains the 16-bit adder with 16-bit carry look-ahead, as the ripple carry adder does not satisfy the clock cycle constraint. If the module generator is asked to generate an adder module with a delay constraint equal to 30 ns, the module generator may generate a 16-bit adder with 4-bit carry look-ahead and a delay of 27 ns. From the second assumption made in Section 2.1.1, we know that the area of the new adder is less than the area of the 16-bit adder with 16-bit carry look-ahead. Hence using the new adder for the implementation leads to an area efficient design without sacrificing the performance, as can be seen in Figure 3.22. The solid curve corresponds to designs generated by the existing module set using a 16-bit carry look-ahead adder, and the dashed curve represents designs using the new area-efficient module set. The initiation delays of the designs with the two module sets is the same; it is only the area which has changed.

From the module selection procedure, the value of the clock cycle is chosen such that corresponding to every operation type there exists at least one module in the library whose delay is less than or equal to the clock cycle. As such, even if the module generator returns an error (indicating that it failed to generate a module meeting the design constraint and/or optimizing function), at least one

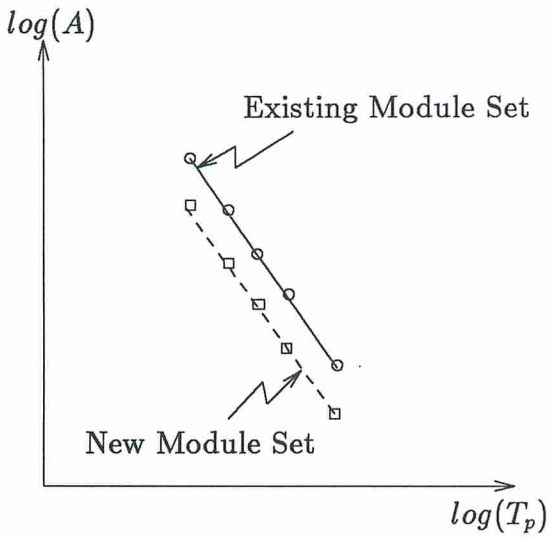


Figure 3.22: Module Generation

module exists which can be used in the implementation. The module generator is required only in cases where the library does not contain every operation type.

3.8 Module Selection Given Design Curves of the Modules

In the discussions so far, we have assumed that the design library consists of components of every type. Components of the same type, for example, a_1, a_2 and a_3, span the design space for 16-bit adders. In reality there can be several other 16-bit adder modules which could lie in between the design points representing a_1, a_2 and a_3. In fact the set of all possible 16-bit adders may form a design curve by itself. Figure 3.23 shows example design curves for four different module types. The question is how to perform module selection given design curves instead of individual modules for every operation type. In this case module selection produces the area-delay parameters of the best module choice. We provide a solution to this problem, given an initiation delay constraint, in this section.

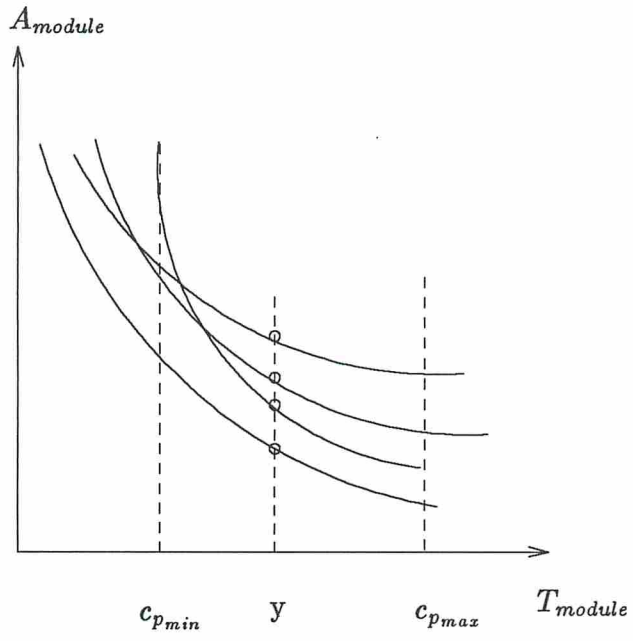


Figure 3.23: Design Curves for Different Module Types

Let us consider a data flow graph with four operation types and an initiation delay constraint of v. The design curves for the modules which can implement these four operations are given in Figure 3.23. From the given design curves of the various modules we can conclude that the design space is bounded by

1. the fastest design which will have a clock cycle of $c_{p_{min}}$ and to satisfy minimum initiation delay $c_{p_{min}} \leq v$, and

2. the smallest design (one module for each operation type) which will have a clock cycle of $c_{p_{max}}$ and minimum initiation delay of $c_{p_{max}} \times maximum(l)$ $= c_{p_{max}} \times maximum(n_i)$.

There are two levels of optimization tasks which need to be performed:

1. determination of the largest possible clock cycle which will satisfy the constraint and minimize the area of the designs, and

2. generation of modules of different types which will minimize area for the selected clock cycle.

The answer to the second optimization is easily found by Theorem 3.1. From Theorem 3.1 we know that the most area-efficient design for a given clock cycle is one which contains modules with delay equal to the clock cycle. The best modules for an example clock cycle of y which produce area-efficient designs are shown as circles in Figure 3.23.

To optimize the first goal we note that clock cycle can take any value between $c_{p_{min}}$ and $c_{p_{max}}$. There may be numerous clock cycles which lie between $c_{p_{min}}$ and $c_{p_{max}}$ and satisfy the constraint, but only a subset of these will minimize the area constraint.

Any clock cycle between the values $c_{p_{min}}$ and $c_{p_{max}}$ can be used for implementation and the area for any of these clock cycles can be minimized by selecting modules with delay exactly equal to that clock cycle. If we can identify the best possible clock cycles, then we know exactly which modules should be used for implementation. The initiation delay constraint of v can be exactly met by initiation interval of one and clock cycle of v, or by initiation interval of two and clock cycle of $\frac{v}{2}$ and so on. The maximum possible value of initiation interval is bounded by $maximum(n_i)$. In the extreme case, the initiation delay constraint of v can be met by a initiation interval of $maximum(n_i)$ and a clock cycle of $\frac{v}{maximum(n_i)}$. Thus, the set of possible values of clock cycles is $(v, \frac{v}{2}, \cdots, \frac{v}{maximum(n_i)})$. There are numerous other possibilities, including all clock cycles between v and $\frac{v}{2}$. These clock cycles between v and $\frac{v}{2}$ will produce designs with initiation interval of one and will be more expensive than the design produced by clock cycle of v and initiation interval one. Thus, all such clock cycles are rejected.

All clock cycles which do not lie between the limits of $c_{p_{min}}$ and $c_{p_{max}}$ are eliminated from the set of possible candidates. Corresponding to every initiation interval, $1 \leq x \leq maximum(n_i)$ there is a clock cycle, given by $\frac{v}{x}$ which can exactly meet the initiation delay constraint. For every possible value of clock

1. read initiation delay constraint v.
2. $l_{max} = maximum(n_i)$.
3. A_{min} = a very large number.
4. for $i = 1$ to l_{max}
5. if $(c_{p_{min}} \leq \frac{v}{i} \leq c_{p_{max}})$ {
6. $area[i] = \sum_{j=0}^{m-1}(a_j \times \lceil \frac{n_j}{i} \rceil)$.
7. if $(A_{min} > area[i])$ $A_{min} = area[i]$.

   }
8. for $i = 1$ to l_{max}
9. if $(area[i] == A_{min})$ add i to set of candidate initiation intervals.
10. print the candidate initiation intervals l and their associated clock cycles $\frac{v}{l}$.

Figure 3.24: Algorithm 5

cycle $\frac{v}{x}$ select the modules which have minimum value of $\sum_{i=0}^{m-1}(a_i \times \lceil n_i/x \rceil)$. This will eliminate all initiation intervals for which module-optimal designs cannot be generated. Further, as the design curves for modules intersect, some module-optimal designs may be more expensive than others and these are also rejected from the set of candidate solutions. The remaining set of clock cycles is the set of possible solutions.

Algorithm 5 shown in Figure 3.24 uses the technique described above to generate the set of best possible clock cycles and their corresponding initiation intervals. It is assumed that $c_{p_{max}}$ and $c_{p_{min}}$ are supplied by the designer who specifies the design curves for each module type. The user then has a choice of selecting the desired data initiation interval and its corresponding clock cycle for implementation.

3.8.1 Area Constraint

Several designs which satisfy a given area constraint u exist . However, only a small subset of these designs will minimize initiation delay. In this section we formulate the problem, without providing the solution.

We know that $u \geq \sum_{i=0}^{m-1}(a_i \times \lceil \frac{n_i}{l} \rceil)$ [3.6] For some fixed initiation interval $l = x$, we have

$$u \geq \sum_{i=0}^{m-1}(a_i \times \lceil \frac{n_i}{x} \rceil) \tag{3.8.10}$$

For minimizing initiation delay, $l \times c_p$ has to be minimized, and $c_p = d_i, \forall i$ [3.7]. Let us assume that for every module type i, $a_i d_i^{\alpha_i} = k_i$ (Kurdahi has fitted such equations for adders [Kur87]). Then, if $d_i = d_j \forall i \neq j$,

$$c_p = (\frac{k_i}{a_i})^{\frac{1}{\alpha_i}} = (\frac{k_j}{a_j})^{\frac{1}{\alpha_j}}, \forall i \neq j \tag{3.8.11}$$

To find the optimal clock cycles Equations 3.8.10 and 3.8.11 have to be solved for every initiation interval and finally the subset of these initiation intervals which minimizes initiation delay will be selected.

3.9 Effect of Register and Multiplexer Area on Module Selection

In this section we predict the effect of register and multiplexer area on design style selection. Prediction of register and multiplexer areas for pipelined and non-pipelined design styles is given in [MP88a]. There are two basic assumptions made in [MP88a] for predicting the register and multiplexer areas. One assumption is that incoming edges do not need registers in non-pipelined designs.

[3.6] If $u < \sum_{i=0}^{m-1} a_i'$, where a_i' is the area of the cheapest module type for operation i, then the area constraint cannot be satisfied.

[3.7] Since in the module set with minimal delay all modules have identical delays (Theorem 3.1).

For pipelined design, the register count is given by

$$R_p = \frac{1}{2}\left[(max(E_{in}, E_{out}) + (u-1)R_{pmax}) + (min(E_{in}, E_{out}) + (u-1)R_{pmin})\right] \tag{3.9.12}$$

where, E_{in} is the number of incoming edges (edges from the source), E_{out} is the number of edges going into the sink ($R_{pmin} = E_{out}$), $u = \lceil \frac{N}{l} \rceil$, R_{pmax} is the maximum cut set of the data flow graph (including the source of the data flow graph), and R_{pmin} is the minimum cutset of the data flow graph. If the number of incoming edges from the source E_{in} is equal to the maximum cutset R_{pmax}, and the outgoing edges to the sink E_{out} is equal to the minimum cut set R_{pmin} the above equation has been shown to reduce to

$$R_p = \frac{u}{2}(E_{out} + E_{in}) \tag{3.9.13}$$

For the AR filter data flow graph (Figure 1.2) $E_{out} = R_{pmin} = 2$ and $E_{in} = R_{pmax} = 26$ and using Equation 3.9.13 we have $R_p = 14u$.

In [MP88a], it has been shown that the multiplexer count for the pipelined design style is proportional to

$$M_p \propto R_p, u \tag{3.9.14}$$

Since the register and multiplexer counts do not depend upon the module style, we can assume that the relative spacing of area-delay curves for different module sets still remain the same, only the origin has shifted. Thus, the module selection solution is not affected by the register and multiplexer area.

3.10 Complexity Analysis

Let the total number of modules in the library be $k = \sum_{i=0}^{m-1} p_i$ and $h = maximum(p_i)$.

3.10.1 Time Complexity of Algorithm 1

Figure 3.25 gives an implementation of Algorithm 1 (Figure 3.5). The following data structures are required for the implementation.

1. An array of size k (in the worst case) called $CLOCKCYCLE$ which stores in ascending order all possible values of clock cycle. These values are obtained by taking all delays of all modules, sorting them, and deleting all duplicate entries.

2. For every operation type i a vector of size p_i called $MODULEID[i][j]$, where $0 \leq j \leq (p_i - 1)$ is defined. This array stores the modules which implement operation i in ascending order of their delays. Actually a module identification token is used which can directly reference the module parameters. For example, $MODULEID[i][0] = 2$ refers to $ar_{i,2}$ and $dl_{i,2}$. Also, as $j = 0$ this module has the least delay amongst all modules implementing operation i.

3. For every operation i we have a pointer which points to the current position in the $MODULEID[i]$ vector. The module of operation type i which is referenced by this pointer $PTR[i]$ has a delay of less than or equal to the current clock cycle and minimum area.

To evaluate the computational complexity of Figure 3.25 we shall consider the dominating steps only. The dominant steps are those which are likely to contribute to the overall complexity of the algorithm. Algorithm 1 is a sorting step. Sorting of x elements can be accomplished in $O(x(logx))$ time. As there are k elements in array $CLOCKCYCLE$ this step takes $O(k(logk))$ time. Step 2 is also a sorting step. However, in this step p_i elements are sorted in the i^{th} iteration. Thus the complexity of the i^{th} iteration is $O(p_i(logp_i))$. In the worst case the complexity of Step 2 over all the iterations is $O(mh(logh))$.

We shall now compute the complexity of the assignment statement in Step 8. We know that $CLOCKCYCLE[i] < CLOCKCYCLE[i+1]$. Thus, if in the first iteration of Step 4, if for some operation j we have chosen module x for implementation such that $dl_{jx} < CLOCKCYCLE[i]$, then in the subsequent iterations we only need to consider modules of operation type j with delay greater than or equal to dl_{jx}. This implies that the assignment statement in Step 8 is

1. Sort the delays of all modules in the library in ascending order and store the result in an array $CLOCKCYCLE[i]$ where $0 \leq i \leq (k-1)$. Delete all duplicate values.
2. For $i = 0$ until $i = m - 1$ {
   Sort modules of type i in ascending value of dl_{ij} and place the sorted result in a matrix $MODULEID[i][j]$, where $0 \leq j \leq (p_i - 1)$.
   }
3. For $i = 0$ until $i = (m-1)$ { $PTR[i] = 0$. } /* Initialize pointers. */
4. For $i = 0$ until $i = size(CLOCKCYCLE) - 1$ {
   /* For every possible value of clock cycle. */
5. $c = CLOCKCYCLE[i]$.
6. For $j = 0$ until $j = m - 1$ { /* For every operation type. */
7. if $(dl_{j,MODULEID[j][PTR[j]]} > c)$ {
   print(This clock cycle not possible).
   go to Step11.
   }
8. while $(dl_{j,MODULEID[j][PTR[j]+1]} < c)$ $PTR[j] = PTR[j] + 1$.
9. $a_j = ar_{j,MODULEID[j][PTR[j]]}$.
   } /* End of j loop */
10. Append this module set to X.
11. } /* End of i loop */

Figure 3.25: Implementation of Algorithm 1

independent of the number of iterations of Step 4 and Step 6 and depends only on p_i for each operation type. Furthermore, it will be executed a maximum of p_i times for each operation. Thus the assignment statement in Step 8 will be executed $O(k)$ times.

Step 9 will be executed $O(km)$ times. Thus the overall complexity of the algorithm is [3.8],

$$O(k(logk) \oplus mh(logh) \oplus km) \tag{3.10.15}$$

3.10.2 Time Complexity of Algorithm 2

Complexity for Algorithm 2 (Figure 3.7) can be easily computed. In the worst case a maximum of k module sets (one for every possible value of the clock cycle) will be generated. Steps 1 and 2 can be computed in $O(m)$ time for each module set and will execute in $O(km)$ time for k module sets. Steps 4 and 6 will execute in $O(k)$ time. The complexity of Algorithm 2 is

$$O(km) \tag{3.10.16}$$

3.10.3 Time Complexity of Algorithms 3 and 4

Algorithm 3 (Figure 3.9) is a searching algorithm. It starts the search at a particular module set $j \in S$ found by Algorithm 2 and then searches the adjacent module sets until the best possible module set is found. In the worst case, the search may go through all the possible module sets in set S, and hence the number of iterations this algorithm will perform is k. However, in an average case, the search should not go beyond a few module sets, and the best possible result will be found very quickly. It is hard to estimate the average complexity of the algorithm, as it depends on the data flow graph and the non-optimal design points of the data flow graph. The design points of a data flow graph for a given module set can be found by the algorithm given in Figure 2.1 in $O(m(maximum(n_i)))$ time, where n_i is the number of nodes of operation type i

[3.8] A complexity of $O(x \oplus y)$ implies $O(maximum(x, y))$.

in the data flow graph. Step 1 of Algorithm 3 is a sorting step and can be done in $O(k(logk))$ time. Hence the worst case time complexity of Algorithm 3 is

$$O(k(logk) \oplus mk(maximum(n_i))) \tag{3.10.17}$$

Time complexity of Algorithm 4 (Figure 3.13) can be found similarly and is given by $O(mk(maximum(n_i)))$.

3.10.4 Time Complexity of the Module Selection Algorithm

The overall complexity of the whole procedure of module selection which consists of executing Algorithms 1, 2 and 3 in sequence can be computed by summing the Expressions 3.10.15, 3.10.16, and 3.10.17,

$$O(k(logk) \oplus mh(logh) \oplus km \oplus km \oplus k(logk) \oplus km(maximum(n_i)))$$

$$= O(k(logk) \oplus mh(logh) \oplus km(maximum(n_i)) \tag{3.10.18}$$

We know that $m \leq k$ and $h \leq k$, simplifying Expression 3.10.18 to $O(k^2(maximum(n_i)))$. Thus, we see that worst-case time complexity of the complete module selection solution for pipelined designs is polynomial.

3.11 Summary

In this chapter we have developed a theoretical model for the module selection problem for pipeline designs. The model we presented is general and applicable to any pipeline synthesis tool with the assumptions made. Based on this theoretical model, we developed a set of polynomial-time algorithms which perform selection of an optimal set of modules for a given pipeline design. These algorithms select an optimal set of modules from a module library for the implementation of a pipelined design with a design constraint. By predicting an optimal set of modules within a given design constraint, pipeline synthesis tools

can produce an optimal pipeline implementation quickly. It has been shown that using these algorithms reduces the design time significantly.

These algorithms allow pipeline synthesis packages to use a module library instead of a fixed set of modules during high-level synthesis. These algorithms are implemented in a program called SLIMOS. SLIMOS is programmed in C and produces optimal results with small resynchronization rates and good results with high resynchronization rates. SLIMOS runs on a SUN 3/280 workstation and took less than 60 *ms* of cpu time to find the best module set meeting the constraint and satisfying the objective function for all experiments described here.

Chapter 4

Lower-Bound Area-Delay Tradeoff Curve for Non-Pipelined Designs

4.1 Introduction

This chapter describes an area-delay estimation model for non-pipelined designs. Predicting the lower-bound area-delay tradeoff curve is performed using a mathematical model for the curve. The model is developed in two stages. In the first stage we identify the clock cycle for the lower-bound design curve and in the second stage we use the clock cycle computation to compute the number of operators required for the lower-bound design. Collectively, these form the lower-bound area-delay curve. Our prediction tool is able to predict an area-delay curve in the design space which forms a lower bound for all possible designs; that is, all the design points lie either on or above the curve; design points which lie on the curve represent optimal designs. The lower bound area-delay tradeoff curve has the shape $AT_{np} = c_{np} \times k$ where k is a constant and c_{np} is the clock cycle. The delay axis represents the total operator time to execute the required behavior (called circuit delay), and the area axis represents the area of the operators used

to implement the required behavior. The chapter concludes with the results of some experiments carried out to verify the model.

The preprocessing steps stated in Section 2.1.1 are also applicable to the non-pipelined area-delay prediction technique.

4.2 Clock Cycle Prediction

Prior to predicting the AT_{np} curve itself, a lower bound on the value of the clock cycle which is used for the lower bound predictions must be derived. Since this is a lower bound, latching delays are not included in clock cycle computations. The clock cycle depends on the module delays (d_i), the delay along the critical path C, and the number of time steps N the data flow graph is partitioned into. The clock cycle computation is used to derive the lower-bound area-delay curve. The clock cycle is first derived and then substituted in the final area-delay equation to give the complete model. The number of partitions the graph is divided into is enumerated from 1 to the total number of operations of all types in order to plot the AT_{np} curve, and, like the clock cycle, does not need to be specified a priori.

Theorem 4.1 *The lower-bound clock cycle c_{np} which produces the best possible designs from a given data flow graph and a module set is*

$$c_{np} = maximum(\frac{C}{N}, maximum(d_i)) \tag{4.2.1}$$

where C is the critical path delay, the data flow graph is divided into N time steps and d_i is the delay for each module type i.

Proof: When $N = 1$, then the clock cycle will be the critical path delay, C. If the circuit is partitioned into two time steps, then the best case is when the clock cycle is equal to exactly half the critical delay. Similarly for $N = 3, 4, \cdots$ the best possible value of clock cycle will be $C/3, C/4, \cdots$ respectively. However, as every operation must complete within one clock cycle, the minimum clock cycle

is equal to the delay of the operator with maximum delay. This completes the proof. ■

For example, consider Figure 1.2. If the delays of the adder and multiplier are 250 ns and 500 ns respectively, then the critical path delay $C = 2750$ ns, and $c_{np} = maximum(2750/N, maximum(250, 500)) = maximum(2750/N, 500)$. If this data flow graph is partitioned into five or fewer time steps, then $c_{np} = 2750/N$. For $N \geq 6$, the clock cycle is equal to $maximum(250, 500) = 500$ ns. The break point for the clock cycle occurs when the number of time steps $N = 2750/500 = 5.5$.

If the delays of all the modules in the module set are equal, then the break point will occur when the number of time steps is equal to the number of nodes in the critical path (in this example data flow graph there are eight nodes in the critical path). This observation can be easily deduced from Equation 4.2.1 as follows. In this special case $d_i = d_j$, and $C = x \times d_i$, where x is the number of nodes in the critical path. Thus, $c_{np} = maximum(\frac{xd_i}{N}, d_i)$, and the break point will occur at $x = N$. Apart from this special case, the number of nodes in the critical path cannot be used to compute the break point.

4.3 Lower-Bound Area-Delay Tradeoff Curve

With a formula for the lower bound on the clock cycle time, the minimum number of modules required for each value of time step N can be readily determined. These results are combined to form the area-delay tradeoff curve for non-pipelined designs. Before explaining how to compute this curve, we provide the following definitions of utilization and module-optimality. The underlying concept of module-optimality has been taken from [Gir84].

Definition 4.1 *The utilization of each module type* $0 \leq i \leq m-1$ *is defined as*

$$u_i = \frac{n_i}{N \times o_i} \tag{4.3.2}$$

where n_i is the number of nodes of type i in the data flow graph, o_i is the number of modules of type i used in the design and m is the number of types of operations in the data flow graph.

For an operation type, a utilization of one is *module-optimal.*

Definition 4.2 *If $u_i = 1$ for all operation types, $0 \leq i \leq m-1$ then the non-pipelined implementation is a module-optimal design.*

For example consider Figure 1.2 where $n_{addition} = 12$ and $n_{multiplication} = 16$. If this data flow graph is implemented with four adder modules and five multiplier modules in four time steps ($N = 4$), then $u_{addition} = \frac{12}{4\times4} = 0.75$ and $u_{multiplication} = \frac{16}{5\times4} = 0.8$. If the implementation with four time steps uses three adders and four multipliers then $u_{addition} = u_{multiplication} = 1$ and the design is module-optimal.

Of course, in practical designs, it is often not possible to utilize all the modules in every cycle, resulting in suboptimal designs. The following theorem uses the above definition to provide a basis for our prediction technique.

Theorem 4.2 *Given a data flow graph and a module set, the lower-bound area-delay tradeoff curve of non-pipelined designs is given by*

$$AT_{np} = c_{np} \times k \tag{4.3.3}$$

where c_{np} is the clock cycle for the design and k is a constant.

Proof: From Equation 4.3.2 module-optimal designs satisfy

$$N = \frac{n_i}{o_i}$$

Multiplying both sides by $a_i o_i$ and summing over all m operation types yields

$$N \sum_{i=0}^{m-1} (a_i \times o_i) = \sum_{i=0}^{m-1} (a_i \times n_i)$$

Multiplying both sides by the clock cycle time c_{np} gives

$$c_{np}N \sum_{i=0}^{m-1}(a_i \times o_i) = c_{np} \sum_{i=0}^{m-1}(a_i \times n_i)$$

Substituting for the definition of area and delay results in

$$AT_{np} = c_{np} \sum_{i=0}^{m-1}(a_i \times n_i)$$

Noting that for a given data flow graph and module set, a_i and n_i are constants, so we get Equation 4.3.3. ■

By combining Equations 4.2.1 and 4.3.3, we have

$$AT_{np} = k \times maximum(\frac{C}{N}, maximum(d_i)) \tag{4.3.4}$$

where C and $maximum(d_i)$ are constants for a given data flow graph and module set. Equation 4.3.4 can be split into following two parts.

(i) For $maximum(d_i) < \frac{C}{N}$,

$$AT_{np} = \frac{kC}{N} \tag{4.3.5}$$

(ii) For $maximum(d_i) \geq \frac{C}{N}$,

$$AT_{np} = k \times maximum(d_i) \tag{4.3.6}$$

Equation 4.3.5 is graphically represented by the circles in Figure 4.1a. (The graph is drawn on a *log* - *log* scale for better readability.) Each circle corresponds to some value of N. For example, the top-most circle corresponds to $N = 1$, for which $AT_{np} = kC$. Equation 4.3.6 is shown by the solid sloping line. Joining the circles with the firm line for $T_{np} > C$ gives the final lower-bound area-delay curve for the data flow graph and is redrawn in Figure 4.1b. Designs with clock cycle equal to C/N have identical lower-bound delay T_{np} and their area A decreases as N increases. What differentiates these designs in practice are latching delays, which increase as the number of time steps increases, and delays due to dead time, since operator delays may not exactly match the clock cycle. Thus, the lower-bound delays for these designs are rarely achieved.

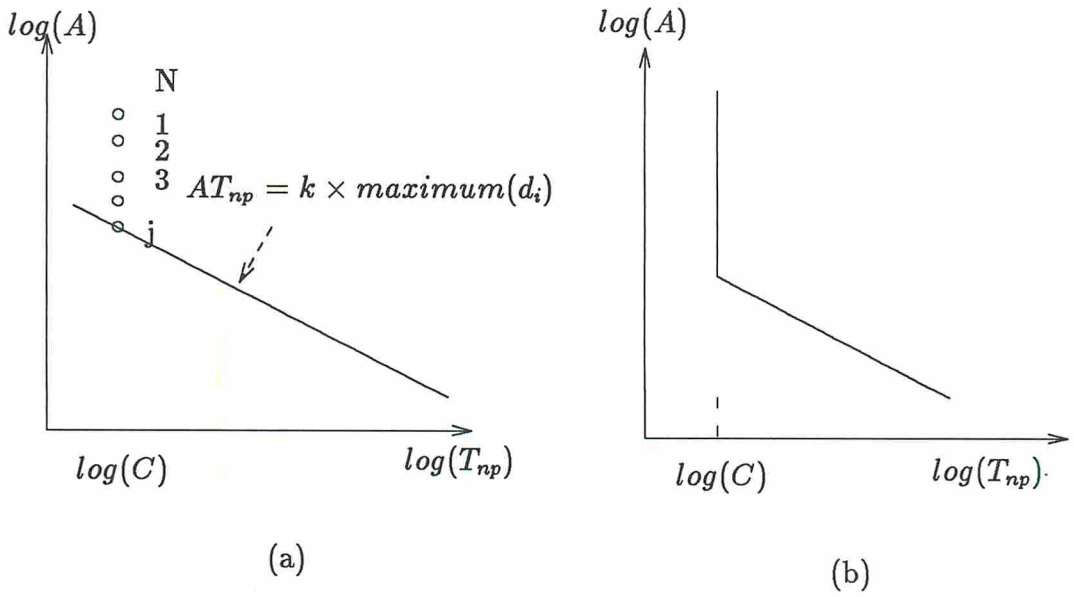


Figure 4.1: Area-Delay Estimation Curve

4.3.1 Non-Optimality of Practical Designs

Practical designs are often non-optimal and these design points have an AT_{np} product inferior to the lower-bound design points. There are several reasons for non-optimality, some of which are illustrated here. In Figure 4.2a, the delay along the critical path is $C = 3d_{adder}$. If the data flow graph is partitioned into two time steps, then the lower-bound clock cycle is $c_{np} = \frac{3}{2}d_{adder}$, which is not possible based our assumptions. The actual clock cycle time is the ceiling function $\lceil n_i/N \rceil = \lceil 3/2d_{adder} \rceil = 2d_{adder}$ which degrades the performance from that predicted using the lower bound clock cycle. For any given design, this loss in performance for non-optimal designs is bounded by $\Delta c_{np} = \frac{1}{2}maximum(d_i)$.

The actual procedure which generates the lower bound area-delay curve for a given data flow graph and module set is given in Figure 4.3. This procedure takes into account the module non-optimality of designs by computing the ceiling function $\lceil n_i/N \rceil$ rather than n_i/N. The procedure is executed for different time steps, $1 \leq N \leq \sum_{i=0}^{m-1} n_i$.

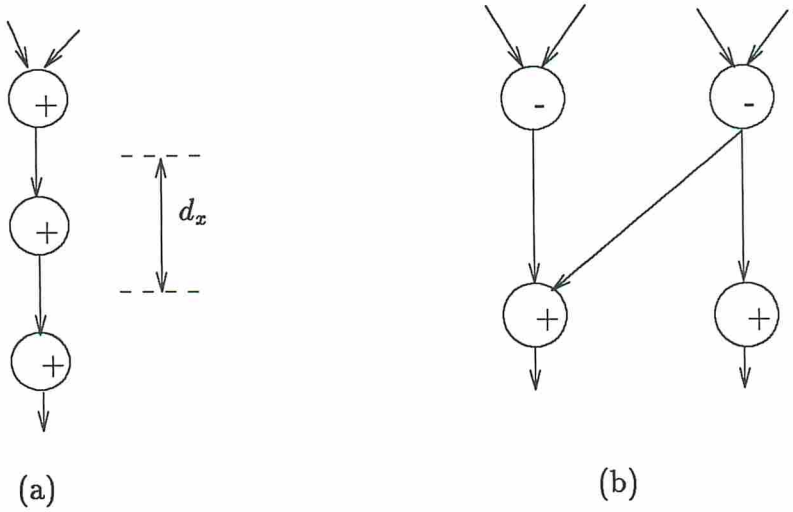


Figure 4.2: Non-Optimal Design Examples

A non-optimal design also results when all the modules cannot be used every clock cycle. Continuing with Figure 4.2a, a design with two partitions will require two adders, of which one adder will lie idle for one clock cycle leading to a non-optimal design.

There is a third cause for non-optimality in non-pipelined designs: feasibility of scheduling. A lower-bound design with two partitions for Figure 4.2b would require one subtractor and one adder and the clock cycle would be half the delay of critical path. However, to meet the clock cycle requirements, the only possible partition is to put both the subtraction nodes in the first partition and both the addition nodes into the second partition. Clearly, this solution is infeasible as the first partition has two subtraction operations to be performed on one subtractor and two addition operations to be performed in one adder in the second partition. Four time steps will be needed to make the schedule feasible, reducing the performance of the design. This type of non-optimality can only be observed after scheduling has been performed and is not known *a priori.* [4.1]

[4.1] This type of non-optimality is not observed in pipelined designs where overlapped input data sets can keep the modules busy.

```
                procedure estimate_lower_bound(N)
                begin
1.                      c_1 = maximum(d_i);
2.                      c_2 = ⌈C/N⌉;
3.                      c_np = maximum(c_1, c_2);
4.                      T_np = N × c_np;
5.                      o_i = ⌈n_i/N⌉, 0 ≤ i ≤ m − 1;
6.                      A = Σ_{i=0}^{m−1} (a_i × o_i);
                        print A, T_np;
                end
```

Figure 4.3: Procedure to Compute Lower-Bound Area-Delay Curve for Non-Pipelined Designs

4.3.2 Complexity Analysis

In this subsection we derive the worst case runtime complexity of the algorithm for computing the lower-bound area-delay non-pipelined tradeoff curve (Figure 4.3). For every value of N, Steps 1, 5 and 6 take $O(m)$ time to execute. The maximum number of time steps can be $\sum_{i=0}^{m-1} n_i$. Thus, the overall complexity of the whole algorithm is

$$O(m \sum_{i=0}^{m-1} n_i) \tag{4.3.7}$$

If $M = \sum_{i=0}^{m-1} n_i$, then the complexity is $O(mM)$. That is, the predictive algorithm performs $O(mM)$ computations to compute the entire area-delay tradeoff curve. On the other hand, the scheduling algorithm of HAL is reported to have a complexity of $O(tQ^2)$ for each design point, where t is the time constraint and $Q = \sum_{i=0}^{m-1} q_i \geq M$ is the total number of operations in the data flow graph [Pau88, page 11], and $O(tQ^3)$ for the entire area-delay tradeoff curve.

Comparing the complexity of the prediction algorithm and the scheduling algorithms, we observe that the prediction tools are fast and can help guide the designer towards a good design quickly.

4.4 Experiments and Results

We present results for five example data flow graphs, the AR filter (Figure 1.2), and an elliptical wave filter (Figure 2.4), a conditional data flow graph (Figure 2.2) and finally the random data flow graph (Figure 2.3). We have used the module set (m_1, a_1, s_1) given in Table 1.1 for our experiments.

For each data flow graph, we used MAHA [PPM86] to synthesize all possible designs by varying cost and delay. The prediction technique described in this chapter was then applied onto all five data flow graphs. The results of the experiments with MAHA and the predictor are shown in Figures 4.4 through 4.7. These results have been normalized and plotted on a log-log scale for better readability.

Results for the elliptical wave filter produced by the predictor, and several other synthesis systems, MAHA, HAL [PK87], SPAID [HE88], EMUCS [TDW*88] and CATREE [GE88] are also compared here [4.2]. The results for HAL, SPAID, EMUCS and CATREE are in terms of *the number of time steps* versus *the number of modules of each type.* We plotted these results on a time step versus area graph, where the area was computed by multiplying the number of modules of each type with their corresponding area as given in Table 1.1. For these results, HAL, SPAID, and CATREE used two-time-step pipelined multipliers while MAHA and EMUCS used one-time-step non-pipelined multipliers. These results are plotted in Figure 4.8. These results strengthen our confidence in the capability of the prediction tool to be used with other similar synthesis programs. It is interesting to observe from the results that none of the synthesis tools, with

[4.2] Results for SPAID, EMUCS, HAL and CATREE are cited from [HE88].

the exception of MAHA, have been reported to explore the design region for fast designs.

Some results produced by BUD [McF86], another non-pipelined synthesis program, are given in [McF87]. The shape of the curve produced by BUD with module area-delay alone (Figure 3e of [McF87]) follows the shape of the curves produced by our prediction program. The values of the design points cannot be compared as the input description and the module sets used by BUD were different.

The impact of registers, multiplexers, busses, and wiring has not been included in the model described in this thesis. However, models for register, multiplexer, control and wiring area have been reported in [MP88a], [MP88c] and [KP86] respectively. By combining all the three models, as shown in Figure 1.6, we can obtain an overall area-delay tradeoff curve. Figure 4.9 shows two curves for the AR filter data flow graph. One curve shows the area-delay curve using the operator area-delay with the predicted register, multiplexer, and control areas added in. The second curve shows the actual synthesis results produced by the scheduler MAHA, module binder and register and multiplexer allocator MABAL, and the Berkeley PLA control generation programs (PEG [Ham83], EQNTOTT and ESPRESSO) along with PLA folding as given in [BB87]. These curves do not include the wiring area.

4.5 Summary

In this chapter we have derived the lower bound area-delay curve for non-pipelined designs. This model accepts a data flow graph and a module set. Given these inputs it predicts the operator-optimal designs which can be generated by a non-pipelined synthesis tool with the above mentioned assumptions. The results have been verified by a non-pipelined synthesis tool MAHA. The model predicts the tradeoff curve in polynomial time.

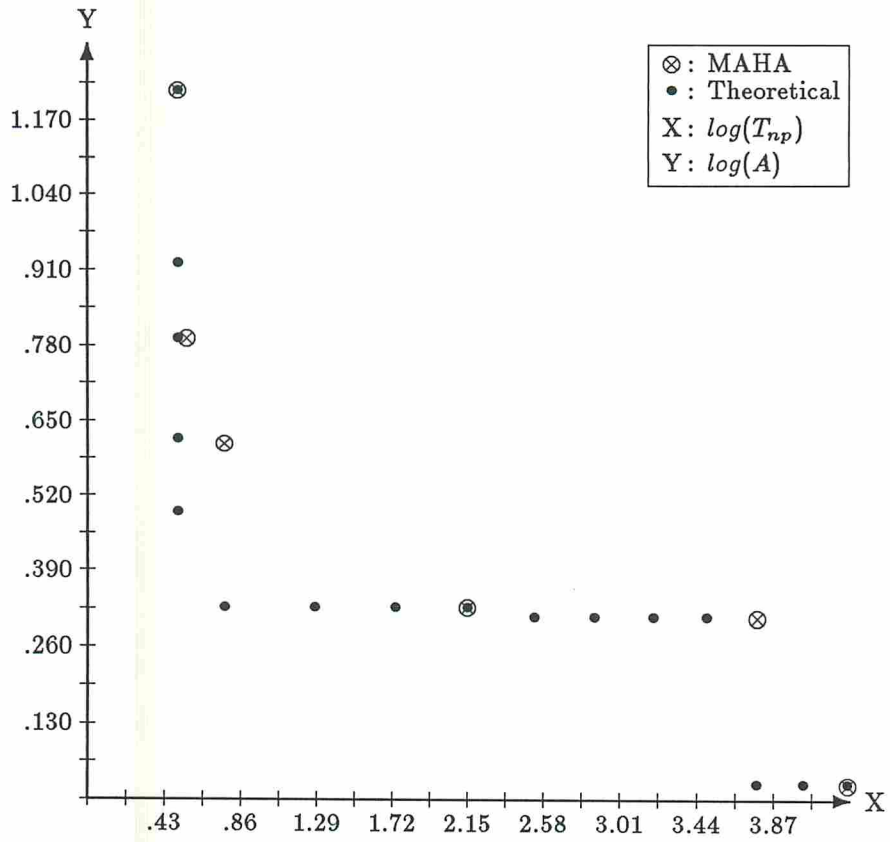


Figure 4.4: Area-Delay Curves For AR Data Flow Graph

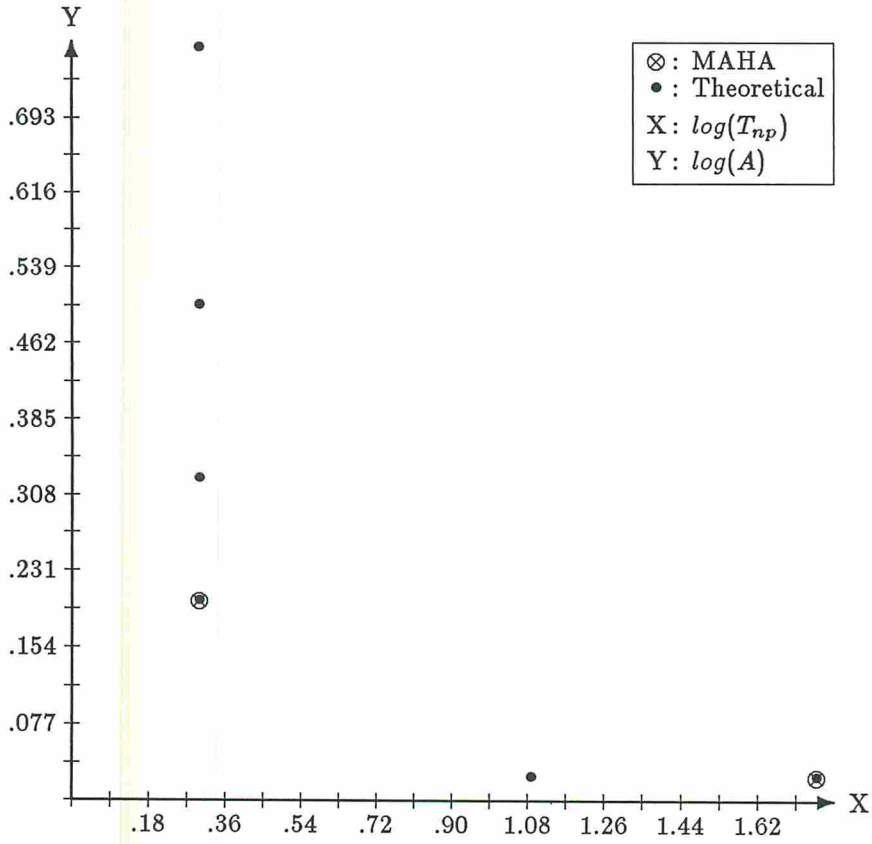


Figure 4.5: Area-Delay Curves For Conditional Data Flow Graph

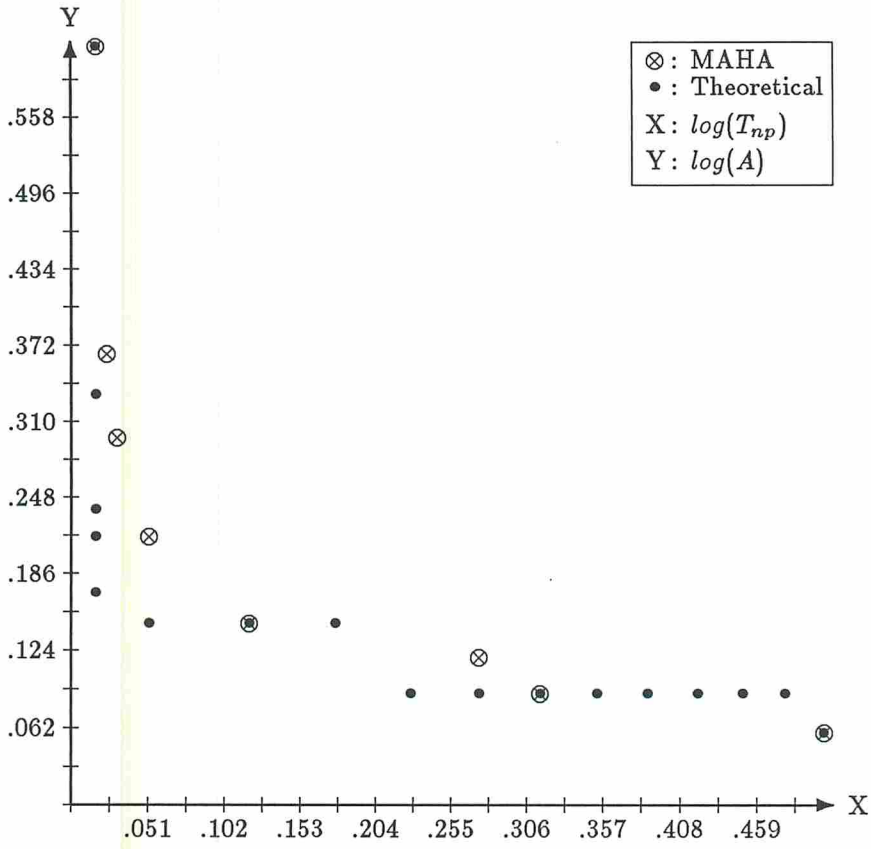


Figure 4.6: Area-Delay Curves For Random Data Flow Graph

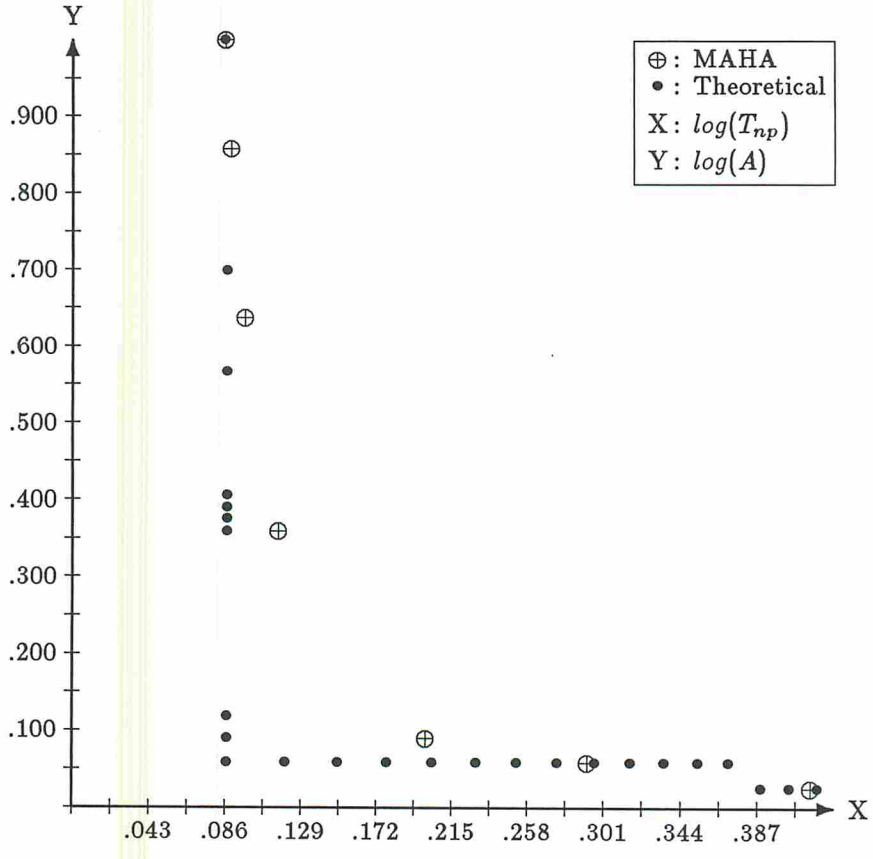


Figure 4.7: Area-Delay Curves For EW Filter Using MAHA

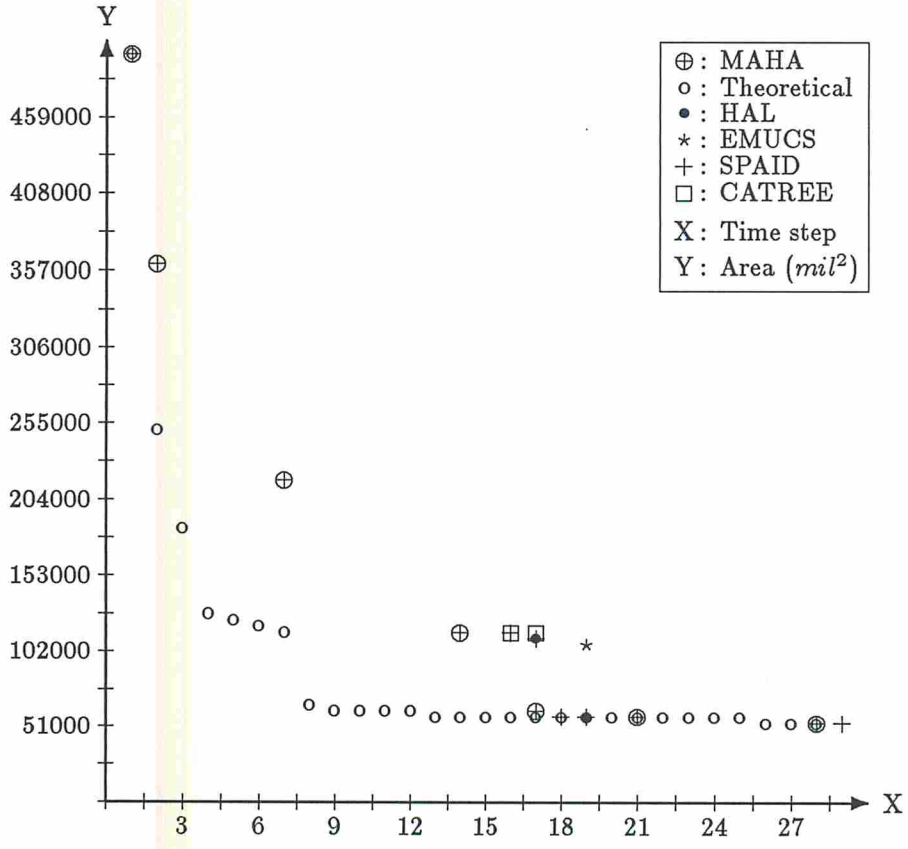


Figure 4.8: AT Curves For EW Filter Using Several Synthesis Systems

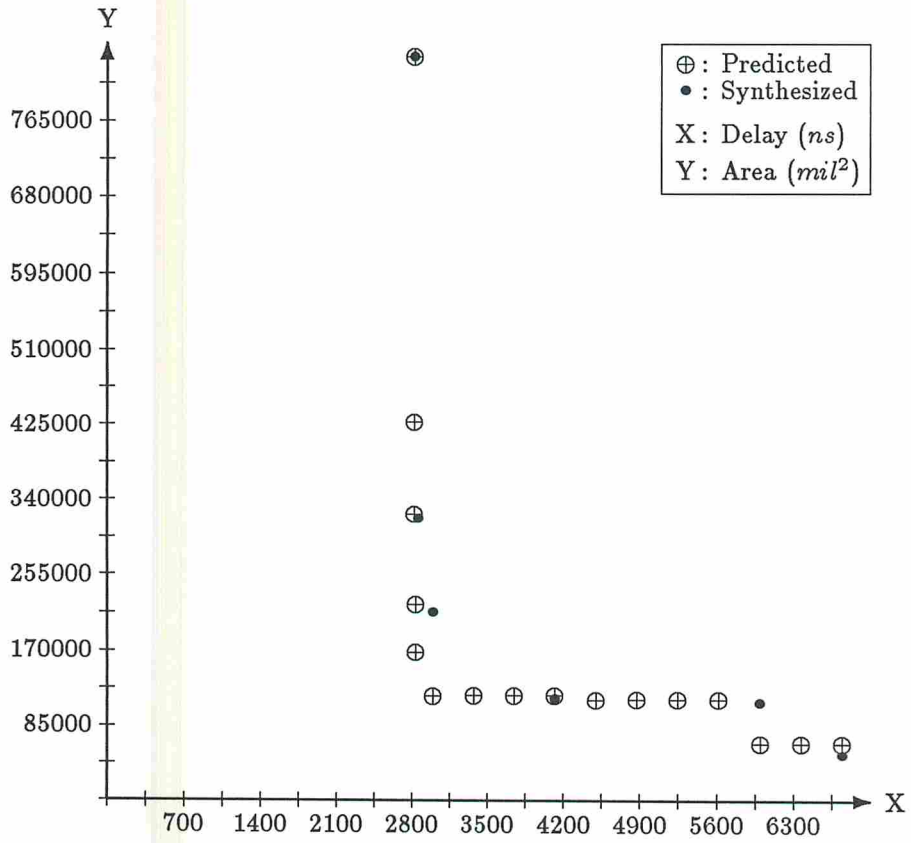


Figure 4.9: AT Curves for AR Filter

Chapter 5

Design Style Selection

5.1 Introduction

In this chapter the area-time estimation tools developed in Chapters 2 and 4 are used to solve the design style selection problem. The chapter is organized as follows. First we revisit the lower-bound area-time models for pipelined (Chapter 2) and non-pipelined (Chapter 4) synthesis. Next, we develop theoretical foundations for design style selection. Finally, we present results of experiments to verify the theory developed in this chapter.

5.1.1 Area-Delay Models Revisited

The model for pipelined designs predicts the lower-bound clock cycle c_p to be

$$c_p = maximum(d_i) \tag{5.1.1}$$

and the area-delay product to be

$$A \times T_p = c_p \sum_{i=0}^{m-1} (a_i \times n_i) = k_p \tag{5.1.2}$$

where d_i (a_i) is the delay (area) of the module which implements operation i, n_i is the effective number of operations of type i in the data flow graph, m is the

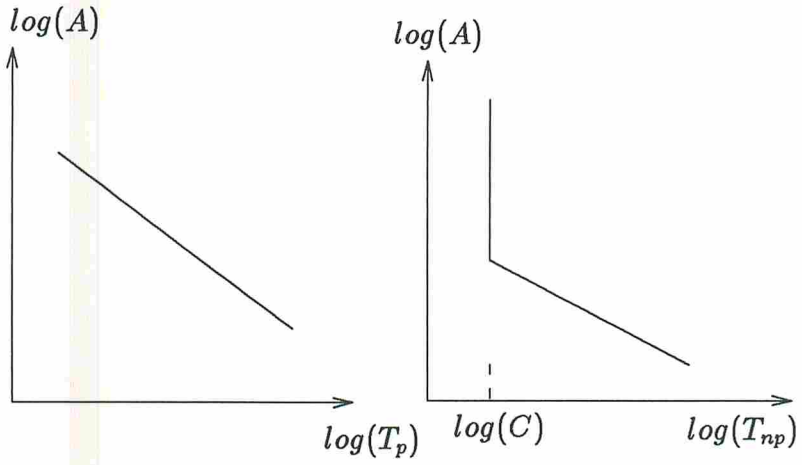


Figure 5.1: Area-Delay Curves For Pipelined and Non-Pipelined Design Styles

number of different types of operations in the data flow graph, A is the functional area of the design, and T_p is the initiation delay of the pipeline. k_p is a constant.

For non-pipelined designs the model predicts the lower-bound clock cycle c_{np} to be

$$c_{np} = maximum(C/N, maximum(d_i)) \tag{5.1.3}$$

and the area-delay to be

$$A \times T_{np} = c_{np} \sum_{i=0}^{m-1} (a_i \times n_i) = c_{np} \times k_{np} \tag{5.1.4}$$

where T_{np} is the circuit delay, C is the critical path delay, and the data flow graph is partitioned into N time steps. k_{np} is a constant.

Equations 5.1.2 and 5.1.4 are shown on a logarithmic scale in Figure 5.1.

5.2 Multi-Type Design

In this chapter we focus on design style selection where the entire data flow graph is chosen to be implemented either as a pipelined or as a non-pipelined design. We do not consider multi-type design where a data flow graph is partitioned into subgraphs some of which are implemented as pipelines and the remaining subgraphs are implemented without pipelining. [5.1] For an example of subgraph partitioning with different implementations, it might be desirable to partition the error recovery part of a data flow graph for a cheap non-pipelined design implementation and the remainder of the data flow graph for a fast and expensive pipelined implementation.

The main problem with the multi-type design scheme is that of matching the data rate of the expensive and fast pipelines with the slower and cheaper non-pipelines. If the non-pipelined circuit cannot keep the pipelined circuit busy by matching the data rates, then there is no gain in performance by designing one part of data flow graph faster than the other. No automatic system which can partition circuits into multi-type designs currently exists.

5.3 Theoretical Foundations

In order to perform design style selection, we compare the pipelined and non-pipelined lower-bound area-delay curves for the input data flow graph and the given module set. Then, of the several designs which meet the constraint we select the design with the best area-delay characteristics. One problem complicates the comparison of pipelined and non-pipelined design models as given by Equations 5.1.2 and 5.1.4. The term *delay* has different meanings for the two design styles. Whereas, T_p for the pipelined designs is the initiation delay [5.2] of the pipe (and a measure of throughput of the pipe), T_{np} is the circuit delay for a

[5.1] This excludes the possibility of structural pipelining [PK89] where a pipelined module is used in a non-pipelined implementation.

[5.2] Iniatiation delay is the delay between two successive inputs to the design. Circuit delay is the delay taken by the design to process one set of input data.

non-pipelined design (time to process one set of input data). Thus, given one of these delays, we need to be able to translate it to the delay corresponding to the other design style. For non-pipelined designs initiation delay is identical to the circuit delay (since the delay between to successive inputs to the non-pipelined design is the same as the circuit delay of the non-pipelined design). On the other hand, the circuit delay of a non-pipelined and pipelined design is lower-bounded by Equation 5.1.3.

In order to perform design style selection for a delay constraint, the designer chooses between the maximum initiation delay (minimum throughput) constraint and the maximum circuit delay constraint. The following two theorems then form the basis for the selection of design style under a delay constraint.

Theorem 5.1 *The lower-bound pipelined design has smaller than or equal initiation delay to the lower-bound non-pipelined design, if module area is held constant.*

Proof: The initiation delay of pipelined designs is given by Equation 5.1.1. Since initiation delay and circuit delay for non-pipelined designs are identical, Equation 5.1.3 gives initiation delay as well as circuit delay for non-pipelined designs. For $C/N \leq maximum(d_i)$, the pipelined and non-pipelined models are identical since $c_{np} \times k_{np} = k_p$. For $C/N > maximum(d_i)$, $A \times T_{np} = C/N \times k_{np}$ is greater than $A \times T_p = k_p$ (since $Ck_{np}/N > k_p$) so that non-pipelined designs will have a larger initiation delay than the pipelined designs for the same area. This proves the theorem. ■

The above theorem states that the optimal design space (with area and initiation delay as its axes) spanned by the non-pipelined design style is contained within the optimal design space spanned by the pipelined design style. A non-pipelined design and a pipelined design may have identical initiation delays for the same area, but the initiation delay for non-pipelined designs will never be smaller than the pipelined designs. This theorem provides a design style selection criteria based on the lower-bound area-delay models. In practice, however,

synthesized designs may not follow the models exactly. Synthesized designs depend upon the quality of the synthesis tools, and the input data flow graph, and further proof that synthesized pipelined designs are better than synthesized non-pipelined designs is difficult. In the several experiments which were conducted with an initiation delay constraint, pipelined designs were always found to be better.

Theorem 5.2 *The lower-bound non-pipelined design has smaller than or equal circuit delay to a synthesized pipelined design, module area remaining the same.*

Proof: Equation 5.1.3 states for any input data flow graph and a given module set, lower-bound circuit delay $T_{np} = Nc_{np} = N \times maximum\ (\frac{C}{N}, maximum(d_i))$. This lower-bound circuit delay is valid for both design styles. Sometimes, in order to optimize resource sharing in the pipelined design style, delays are inserted in the pipe [PD76] [Sto87] [Par85] which increases the circuit delay of the pipe. Hence, a synthesized pipelined design cannot do better than the non-pipelined lower-bound area-delay curve. ■

Theorem 5.2 states that the design space (with area and circuit delay as the two axes) spanned by the lower-bound area-delay tradeoff curve contains the design space spanned by the *synthesized* pipelined designs. In practical cases, non-optimal scheduling of non-pipelined designs (see Section 4.3.1 for examples) may result in synthesized non-pipelined designs having a larger circuit delay than synthesized pipelined designs for a maximum circuit delay constraint; however synthesized pipelined designs will not have a smaller circuit delay than the non-pipelined lower-bound designs. The lower-bound area-delay model is incapable of identifying non-optimal situations and it is not always possible to predict the correct design style for a maximum circuit delay constraint.

So far we have looked at design style selection for delay constraints. Solving design style selection with area constraints is identical. When performing design style selection to meet an area constraint, the designer needs to specify the optimizing function as *minimum circuit delay* or *minimum initiation delay*, and above theorems are used to select the appropriate design style.

5.4 Experiments and Results

Experiments were conducted to verify Theorems 5.1 and 5.2 using the three example data flow graphs (Figures 1.2, 2.2 and 2.3). In this section we present results of the experiments conducted with the module set (a_1, m_1, s_1) from the design library given in Table 1.1. The area vs. initiation delay curves generated by Sehwa [PP88] (pipelined synthesis tool) and MAHA [PPM86] (non-pipelined synthesis tool) for the example data flow graphs are shown in Figures 5.2 through 5.4. The results verify Theorem 5.1. The area vs. circuit delay curves generated by Sehwa and MAHA for the three example data flow graphs are shown in Figures 5.5 through 5.7. The non-pipelined lower-bound area-time curve is also shown in Figure 5.5 through 5.7. The results for a circuit delay constraint verify Theorem 5.2 to be correct.

We know that increasing the resynchronization rate reduces the number of stages in a pipeline which would reduce the circuit delay of the pipelined design[Par85]. We performed experiments with a resynchronization rate of 40 % (i.e. an average of one instruction out of 2.5 instructions cause the pipeline to be flushed) and the results, plotted in Figures 5.8 and 5.9, show that taking resynchronization into account reduces the circuit delay of the pipelined designs. However, as predicted by Theorem 5.2, pipelined designs do not have a smaller circuit delay than the non-pipelined lower-bound area-delay curve.

5.5 Summary

To summarize this chapter, we first developed a common platform for comparing pipelined and non-pipelined designs by identifying two types of delay constraints, the maximum initiation delay constraint and the maximum circuit delay constraint. Next we developed the theory for design style selection and finally verified the theory with some experiments.

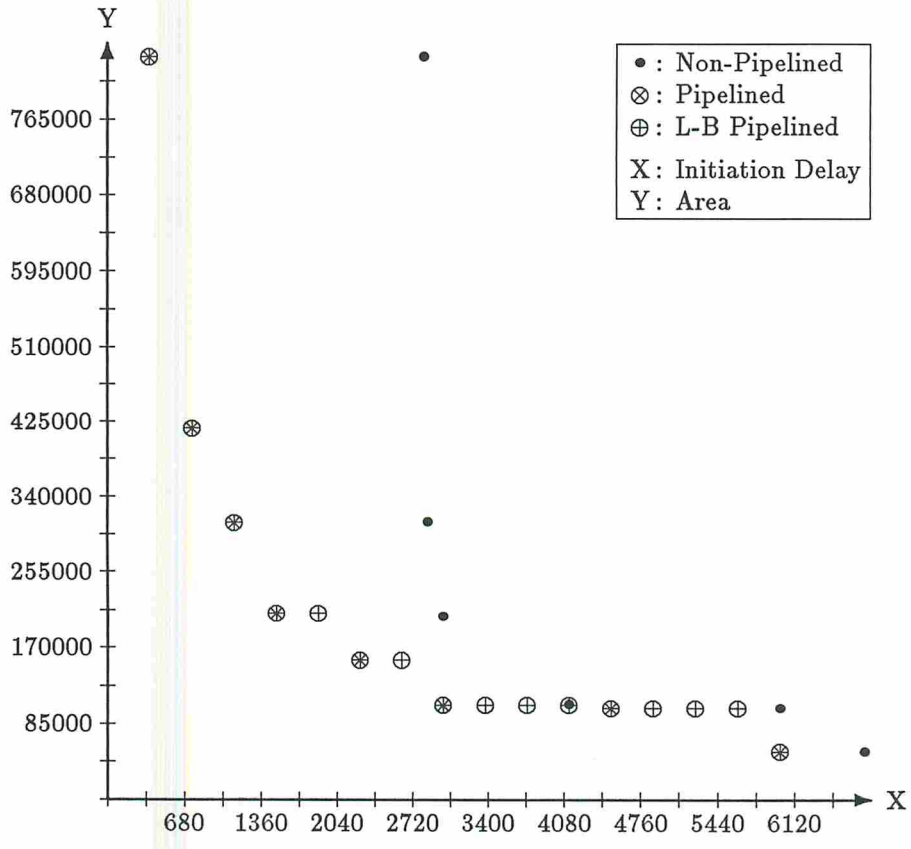


Figure 5.2: Area vs. Initiation Delay For AR-Lattice Filter

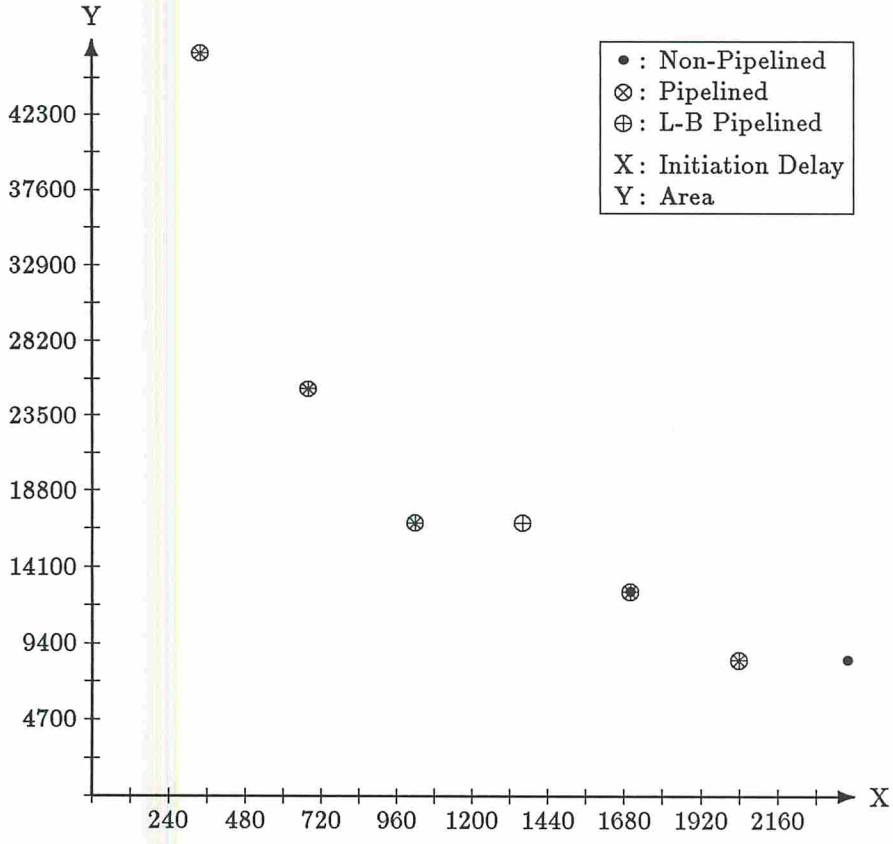


Figure 5.3: Area vs. Initiation Delay For Conditional DFG

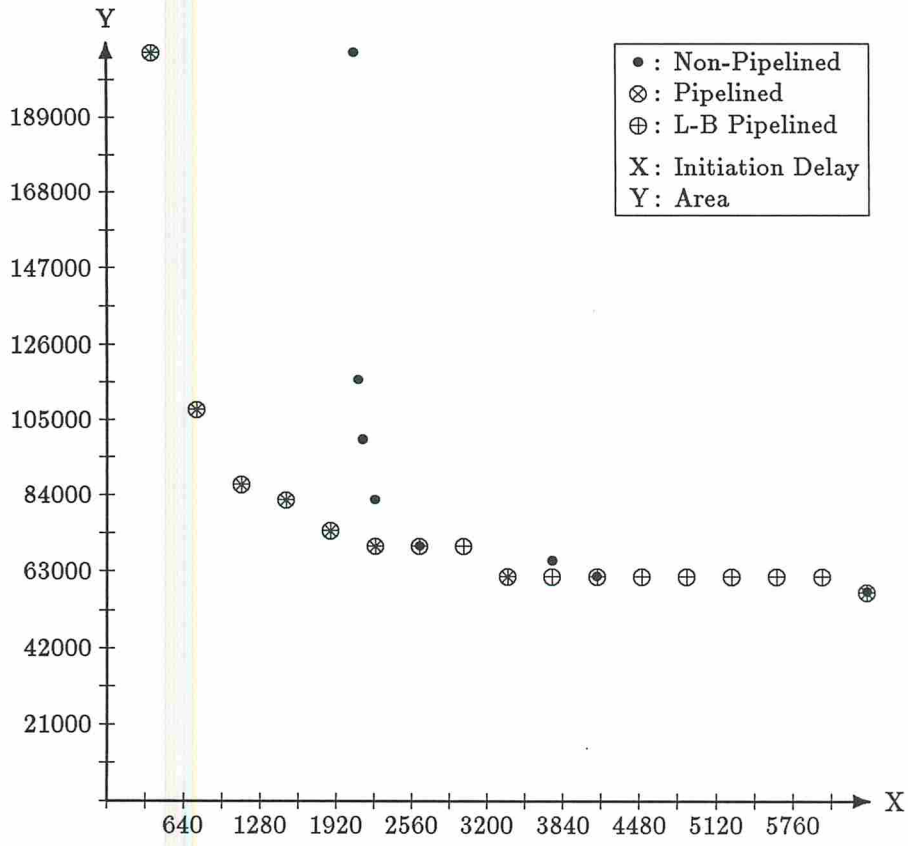


Figure 5.4: Area vs. Initiation Delay For Random DFG

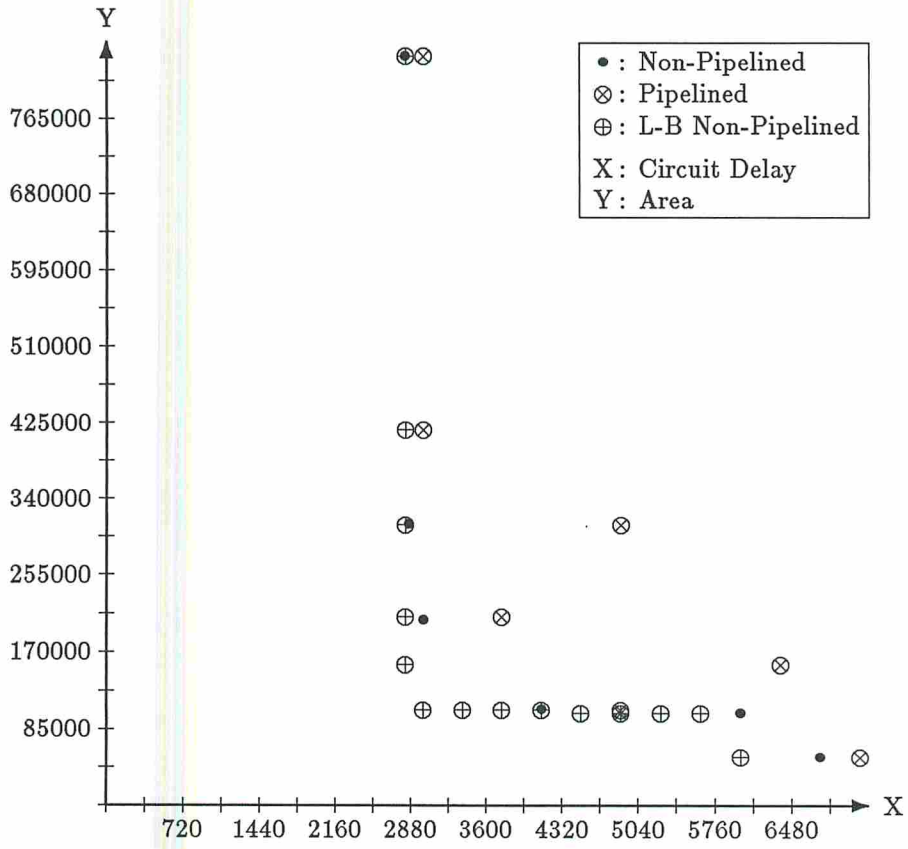


Figure 5.5: Area vs. Circuit Delay Curve For AR-Lattice Filter

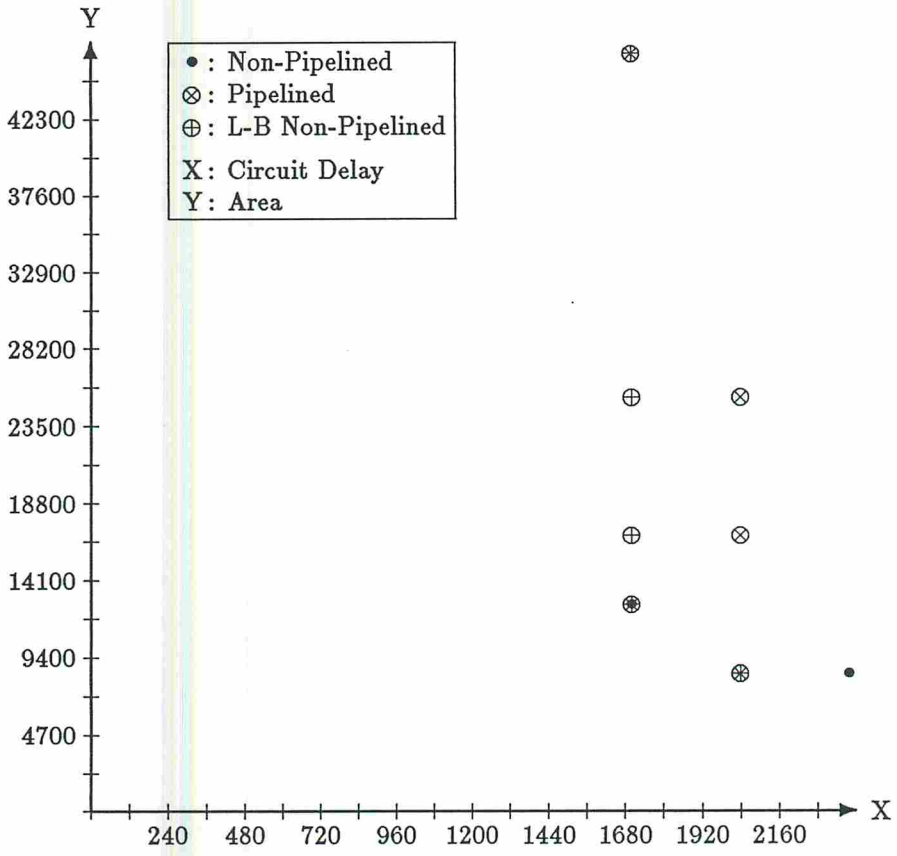


Figure 5.6: Area vs. Circuit Delay Curve For Conditional DFG

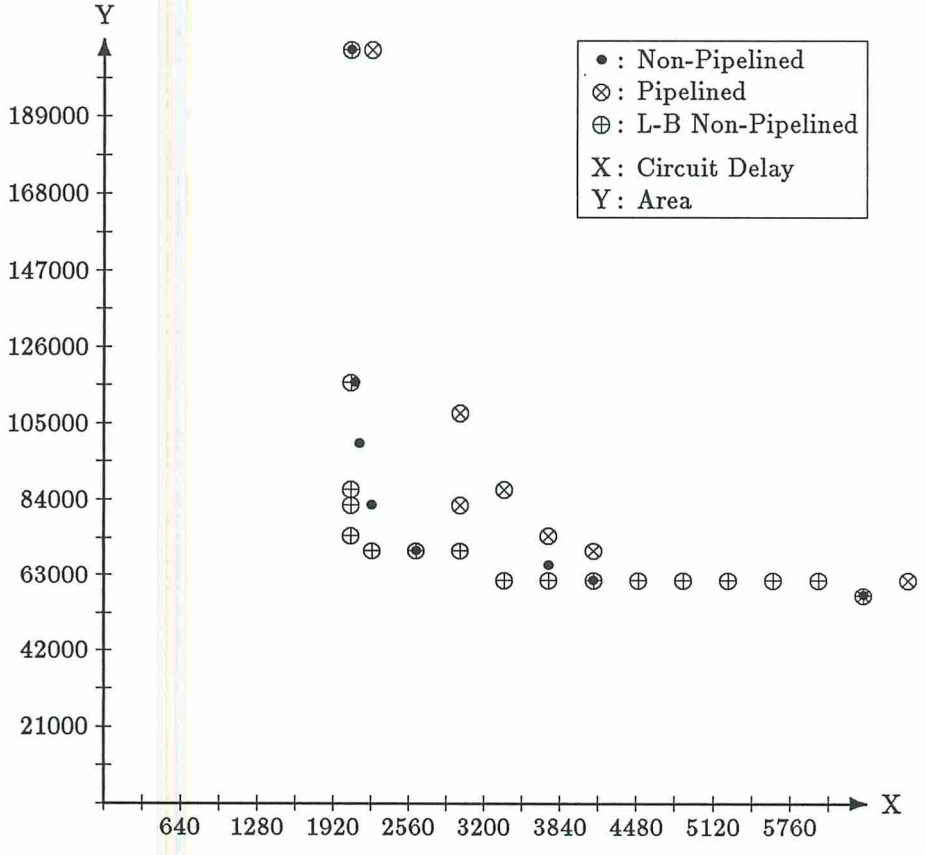


Figure 5.7: Area vs. Circuit Delay Curve For Random DFG

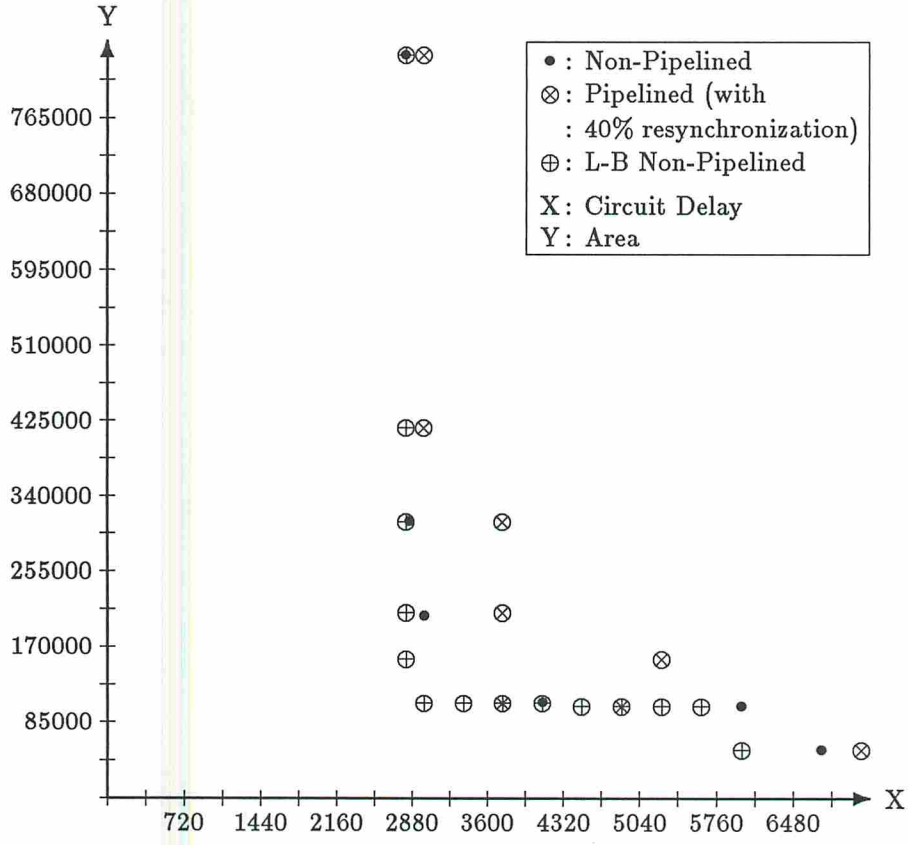


Figure 5.8: Area vs. Circuit Delay Curve For AR-Lattice Filter

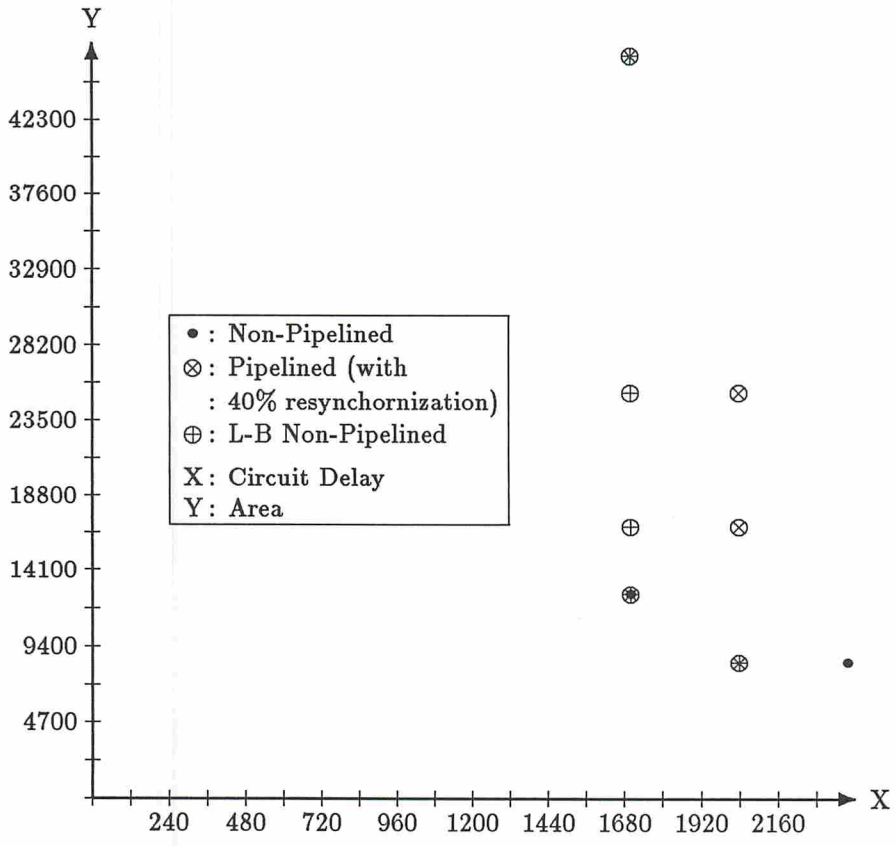


Figure 5.9: Area vs. Circuit Delay Curve For Conditional

Chapter 6

Evaluating Data Flow Graph Transformations

6.1 Introduction

In this chapter we apply the pipelined and non-pipelined area-delay models developed in Chapters 2 and 4 to evaluate the impact of three data flow graph transformations on the AT tradeoff curve for each of the design styles. The chapter is organized as follows. First we summarize the lower-bound area-time models for the two design styles. Next, we identify three data flow graph transformations which will be analyzed and used to demonstrate the potential of the technique. The predictive nature of the models allows us to evaluate the impact of the transformations without actually synthesizing the design. Finally, we present some experiments which were conducted to verify the theory.

6.1.1 The Lower-Bound Area-Delay Prediction Models

The model for pipelined designs predicts the clock cycle to be

$$c_p = maximum(d_i) \tag{6.1.1}$$

and the area-delay product to be

$$A \times T_p = c_p \sum_{i=0}^{m-1} (a_i \times n_i) = constant \tag{6.1.2}$$

Here c_p is the clock cycle of the design, d_i (a_i) is the delay (area) of the module which implements operation i, n_i is the effective number of operations of type i in the data flow graph, m is the number of different types of operations in the data flow graph, A is the functional area of the design, and T_p is the delay between two successive initiations of the input data (also a measure of throughput). The above equations state that the lower-bound clock cycle is the delay of the slowest module in the module set, and the lower-bound area-delay product of the pipelined designs is a constant.

For non-pipelined designs the model predicts the clock cycle to be

$$c_{np} = maximum(C/N, maximum(d_i)) \tag{6.1.3}$$

and the area-delay to be

$$A \times T_{np} = c_{np} \sum_{i=0}^{m-1} (a_i \times n_i) = c \times constant \tag{6.1.4}$$

where T_{np} is the circuit delay, C is the critical path delay, and N is the number of time steps that the data flow graph is partitioned into.

6.2 Transformations Under Consideration

In this chapter we evaluate three data flow graph transformations whose impact can be predicted by the lower-bound area-delay model. The evaluation technique presented in this chapter is general enough to be applied to other transformations and design styles as well. The following three transformations are analyzed in this chapter.

1. Transformations which alter the number of nodes in the data flow graph (for example, common subexpression elimination, strength reduction and dead code elimination [ASU86]),

2. transformations affecting critical path delay (for example, a tree height reduction transformation [Tri85]), and

3. hierarchical decomposition of operations into sub-operations.

Transformations which affect resource sharing and scheduling are not evaluated. There are several instances where it is not possible to schedule a data flow graph for module-optimal or time-optimal non-pipelined designs (see Section 4.3.1 for some examples) due to the inherent data flow graph structure. An example of infeasible scheduling is shown in Figure 4.2c. Transformations which can replace such data flow graph structures by other equivalent structures may help in achieving optimal schedules. It might be necessary to increase the number of nodes in the data flow graph to achieve this goal.

6.3 Evaluating Transformation Impact

In this section we characterize the effect of the above mentioned transformations on the area-delay curve for each design style.

6.3.1 Altering the Number of Operations

First, we analyze the transformations which alter the number of operations in the data flow graph as their impact is easiest to quantify. These transforms always improve the cost and/or performance if the number of nodes is decreased. We use our models to verify this intuitive result. From Equations 6.1.2 and 6.1.4 we observe that the area-delay product of the design is proportional to the number of nodes in the data flow graph. Any increase (decrease) in the number of nodes of a given type in the data flow graph increases (decreases) the area-delay product of the design. That is, for the same performance requirement, more area will be required. This is true for both pipelined and non-pipelined design styles (assuming there is no change in the critical path delay).

Analyzing the impact of a transformation on the area-delay tradeoff curve is the same as performing a sensitivity analysis of Equations 6.1.2 and 6.1.4. For example, for non-pipelined designs varying n_i while keeping other parameters of Equation 6.1.4 constant implies that nodes of the existing operation type are deleted or added to the data flow graph without changing the critical path delay. Let us examine the variation of n_i in greater detail. We perform the analysis on the non-pipelined design style only as the pipelined design style can be similarly analyzed. We have seen that the non-pipelined area-delay tradeoff curve consists of two parts, namely, $C/N > maximum(d_i)$ which is given by a vertical line and $C/N < maximum(d_i)$ which is the sloping line. If we fix all parameters in Equation 6.1.4 and vary n_i alone, we note that the sloping part of the AT curve shifts away from the origin (if n_i is increased) or moves towards the origin (if n_i is decreased). The vertical line does not move. Figure 6.1a shows the lower-bound area-delay tradeoff curves before and after an increase in n_i. In this figure we observe that increasing n_i has moved the sloping part of the area-delay tradeoff curve away from the origin towards the inferior part of the design space. Further all design points of the transformed data flow graph are inferior to the design points of the untransformed data flow graph.

Next consider the effect of changing n_i and m, keeping the other parameters constant. This implies that either nodes of a new operation type are introduced, or nodes of existing operation types are deleted or both. However, the nodes in the transformed data flow graph still have at least one node with a delay equal to $maximum(d_i)$ (as the clock cycle term has not changed), and the critical path is also unchanged. The effect of increasing m and n_i on the lower-bound area-delay tradeoff curve is shown in Figure 6.1b. In this figure we observe that after increasing n_i and m, the sloping part of the area-delay tradeoff curve has moved away from the origin towards the inferior part of the design space. Further, the transformed data flow graph cannot satisfy the area constraint shown by the horizontal dashed line, which could be satisfied by the untransformed data flow graph; i.e. the design space spanned by the transformed data flow graph has reduced.

If we change n_i, m and $maximum(d_i)$ simultaneously keeping the remaining parameters constant, we are considering the situation where nodes slower than existing nodes are introduced into the data flow graph or the slowest nodes are deleted. The effect of an increase in $maximum(d_i)$, n_i, and m on the area-delay tradeoff curve is shown in Figure 6.1c. In this case, the entire area-delay curve, sloping part as well as the vertical part, move towards the inferior design space. Also, the design space spanned by the transformed is smaller than the design space spanned by the untransformed data flow graph.

6.3.2 Reducing Critical Path Length

In order to analyze the critical path reduction transformation we assume that the number of nodes of each operation type remains unchanged before and after the transformation. Evaluating the effect of varying n_i, m, and $maximum(d_i)$ and critical path delay C in the area-delay equations simultaneously can be analyzed similarly. Figure 6.2 shows an example of reducing critical path delay in which the number of nodes in the data flow graph remain unaltered. We show in this paragraph that tree height reduction has no impact on pipelined design style without considering resynchronization. For pipelined designs, the lower-bound area-delay product is dependent only on the area and delay (as clock cycle is dependent on the delay) of the modules used for implementation, and the number of nodes of each type in the data flow graph, and is independent of the critical path delay. This implies that a change in the critical path delay does not impact the area-delay tradeoff curve for pipelined design. Figure 6.3a shows the impact of critical path reduction on pipelined design style, i.e. there is no difference. Changes to the critical path will only affect the area-delay curve of the pipelined design if resynchronization occurs.

A factor which affects the throughput of pipelined designs is resynchronization. Performance degradation due to resynchronization depends on the number of time steps the data flow graph is partitioned into. Reducing the critical path delay is advantageous if resynchronization is considered. From [PP88], we

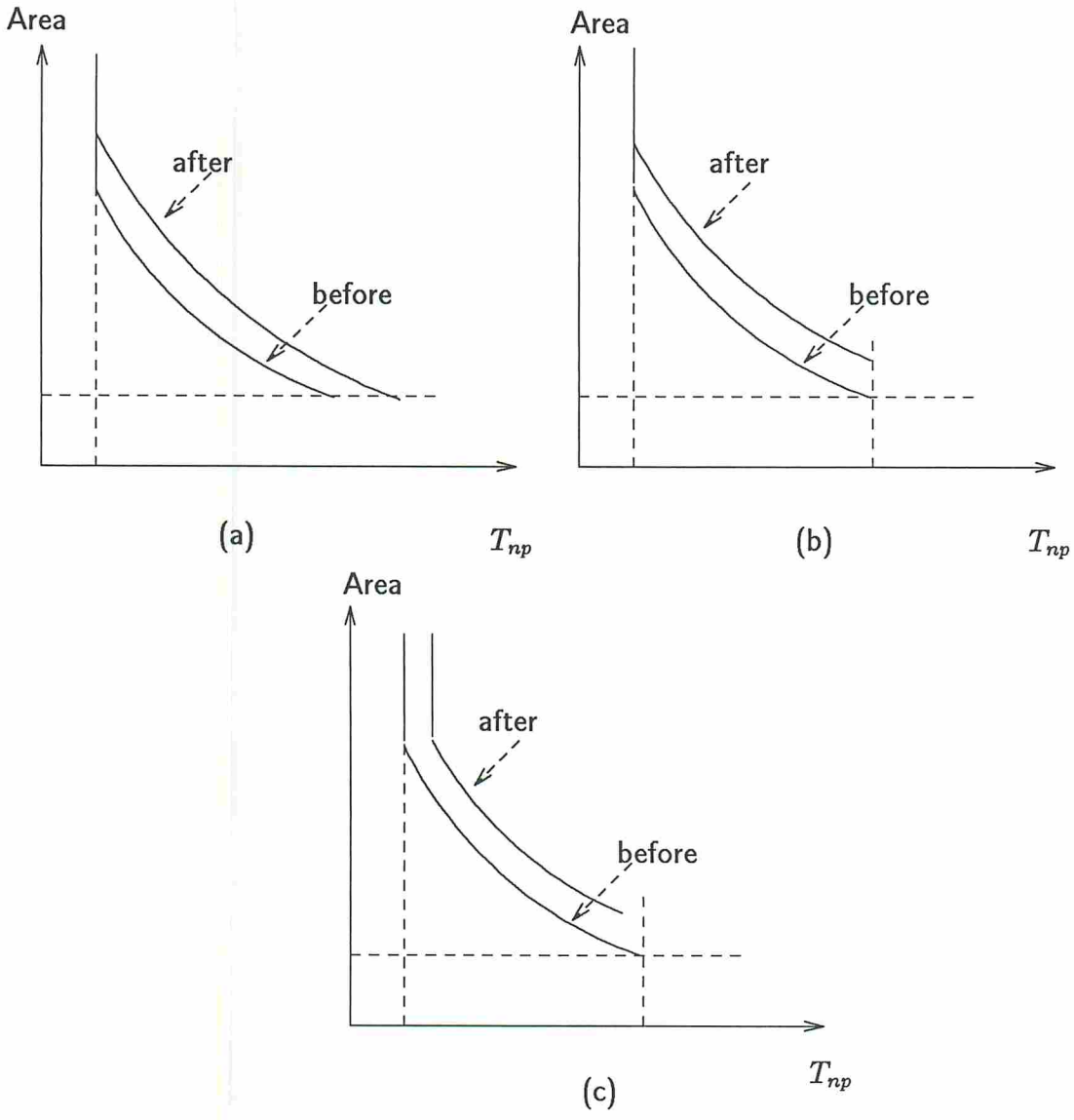
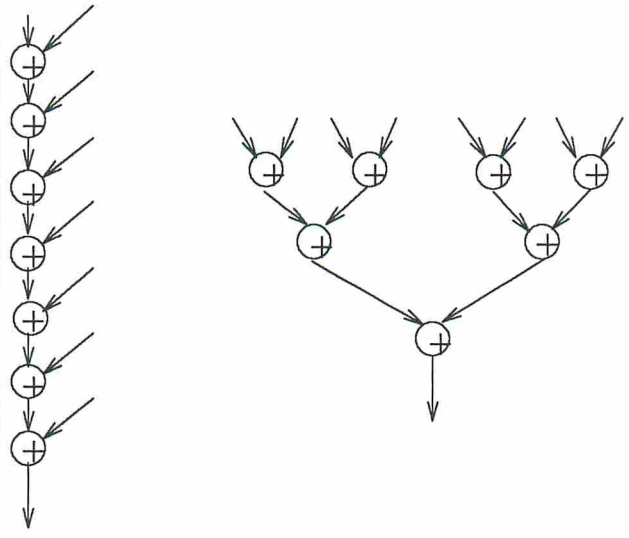
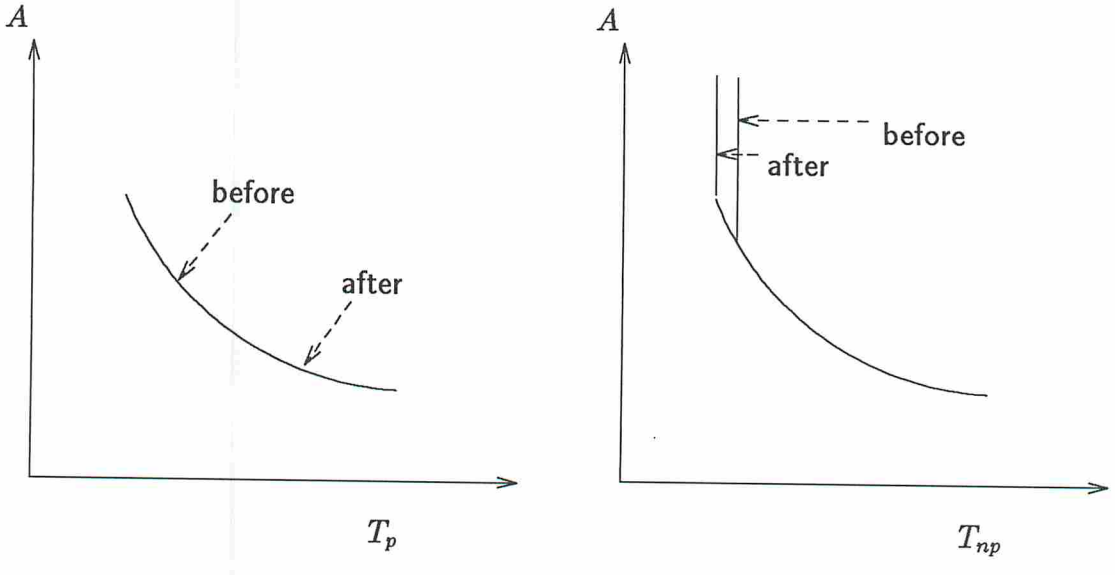


Figure 6.1: Effect of Increase in Node Count for Non-Pipelined Designs



(a) before (b) after

Figure 6.2: A Critical Path Reduction Transformation



(a) Pipelined (b) Non-Pipelined

Figure 6.3: Effect of Critical Path Reduction

know that with resynchronization rate of ρ, $0 \leq \rho \leq 1$, the average performance of a pipelined design is given by

$$T_{p-avg} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \tag{6.3.5}$$

where P is the number of stages of the pipeline, and l is the initiation interval of the pipeline. In this case, the area-delay product of the pipeline is given by

$$A \times T_{p-avg} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \times \sum_{i=0}^{m-1} (a_i \times o_i) \tag{6.3.6}$$

The lower-bound clock cycle for designs with resynchronization cannot be computed using Equation 6.1.1. However, if the clock cycle c_p, resynchronization rate ρ and number of stages of pipeline P are given for all possible values of initiation interval, then the area-delay curve for the original and the transformed data flow graph can be generated and their difference can be easily computed. Figure 6.2 was synthesized by Sehwa for different resynchronization rates. These results are tabulated in Table 6.1 and show that by reducing critical path delay performance of the design can be improved as the resynchronization rate increases.

For non-pipelined designs, the clock cycle depends on the critical path delay. From Equation 6.1.3 we know that $c_{np} = maximum(C/N, maximum(d_i))$. By reducing the critical path delay C, C/N will approach $maximum(d_i)$ for smaller number of partitions p. Reducing the length of the critical path increases parallelism in the data flow graph [Tri85]. However, whereas the scope for parallelism in the data flow graph has increased, the lower-bound $A \times T_{np}$ remains same (assuming there is no change in the number of nodes in the data flow graph). Reducing critical path delay and increasing parallelism in the data flow graph increases the number of possible designs which can be generated and a larger design space can be explored for a faster design. Figure 6.3b shows the impact of critical path reduction on the non-pipelined design style. In this figure we observe that by reducing the critical path delay, a larger design space can be explored for a faster design.

It has been observed in the two examples from [Tri85] (Figures 27 and 35) that reduction in tree height has increased the number of nodes of the data

Resynchronization (%)	Area (A)	Delay (T_p)	
		Before	After
0	29400	340	340
	16800	680	680
	12600	1020	1020
	8400	1360	1360
	4200	2380	2380
10	29400	544	408
	16800	884	748
	12600	1224	1122
	8400	1496	1360
	4200	2380	2380
20	29400	748	476
	16800	1088	748
	12600	1428	1224
	8400	1632	1360
	4200	2380	2380
30	29400	952	544
	16800	1292	884
	12600	1632	1326
	8400	1768	1360
	4200	2380	2380
40	29400	1156	612
	16800	1497	952
	12600	1837	No Result
	8400	1905	1360
	4200	2380	2380

Table 6.1: Effect of Resynchronization on Tree Height Reduction Example

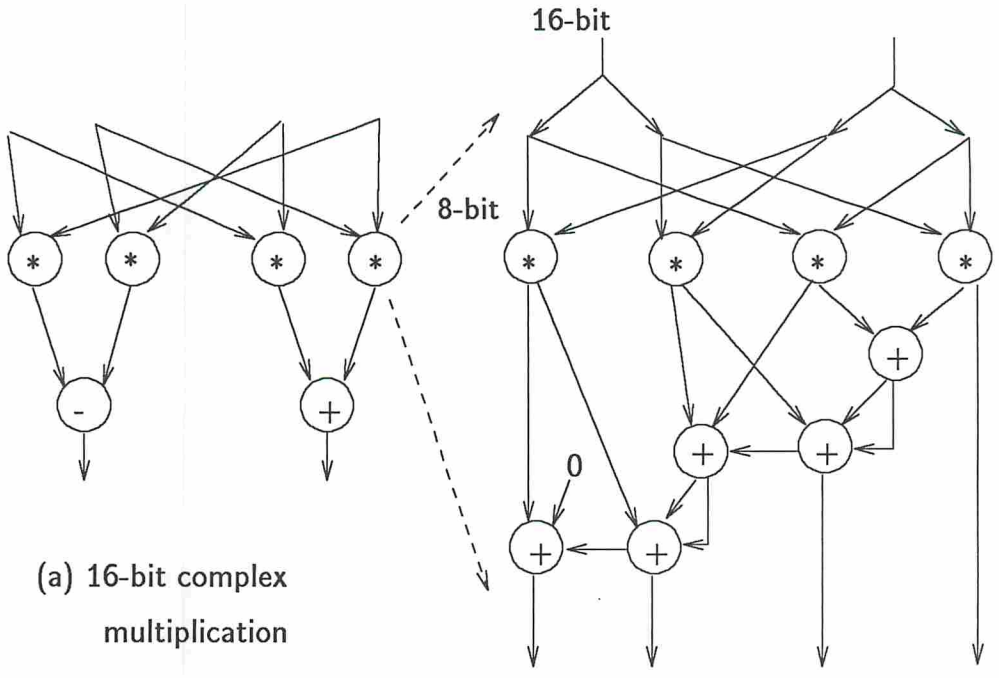


Figure 6.4: Hierarchical Decomposition

flow graph, which moves the area-delay tradeoff curve away from the origin towards the inferior design space. It is not clear if tree height reduction will always increase the number of nodes in the data flow graph.

6.3.3 Hierarchical Decomposition

The effect of transformations which decompose operations into sub-operations is more complicated. Figure 6.4 shows an example hierarchical decomposition where a 16-bit multiplication node can be replaced by a subgraph composed of 8-bit multiplication and addition nodes.

6.3.3.1 Pipelined Designs

Let us examine the pipelined case first. Pipelined designs are characterized by Equations 6.1.1 and 6.1.2. Let the data flow graph before and after the transformation be represented by the following two equations respectively:

$$A_1 \times T_{p1} = c_{p1} \sum_{i=0}^{m1-1} (a_i \times n_i) \tag{6.3.7}$$

$$A_2 \times T_{p2} = c_{p2} \sum_{i=0}^{m2-1} (a_i \times n_i) \tag{6.3.8}$$

where $m1$ is the number of operation types before decomposition and $m2$ is the number of operation types after decomposition. Decomposition of an operation *may* produce a change in the clock cycle, in the number of different types of operations in the data flow graph and in the area of the design. The lower-bound clock cycle is equal to the delay of the slowest operation in the data flow graph. A change in the clock cycle will occur if the slowest operation in the data flow graph is decomposed and the sub-operations produced by the decomposition have a smaller delay than the original operation. Further, as each new sub-operation will have a delay less than or equal to the delay of the original operation, the new clock cycle c_2 will be less than or equal to the old one, c_1. Whenever $c_2 < c_1$, then the new data flow graph will generate designs with higher throughput. The area-delay characteristic of the new data flow graph will depend on the amount of increase in the summation term on the right hand side. If

$$\sum_{i=0}^{m2-1} (a_i \times n_i) < \frac{c_{p1}}{c_{p2}} \sum_{i=0}^{m1-1} (a_i \times n_i) \tag{6.3.9}$$

then the new design will be more area-delay efficient than the original data flow graph. Further, if $\sum_{i=0}^{m1-1} a_i > \sum_{i=0}^{m2-1} a_i$, then cheaper designs might be generated for the new data flow graph. For each hierarchical decomposition employed, the effects can be computed using Equation 6.3.9.

The design characteristics can be improved by decreasing the clock cycle if the areas of the designs before and after the transformation are related by Equation 6.3.9. For pipelined designs we know that the clock cycle is equal to

the delay of the slowest module in the module set (Equation 6.1.1). Decreasing the clock cycle can be achieved either by using a faster module for the slowest operation or decomposing the slowest node of the data flow graph into faster sub-operations. For example, in a data flow graph with 16-bit multiplications and 16-bit additions, the 16-bit multiplication nodes may be decomposed into 8-bit or 4-bit multiplication and addition nodes. As our model does not consider the register and multiplexer area/delay the model suggests an indiscriminate decomposition of the current biggest operation till the data flow graphs contains nothing but basic primitives.

The main advantage of decomposing operations into sub-operations is the increase in potential parallelism and resource sharing which helps the designer in searching a larger design space for a higher performance design or a cheaper design.

6.3.3.2 Non-Pipelined Designs

The only difference in the analysis for pipelined and non-pipelined designs is the critical path delay factor which occurs in the non-pipelined model. If the delay of the operation being decomposed is the same as the critical path delay of the decomposed sub-graph then the non-pipelined design is similar to the pipelined case. Otherwise, the effect of the change in critical path delay can be easily understood by considering the area-delay equations for the two data flow graphs:

$$A_1 \times T_{np1} = maximum(C_1/N, maximum(d_i)) \sum_{i=0}^{m1-1} (a_i \times n_i)$$

$$A_2 \times T_{np2} = maximum(C_2/N, maximum(d_i)) \sum_{i=0}^{m2-1} (a_i \times n_i)$$

An analysis similar to that for pipelined design can be easily done and the break point for the improvement in the area-delay characteristic occurs when

$$\sum_{i=0}^{m2-1} (a_i \times n_i) < \frac{maximum(C_1/N, maximum(d_i))}{maximum(C_2/N, maximum(d_i))} \sum_{i=0}^{m1-1} (a_i \times n_i) \qquad (6.3.10)$$

Similar to the pipelined case, decomposing operations helps in exploring a larger design space for faster and cheaper designs. An example analysis for bit-width hierarchical decomposition (Figure 6.4) for pipelined and non-pipelined designs is given in the next section.

Figure 6.5a (Figure 6.5b) shows the effect of hierarchical decomposition on pipelined design style where the slowest operation has been decomposed into smaller suboperations with an increase (decrease) in area-delay product of the design. In Figure 6.5, $A_1' = \sum_{i=0}^{m1-1} a_i$, and $A_2' = \sum_{i=0}^{m2-1} a_i$.

6.4 Experiments and Results

Experiments were conducted to verify the theory given in the preceding section. For every transformation type a data flow graph was selected and synthesized before applying the transformation and again after applying the transformation. For every such pair of data flow graphs pipelined and non-pipelined designs were synthesized. Sehwa [PP88] and MAHA [PPM86] were the pipelined and non-pipelined synthesis tools used in the experiments. The modules $a_1, m_1 s_1$ shown in Table 1.1 were used in the experiments.

To demonstrate the effect of increase in number of nodes in the data flow graph we used the *presum* computation and the data flow graphs given in Figures 32 and 35 of [Tri85]. The number of addition nodes in the data flow graph prior to applying the transformation was seven and after the transformation it was twelve. The critical path had seven addition nodes before the transformation and five addition nodes after the transformation. The results for pipelined and non-pipelined synthesis are given in Figures 6.6 and 6.7 and are similar in shape to the curves shown in Figure 6.1.

To evaluate the effect of reducing the critical path delay we used the simple data flow graphs given in Figure 6.2. The results produced by Sehwa and MAHA are shown in Figures 6.8 and 6.9 respectively. The results are in consonance with the prediction.

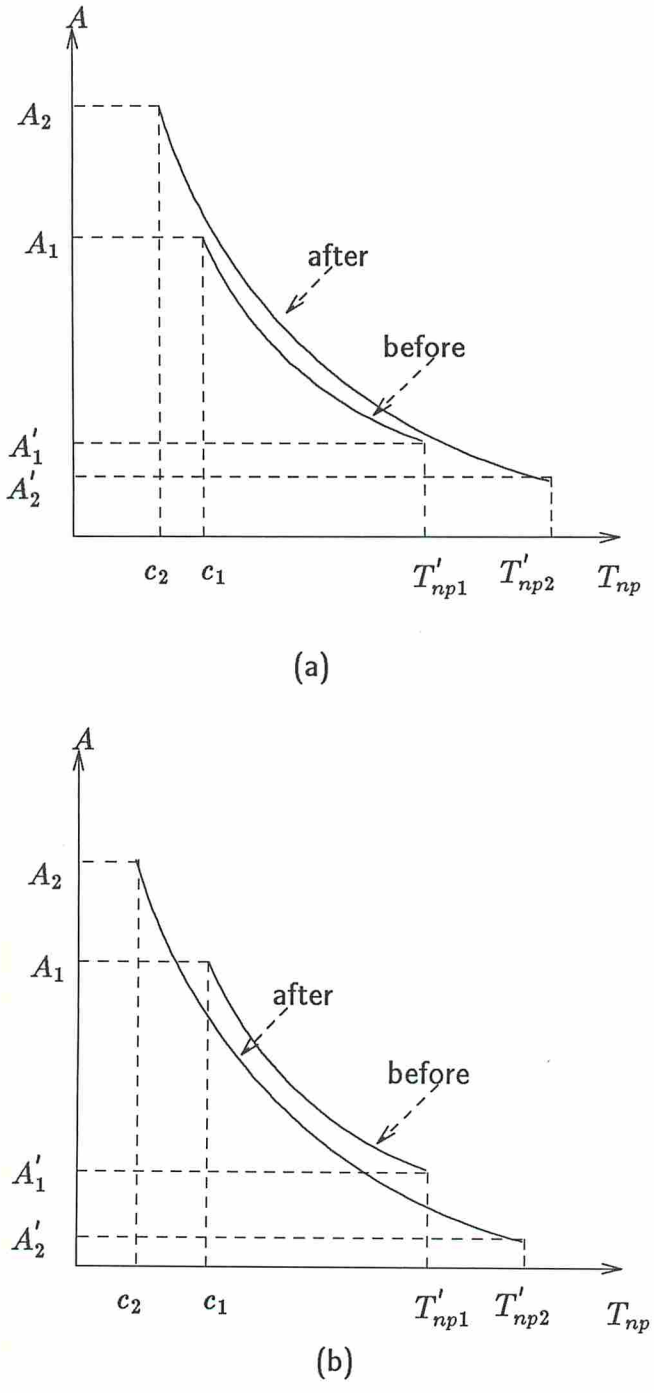


Figure 6.5: Effect of Hierarchical Decomposition on Pipelined Designs

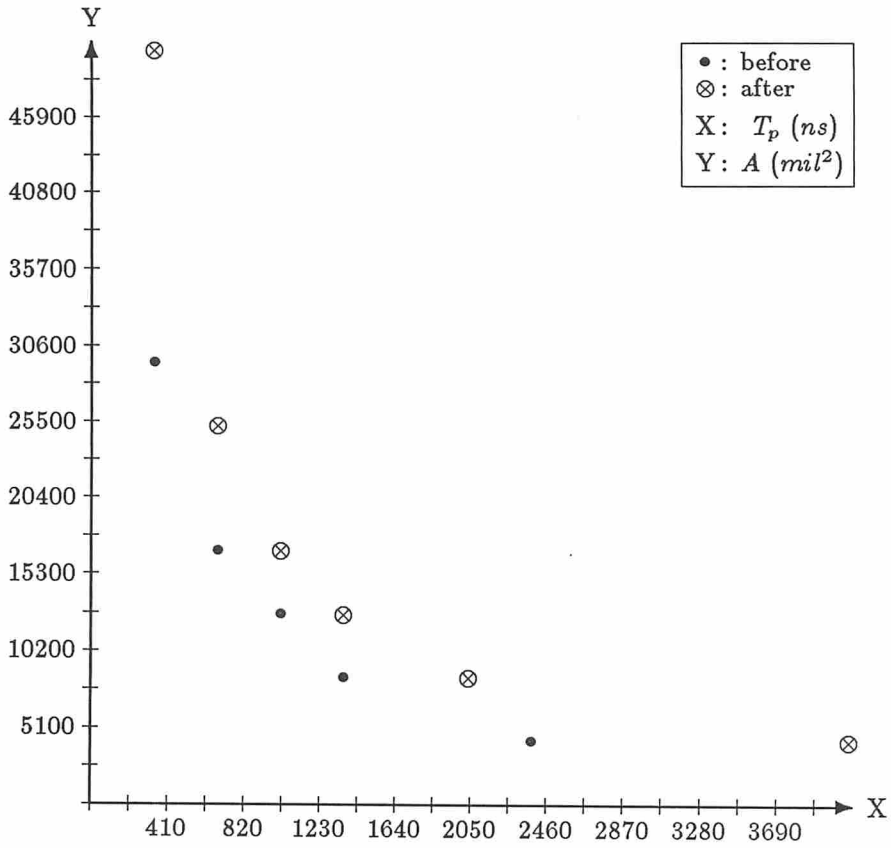


Figure 6.6: Pipelined Designs: Increase in Node Count

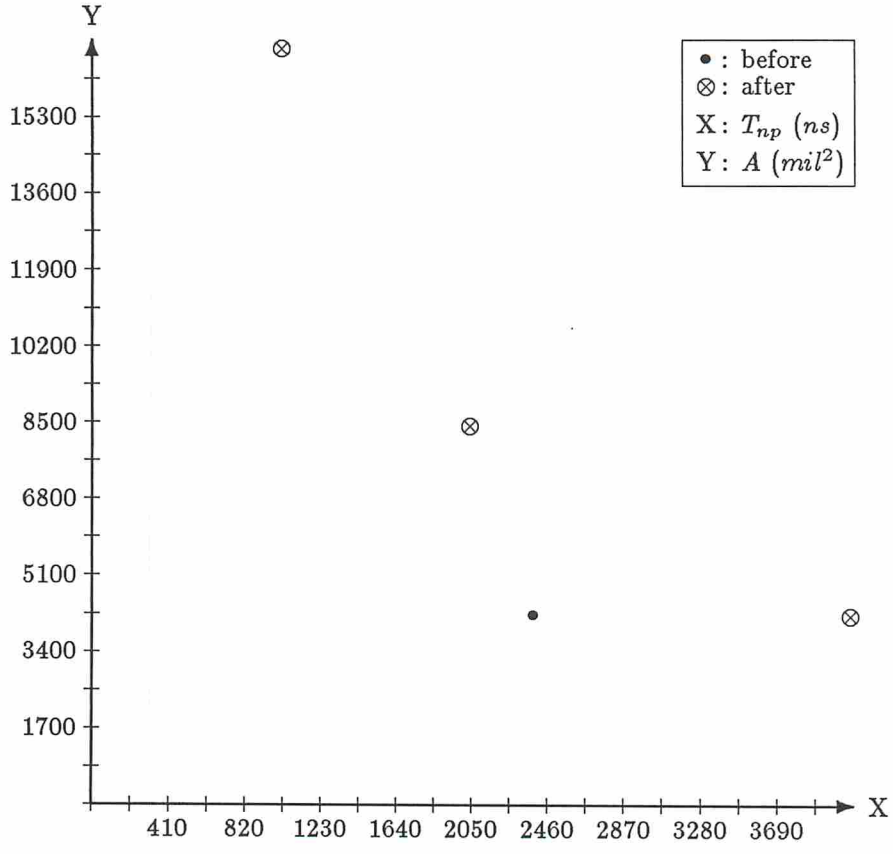


Figure 6.7: Non-Pipelined Designs: Increase in Node Count

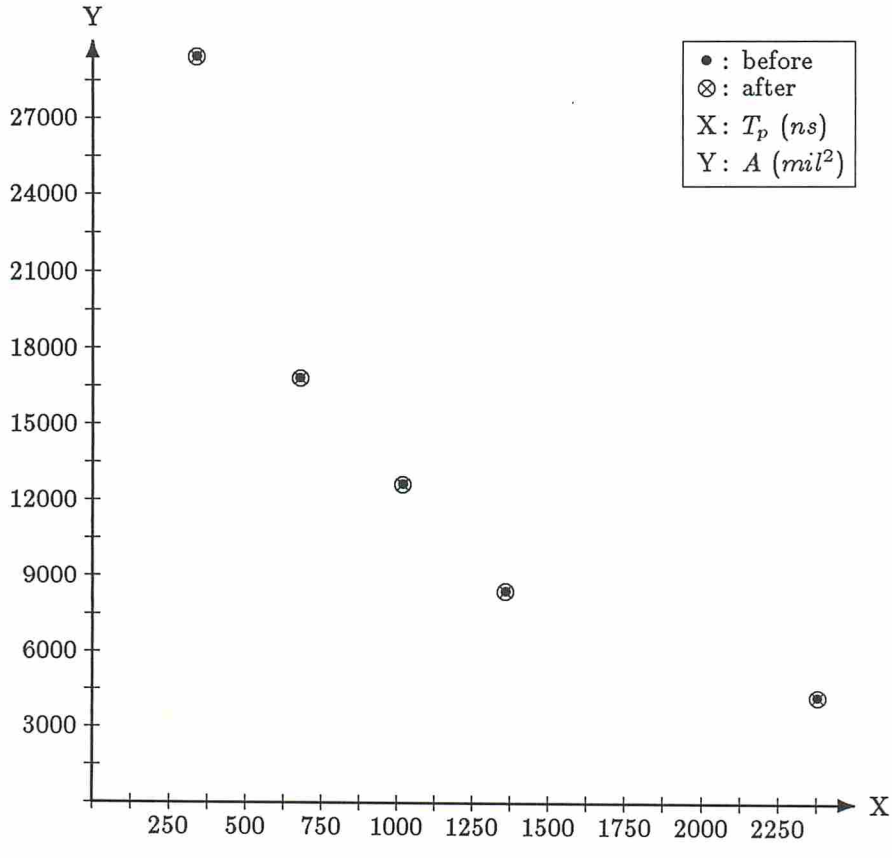


Figure 6.8: Pipelined Designs: Tree Height Reduction Transformation

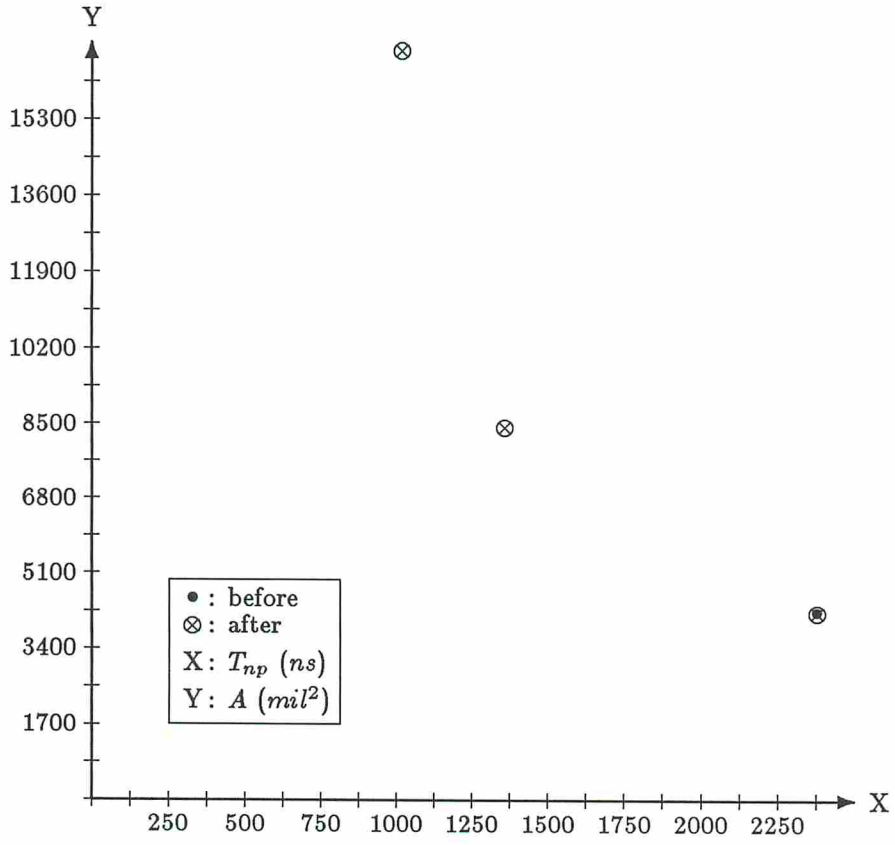


Figure 6.9: Non-Pipelined Designs: Tree Height Reduction

The 16-bit complex multiplication data flow graph shown in Figure 6.4a is used to demonstrate the effect of hierarchical decomposition. Figure 6.4b shows the decomposition of a 16-bit multiplication node into 8-bit multiplications. The original data flow graph has four 16-bit multiplication, one 16-bit addition and one 16-bit subtraction node. After substituting Figure 6.4b for all 16-bit multiplication nodes, the data flow graph has 16 8-bit multiplications, 20 8-bit additions, one 16-bit addition and one 16-bit subtraction node. The critical path delay of the original data flow graph consists of one 16-bit multiplication and one 16-bit addition delay (assuming a 16-bit addition delay is equal to a 16-bit subtraction delay). The transformed data flow graph has a critical path delay of one 8-bit multiplication, five 8-bit additions and one 16-bit addition delay.

Using Equations 6.3.7 and 6.3.8 for pipelined designs the area-delay characteristic of the two data flow graphs can be easily computed. For the original data flow graph given in Figure 6.4a,

$$A_1 \times T_{p1} = c_{p1}(4 \times a_{16-bitmultiplier} + a_{16-bitadder} + a_{16-bitsubtractor})$$

Using the area and delay of modules (a_1, m_1) given in Table 1.1, $c_{p1} = maximum\ (340, 375) = 375$ and

$$A_1 \times T_{p1} = 375(4 \times 49000 + 4200 + 4200) = 2.15 \times 10^7 mil^2 ns$$

Similarly, for the data flow graph with the decomposed operations, we have

$$A_2 \times T_{p2} = c_{p2}(16 \times a_{8-bitmultiplier} + 20 \times a_{8-bitadder} + a_{16-bitadder} + a_{16-bitsubtractor})$$

$$= 340(16 \times 13800 + 20 \times 2000 + 4200 + 4200) = 9.15 \times 10^7 mil^2 ns$$

From the above analysis of the two data flow graphs we see that the original design has a better area-delay characteristic than the second. However, the second design has a higher throughput as compared to the first and it allows to user to explore a larger design space, especially in the spectrum of designs with cheap area, for a design satisfying the area constraint. The results of the pipeline synthesis program Sehwa are given in Figure 6.10 and support the analysis.

Figure 6.11 shows the results produced for the hierarchical decomposition transformation by MAHA. The original graph had a critical path delay of $C_{np1} = 715nS$, and $c_{np1} = maximum\ (715/N, maximum(375, 340))$. The critical path delay of the new data flow graph is $C_2 = 1815nS$, and $c_{np2} = maximum\ (1815/N, maximum(225, 250, 340))$. Thus,

$$A_1 \times T_{np1} = c_{np1}(4 \times 49000 + 4200 + 4200) = 204400 \times c_{np1}$$
$$A_2 \times T_{np2} = c_{np2}(16 \times 13800 + 20 \times 2000 + 4200 + 4200) = 269200 \times c_{np2}$$

For any $p < 5$, $c_{np1} < c_{np2}$ and $A_1 \times T_{np1} < A_2 \times T_{np2}$. For $p > 4$, $c_{np1} = 375nS$ and $c_{np2} = 340nS$. Substituting these values we again get the result $A_1 \times T_{np1} < A_2 \times T_{np2}$. Thus, the original design has a better area-delay product than the design for the new data flow graph. Its possible that the decomposition would allow more module-optimal designs to be synthesized.

6.5 Summary

In this chapter we have demonstrated a technique for evaluating the impact of data flow graph transformations on pipelined and non-pipelined design styles. We have analyzed three transformations and verified the theoretical predictions with experimental results. The method is general enough to be applied to several other data flow graph transformations as well. The models used in this chapter are restricted in that only functional area of the designs is considered. In practice, register, multiplexer, and wiring area and delay estimates need to be incorporated into the model.

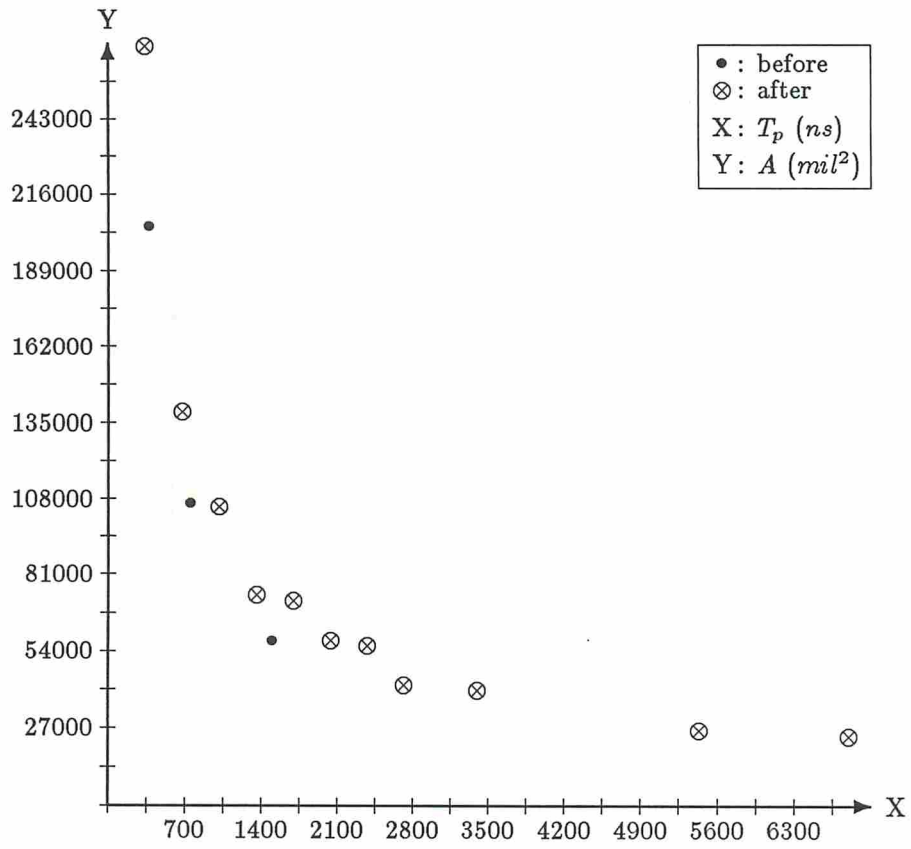


Figure 6.10: Pipelined Designs: Hierarchical Decomposition

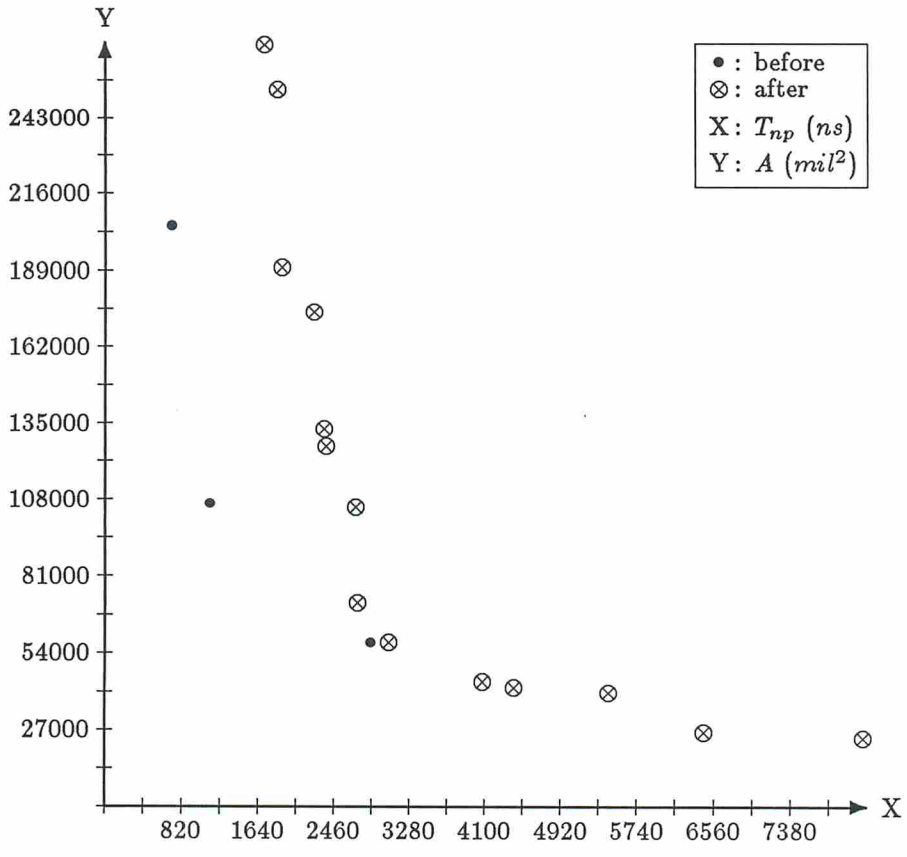


Figure 6.11: Non-Pipelined Designs: Hierarchical Decomposition

Chapter 7

Conclusion and Future Research

"This is not the end;
This is not even the beginning of the end;
But perhaps, this is the end of the beginning."
—Winston Churchill, *1940*

7.1 Introduction

In this thesis we have looked at some high-level problems in the synthesis of digital systems. We have developed mathematical models for area-delay trade-off characteristics for pipelined and non-pipelined design styles and used these models for solving module selection problem for pipelined designs, for design style selection and for evaluating the impact of data flow graph transformations on area-delay characteristics for pipelined and non-pipelined design styles. In this chapter, we present critical evaluations of the work which has been accomplished in this thesis. We also propose areas for future research.

7.2 Module Selection for Non-Pipelined Design Style

The applicability of the module selection techniques described in Chapter 3 to the non-pipelined design style needs to be examined. Since the lower-bound area-delay model for the two design styles is the same whenever the clock cycle is equal to the delay of the slowest operator, it suggests that the module selection results for the pipelined design style should be valid for non-pipelined design style as well. The relationship is complicated, however, by the fact that module sets with identical slowest operator delay can have different critical path delays. Consider a data flow graph with a critical path having an addition and a multiplication in series and module sets (a_1, m_2) and (a_2, m_2) and the implementation style is non-pipelined. Both module sets have same slowest operator m_2. However, the critical path delay with the first module set is $340 + 2950 = 3290ns$ and the critical path delay with the second module set is $530 + 2950 = 3480ns$. Also the area of the design using module set 1 is larger than the design using module set 2 for identical time-steps. In Figure 7.1 the design space with the two module sets are plotted. We observe that the dashed vertical line separates the design space into two: to the left of the dashed line, using module set 1 produces non-inferior designs and to the right of the dashed line using module set 2 produces non-inferior designs and using module set 1 produces inferior designs. If the module selection theory for pipelined designs is used for non-pipelined design style, module set 1 will not be generated. We have however, shown by an example that using module set 1 does produce non-inferior designs and cannot be eliminated.

We conducted some experiments to verify this observation. Figures 7.2 and 7.3 show the synthesized design points for the AR filter and the conditional data flow graphs produced by MAHA [PPM86]. These design points were produced for all combinations of the module sets given in Table 1.1. The pipelined module selection theory produces good but non-optimal results for non-pipelined

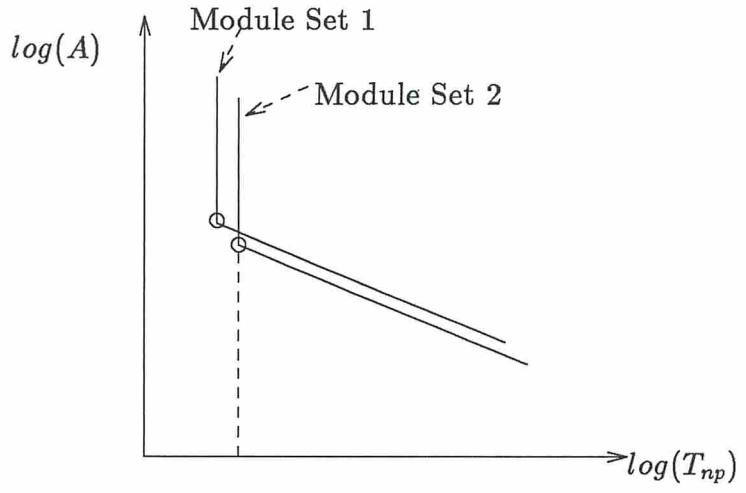
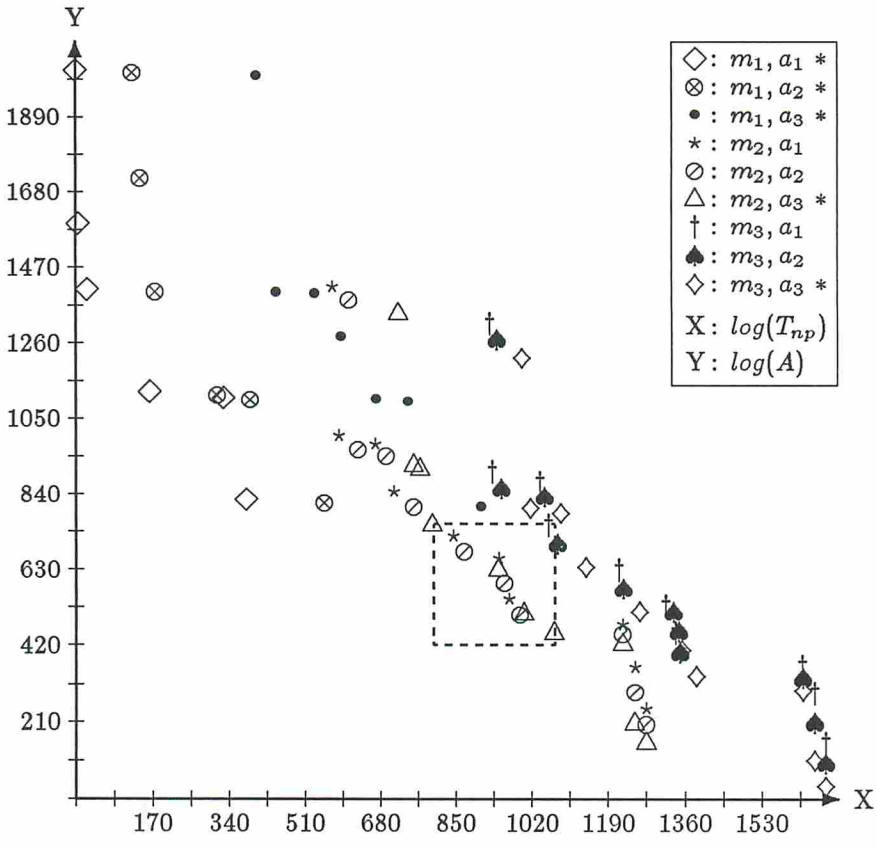


Figure 7.1: Effect of Different Module Sets on Non-Pipelined Designs

design style for both data flow graphs. The non-optimal results are enclosed in dashed boxes in Figures 7.2 and 7.3.

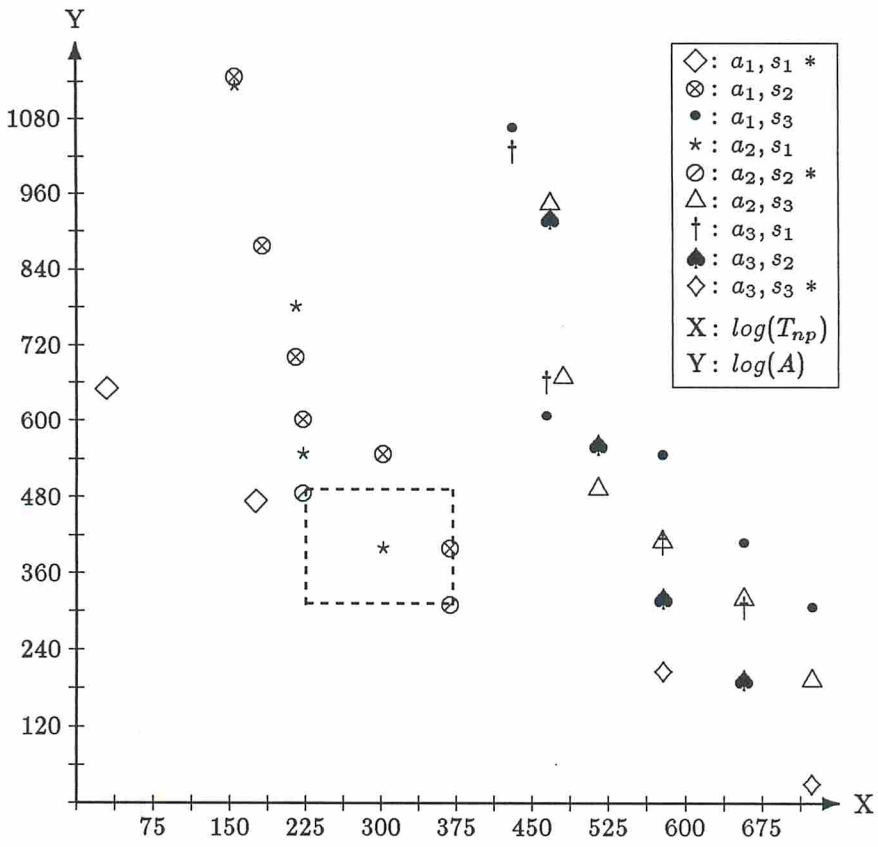
7.3 Limitations and Future Research

There are two major limitations to the area-time models developed for the pipelined and non-pipelined design styles. Firstly, only functional area and delay of the design have been considered in the models. The costs and delays of registers, multiplexers, and wiring need to be incorporated into the model. Some preliminary work in register estimation has been done by Kurdahi [KP87]. Mlinar [MP88a] has derived upper and lower bounds of multiplexers and registers required for pipelined and non-pipelined design styles. He has also estimated the average number of registers and multiplexers required in the design. The two estimates, the operator area-delay and the register and multiplexer area estimates have been combined. The register and multiplexer estimator requires the number of stages as an input. The number of stages for non-pipelined designs can be easily estimated, however for pipelined designs the number stages is not easy to estimate. Apart from this drawback, the combined results are



* denotes the module set generated by SLIMOS.

Figure 7.2: AR Filter Synthesized by MAHA



* denotes the module set generated by SLIMOS.

Figure 7.3: Conditional DFG Synthesized by MAHA

promising. Figure 7.4 compares graph the predicted results with the actual non-pipelined synthesized results for the AR filter. The predicted results include the AT prediction and the register and multiplexer estimation done by Mlinar. The synthesized results are produced by MAHA and MABAL [KP]. The next step in the estimation process is computing the wiring area and delay. PLEST [KP86] is a standard cell wiring estimation program which accepts average wire length and the number of 2-wire nets to estimate the wiring area of the design. Hence, we see that a complete estimator for the ADAM system is well on its way.

A second limitation of the work has been the assumption that the delay of every operation is less than the clock cycle length. One way of circumventing this problem was mentioned earlier (Section 2.1.2).

The resynchronization rate has not been taken into account in the pipelined area-time model. Equations which consider the resynchronization rate have been derived in [PP88]. With resynchronization, however, the evaluation of lower-bound clock cycle becomes difficult and $AT_p = constant$ is not valid. Resynchronization limits the results of module selection for high resynchronization rates and data flow graph transformation evaluation.

The model is restricted in that throughput of the pipeline is assumed to be the most important timing quantity. Other pipeline synthesis systems, such as the pipelined synthesis system under development at GE Research Laboratory [HCDd88], consider the minimization of the number of stages to be important. Yet another criteria for design might be design time. A general model which could factor in all the desired options would be beneficial.

The solution to the design style selection problem assumed a fixed module set as an input. In case the entire design library is given as the input, the appropriate module set satisfying the design constraint has to be selected. For the same design constraint it is possible for the best module sets for the two design styles to be different. For pipelined designs SLIMOS can generate the best module set. Similarly the best module set must be generated for non-pipelined designs prior to the actual comparison of the better design style

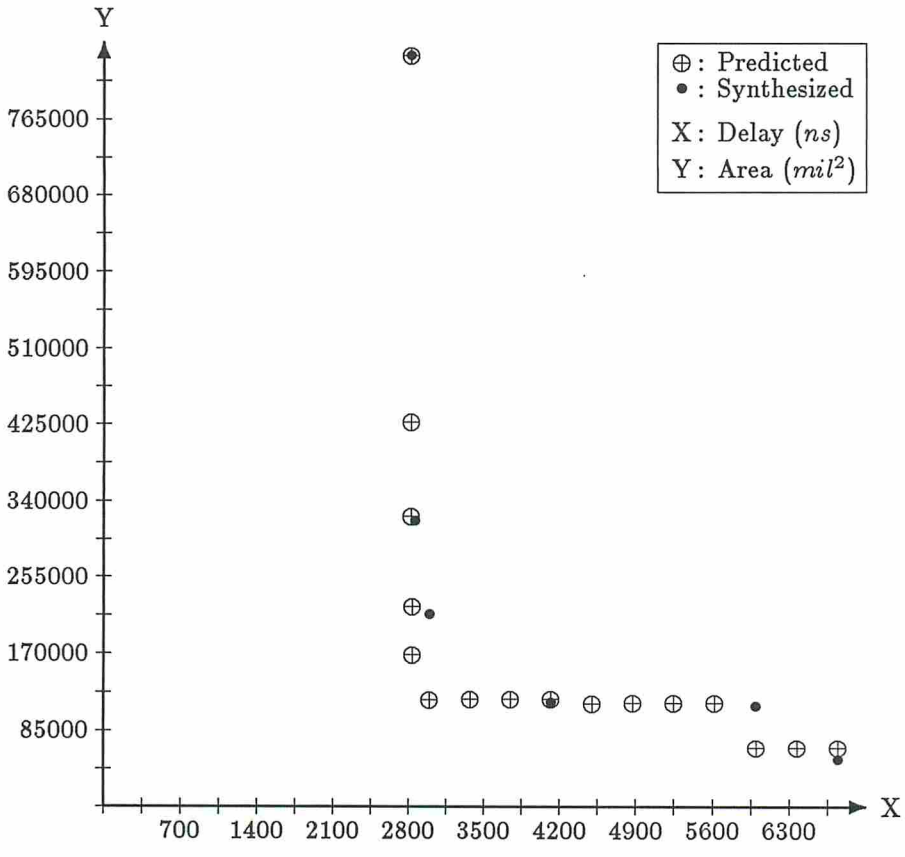


Figure 7.4: AT Curves for AR Filter

The limitations on the area-delay models affect module selection results, design style selection results and evaluation of transformations. Aside from these limitations, the solution technique for these problems remains much the same as described in this thesis.

Appendix A

Notation

In this appendix we summarize the nomenclature used in the thesis.

- $A = \sum_{i=0}^{m-1}(a_i \times o_i)$ is the total functional area of the design.
- a_i is the area of the module which has been selected for implementation of operation i. $a_i = ar_{ij}$ if module j is selected to implement operation i.
- ar_{ij} is the area of module type j which can implement operation type i.
- c_p is the length of the clock cycle used for pipelined design.
- c_{np} is the length of the clock cycle used for non-pipelined design.
- d_i is the delay of the module which has been selected for implementation of operation i. $d_i = dl_{ij}$ if module j is selected to implement operation i.
- dl_{ij} is the delay of module type j which can implement operation type i.
- l is the initiation interval of the design. Initiation interval is the number of clock cycles between initiations of two successive data inputs [Kog81] and is used in pipelined designs only.
- $M = \sum_{i=0}^{m-1} n_i$ is the total number of mutually exclusive nodes in the data flow graph. See the definition of n_i below for details.
- m is the number of different types of operations in the data flow graph.
- N is the number of time steps the data flow graph is partitioned into.
- n_i is the effective number of nodes (operations) of type i in the data flow graph. For a data flow graph with conditional branches, we sum the maximum number of type i nodes to be executed during any single execution instance of the data flow graph, which is given by the sum of:

1. the number of unconditional nodes of type i, and
2. for every pair of outermost *if* - *join* nodes, the maximum number of nodes of type i in that conditional branch. For more details refer to Lemma 4.5.3 of [Par85].

- o_i is the number of modules required in a completed design for each operation type i.
- p_i is the number of module types which can implement an operation type i.
- q_i is the total number of nodes of type i. Note, $q_i \geq n_i$.
- $Q = \sum_{i=0}^{m-1} q_i$ is the total number of nodes in the data flow graph.
- $S \subseteq X$ is the set of module sets selected from X by Algorithm 2 which satisfy the design constraint.
- $T_p = l \times c_p$ is the delay between two successive initiations of data for pipelined designs.
- $T_{np} = N \times c_{np}$ is the front to end delay of the non-pipelined design.
- X is a set of module sets generated by Algorithm 1 (Chapter 3).

Bibliography

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

[ASU86] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[Bar81] M. Barbacci. Instruction Set Processor Specification (ISPS): The Notation and its Applications. *IEEE Transactions on Computers*, 30(1):24–40, January 1981.

[BB87] N. N. Biswas and C. Bhat. A Maximum PLA Folding Algorithm. In *Proceedings of the International Conference on Computer-Aided-Design*, IEEE/ACM, October 1987.

[BCM*88] R. K. Brayton, R. Camposano, G. De Micheli, R. H. J. M. Otten, and J. van Eijndhoven. The Yorktown Silicon Compiler System. In D. D. Gajski, editor, *Silicon Compilation*, pages 204–310, Addison-Wesley Publishing Company, Reading, Massachusetts, 1988.

[BG87] F. D. Brewer and D. D. Gajski. Knowledge Based Control in Micro-Architecture Design. In *Proceedings of the 24th Design Automation Conference*, ACM/IEEE, June 1987.

[BTF83] H. Brown, C. Tong, and G. Foyster. Palladio: An Exploratory Environment for Circuit Design. *IEEE Computer*, 16(10):41–56, December 1983.

[DDT83] M. Davio, J. -P. Deschamps, and A. Thayse. *Digital Systems with Algorithm Implementation.* John Wiley & Sons, New York, 1983.

[DLS81] S. Davidson, D. Landskov, and B. D. Shriver. Some Experiments in Local Microcode Compaction for Horizontal Machines. *IEEE Transactions on Computers*, 30(7):460–477, July 1981.

[DN87] S. Devdas and A. R. Newton. Algorithms for Hardware Allocation in Data Path Synthesis. In *Proceedings of the International Conference on Computer Design*, ACM/IEEE, October 1987.

[DRSC86] H. De Man, J. Rabaey, P. Six, and L. Classen. CATHEDRAL II: A Silicon Compiler for Digital Signal Processing. *IEEE Design and Test*, 3(6):13–25, December 1986.

[Eve79] S. Even. *Graph Algorithms.* Computer Science Press, Rockville, Maryland, 1979.

[FK86] Y-P. S. Foo and H. Kobayashi. A Knowledge-Based System for VLSI Module Selection. In *Proceedings of the International Conference on Computer-Aided-Design*, ACM/IEEE, October 1986.

[GBK85] E. Girczyc, R. J. A. Buhr, and J. P. Knight. Applicability of a Subset of Ada as an Algorithmic Hardware Description Language for Graph-Based Hardware Compilation. *IEEE Transactions on Computer-Aided-Design*, 4(2):134–142, April 1985.

[GE88] C. H. Gebotys and M. I. Elmasry. VLSI Design Synthesis with Testability. In *Proceedings of the 25th Design Automation Conference*, ACM/IEEE, June 1988.

[Gir84] E. Girczyc. *Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions.* PhD thesis, Department of Electronics, Carleton University, July 1984.

[Gir87] E. Girczyc. Loop Winding - A Data Flow Approach to Functional Programming. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, IEEE Computer Society, May 1987.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[Ham83] G. Hamachi. *Designing Finite State Machines with PEG.* UC Berkeley, 1983.

[HCDd88] K. S. Hwang, A. E. Casavant, M. Dragomirecky, and M. A. d'Abreu. Constrained Conditional Resource Sharing in Pipeline Synthesis. In *Proceedings of the International Conference on Computer-Aided-Design*, ACM/IEEE, November 1988.

[HE88] B. S. Haroun and M. I. Elmasry. Automatic Synthesis of a Multi-Bus Architecture for DSP. In *Proceedings of the International Conference on Computer-Aided-Design*, ACM/IEEE, November 1988.

[HJ88] R. I. Hartley and J. R. Jasica. Behavioral to Structural Translation in a Bit-Serial Silicon Compiler. *IEEE Transactions on Computer-Aided-Design*, 7(8):877–886, August 1988.

[HP83] L. Hafer and A. Parker. A Formal Method for the Specification Analysis, and Design of Register-Transfer Level Digital Logic. *IEEE Transactions on Computer-Aided-Design*, 2(1):4–18, January 1983.

[HPK87] Y. S. Hong, K. H. Park, and M. Kim. Automatic Synthesis of Data Paths based on the Path- Search Algorithm. In *Proceedings of the International Conference on Computer-Aided-Design*, ACM/IEEE, November 1987.

[JKMP89] R. Jain, K. Kucukcakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proceedings of the 26th Design Automation Conference*, ACM/IEEE, June 1989.

[Kog81] P. M. Kogge. *The Architecture of Pipelined Computers.* McGraw-Hill Publishing Company, New York, 1981.

[KP] K. Kucukcakar and A. C. Parker. MABAL - A Software Package for Module And Bus ALlocation. *International Journal of Computer-Aided VLSI Design.* To appear.

[KP85] D. Knapp and A. Parker. A Unified Representation for Design Information. In *Proceedings of the IFIP Conference on Hardware Description Languages*, IFIP, August 1985.

[KP86] F. J. Kurdahi and A. C. Parker. PLEST: A Program for Area Estimation of VLSI Integrated Circuits. In *Proceedings of the 23rd Design Automation Conference*, ACM/IEEE, June 1986.

[KP87] F. J. Kurdahi and A. C. Parker. REAL: A Program for REgister ALlocation. In *Proceedings of the 24th Design Automation Conference*, ACM/IEEE, June 1987.

[Kun84] S. Y. Kung. On Supercomputing with Systolic/Wavefront Array Processors. *Proceedings of IEEE*, 72(7):531–548, July 1984.

[Kur87] F. J. Kurdahi. *Area Estimation of VLSI Circuits.* PhD thesis, Department of Electrical Engineering, University of Southern California, August 1987.

[Lei81] G. W. Leive. *The Design, Implementation, and Analysis of an Automated Logic Synthesis and Module Selection System.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, January 1981.

[McF78] M. McFarland. *The Value Trace: A Data Base for Automated Digital Design.* Master's thesis, Dept. of Electrical Engineering, Carnegie-Mellon University, December 1978.

[McF81] M. C. McFarland. *Allocating Registers, Processors, and Connections.* Technical Report, Department of Electrical Engineering, Carnegie-Mellon University, August 1981.

[McF83] M. C. McFarland. Computer-Aided Partitioning of Behavioral Hardware Description. In *Proceedings of the 20th Design Automation Conference*, ACM/IEEE, June 1983.

[McF86] M. C. McFarland. Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *Proceedings of the 23rd Design Automation Conference*, ACM/IEEE, June 1986.

[McF87] M. C. McFarland. Reevaluating the Design Space for Register-Transfer Hardware Synthesis. In *Proceedings of the International Conference on Computer-Aided-Design*, ACM/IEEE, November 1987.

[MP88a] M. J. Mlinar and A. C. Parker. *Estimating Register and Multiplexer Costs in VLSI Design.* Technical Report, Department of Electrical Engineering, University of Southern California, 1988.

[MP88b] M. J. Mlinar and A. C. Parker. *Loop Transformations for Acyclic Data Path Synthesis.* Technical Report, Department of Electrical Engineering, University of Southern California, 1988.

[MP88c] M. J. Mlinar and A. C. Parker. *PASTA: A Model for Estimating Control Area.* Technical Report, Department of Electrical Engineering, University of Southern California, 1988.

[MPC88] M. C. McFarland, A. C. Parker, and R. Camposano. Tutorial on High-Level Synthesis. In *Proceedings of the 25th Design Automation Conference*, ACM/IEEE, June 1988.

[MR85] G. L. Miller and J. H. Reif. Parallel Tree Contraction and its Applications. In *Proceedings of the 26th Symposium on Foundations of Computer Science*, ACM, 1985.

[Nes87] J. A. Nestor. *Specification and Synthesis of Digital Systems with Interfaces.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1987.

[NT86] J. A. Nestor and D. E. Thomas. Behavioral Synthesis with Interfaces. In *Proceedings of the International Conference on Computer-Aided-Design*, IEEE/ACM, November 1986.

[Par85] N. Park. *Synthesis of High Speed Digital Systems.* PhD thesis, University of Southern California, August 1985.

[Pau88] P. G. Paulin. *High-Level Synthesis of Digital Circuits Using Global Scheduling and Binding Algorithms.* PhD thesis, Department of Electronics, Carleton University, January 1988.

[PD76] J. H. Patel and E. S. Davidson. Improving the Throughput of a Pipeline by Insertion of Delays. In *Proceedings of the 3rd Annual Symposium on Computer Architecture*, ACM/IEEE, January 1976.

[PG87] B. M. Pangrle and D. D. Gajski. Design Tools for Intelligent Silicon Compilation. *IEEE Transactions on Computer-Aided-Design*, 6(6):1098–1112, November 1987.

[PK] N. Park and F. J. Kurdahi. Module Assignment for Pipelined Data Path Designs. Technical Report, Department of Electrical Engineering, University of California, Irvine, 1988.

[PK87] P. G. Paulin and J. P. Knight. Force-Directed Automatic Datapath Synthesis. In *Proceedings of the 24th Design Automation Conference*, ACM/IEEE, June 1987.

[PK89] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided-Design*, 8(6):661–679, June 1989.

[PP88] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided-Design*, 7(3):356–370, March 1988.

[PPM86] A. C. Parker, J. Pizarro, and M. J. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23rd Design Automation Conference*, ACM/IEEE, June 1986.

[RD87] J. Rabaey and H. DeMan. Computer Aided Design of Digital Signal Processing Systems: The IMEC View. In *Proceedings of the International Conference on Computer Design*, ACM/IEEE, October 1987.

[SJB*88] B. Sharma, R. Jain, M. A. Breuer, A. C. Parker, C. S. Raghavendra, and C. Tseng. The POTATO Chip Architecture: A Study in Tradeoffs for Signal Processing Chip Design. In *Proceedings of the International Conference on Computer Design*, ACM/IEEE, October 1988.

[SLP88] L. -F. Sun, J. -M. Liaw, and T. -M. Parng. Automated Synthesis of Microprogrammed Controllers in Digital Systems. *IEE Proceedings*, 135(4):231–240, July 1988.

[Sno78] E. Snow. *Automation of Module Set Independent Register Transfer Level Design.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1978.

[Sto87] H. S. Stone. *High-Performance Computer Architecture.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[TDW*88] D. E. Thomas, E. M. Dirkes, R. A. Walker, J. V. Rajan, J. A. Nestor, and R. L. Blackburn. The System Architect's Workbench. In *Proceedings of the 25th Design Automation Conference*, ACM/IEEE, June 1988.

[Tho77] D. Thomas. *The Design and Analysis of an Automated Design Style Selector.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1977.

[TL83] D. E. Thomas and G. W. Leive. Automating Technology Relative Logic Synthesis and Module Selection. *IEEE Transactions on Computer-Aided-Design*, 2(2):94–105, April 1983.

[TN83] D. E. Thomas and J. A. Nestor. Defining and Implementing a Multilevel Representation With Simulation Applications. *IEEE Transactions on Computer-Aided-Design*, 2(3):135–145, July 1983.

[Tri85] H. Trickey. *Compiling Pascal Programs into Silicon.* PhD thesis, Department of Computer Science, Stanford University, July 1985.

[Tri87] H. Trickey. Flamel: A High-Level Hardware Compiler. *IEEE Transactions on Computer-Aided-Design*, 6(2):259–269, March 1987.

[TS86] C. -J. Tseng and D. P. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided-Design*, 5(3):379–395, July 1986.

[Wal88] R. A. Walker. *Design Representation and Behavioral Transformation for Algorithmic Level Integrated Circuit Design.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1988.

[YJHN87] F. F. Yassa, J. R. Jasica, R. I. Hartley, and S. E. Noujaim. A Silicon Compiler for Digital Signal Processing: Methodology, Implementation and Applications. *Proceedings of the IEEE*, 75(9):1272–1282, September 1987.