GUIDING DESIGN DECISIONS IN RT-LEVEL LOGIC SYNTHESIS

by

Shiv Prakash

B.Tech., Indian Institute of Technology Kanpur, 1982

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY

April 1987

Approval

Name : Shiv Prakash

Degree : Master of Science

Title of Thesis : Guiding Design Decisions in RT-level Logic Synthesis

Examining Committee:
Chairman: Dr. Binay Bhattacharya

Dr. Louis Hafer
Senior Supervisor

Dr. Pavol Hell

Dr. Joseph G. Peters
External Examiner
School of Computing Science
Simon Fraser University

April 3, 1987
Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

GUIDING DESIGN DECISIONS IN RT-LEVEL LOGIC SYNTHESIS

Author: _____
(signature)

SHIV PRAKASH
(name)

July 29, 1987
(date)

Abstract

This thesis addresses an efficiency problem in a mixed-integer linear programming (MILP) approach to automated synthesis of RT-level digital logic. The approach to the problem is to use simplified design paradigms practiced by human designers to guide the progress of a branch-and-bound MILP algorithm.

Three simplified human design strategies have been considered: storage elements first (SEF), operators first (OPF), and critical path first (CPF). In addition, a few artificial strategies, not directly derivable from human paradigms, have been investigated. The strategies have been incorporated into the branch-and-bound MILP program to guide the solution process. Three examples are examined to investigate the effectiveness of the strategies. Results indicate the approach to be promising in improving the speed of the solution process. Broadly speaking, the simpler strategies SEF and OPF seem to be more effective than the more sophisticated idea of CPF.

Acknowledgements

On the academic side, I owe a great debt of gratitude to my Senior Supervisor, Prof. Lou Hafer. It was he who hooked me on design automation to, couple of years ago, and since then has alternately inspired and dragged me to the point where I can write this page. His support, patience, and invaluable insights have enabled me to complete an endeavor which I thought would never finish. I would also like to thank Prof. Pavol Hell for his constructive comments and suggestions while this thesis was still forming. I am very grateful to Prof. Joe Peters for being my external examiner and for helping with the revision of the thesis. Finally, thanks are also due to Wuyi Wu for her help with some of the examples used in the thesis.

There is life outside academia, however, and I owe thanks to my friend Hafeez who kept me going. Thanks to faculty, staff, and graduate students of the School of Computing Science and all other friends for making my stay in SFU both motivating and enjoyable.

To my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The research reported in this thesis is directed at improving the efficiency of a mixed-integer linear programming (MILP) approach to automated synthesis of the data path for a piece of digital hardware, given a description of the behaviour the hardware is to implement. The approach is to use design paradigms practiced by human designers to guide the progress of a branch-and-bound MILP algorithm.

1.1. Motivation

With the advent of VLSI technology, the functional complexity of a chip has increased drastically. Accordingly, the complexity of the process of VLSI design has increased so that a designer must keep track of vast amounts of information during the design process. It is well known that humans are not very good at handling lots of information at the same time. Therefore, we need to develop systems which will automate the process of hardware design.

The problem faced by digital designers is to come up with a hardware implementation of a digital system which is optimal with respect to some set of design objectives. More often than not the design objectives are in conflict with each other. Therefore, instead of an absolute optimal design we have a set of

noninferior designs, where a noninferior design is defined as an implementation which can be improved with respect to any one objective only at the expense of other objectives. The noninferior designs reflect the tradeoffs among the design goals. The designer has to get to the best solution by generating the whole set of noninferior designs and then selecting the element of the set which embodies the most acceptable tradeoff among the various objectives. The number of noninferior designs is usually large enough to make it almost impossible for an unaided designer to explore all of them because of the design cycle time that has to be met. Therefore, for a CAD/DA system to be really useful, it is imperative that it be able to generate the set of noninferior designs as quickly as possible. In other words, an automated synthesis system must have the ability to produce different implementations quickly.

1.2. The RT-Level Logic Synthesis System

The register-transfer (RT) level logic synthesis system described in [Hafer 81] can be used for the automated synthesis of RT-level data parts for digital systems. Given an RT-level behavioural specification of the digital logic, it develops algebraic relations which express the timing relations that must be satisfied by any correct implementation. The system introduces binary variables into the model to allow the inclusion of implementation decisions. The model, viewed as a system of constraints, is first linearized and then solved as an MILP problem to optimize the objective function supplied by the designer for evaluating candidate

implementations. The MILP solution is found with the BANDBX program [Martin 78], which uses a branch-and-bound technique to solve the problem.

The system is certainly a step towards automating the data part design, however, it is not very successful in terms of producing an implementation quickly. Unspecialized MILP programs like BANDBX require an infeasible amount of computational time for any reasonable size design. The ineffectiveness of unspecialized MILP, in particular the BANDBX program, is mainly due to the fact that it is based on a generic branch-and-bound algorithm. To make the MILP approach practical for large design problems, we have to devise a more specialized branch-and-bound algorithm.

1.3. The Problem and The Approach

The problem is that the RT-level logic synthesis system requires an infeasible amount of computational time, especially if the designer wishes to explore alternative implementations. This thesis will focus on this problem and investigate how to devise a faster branch-and-bound algorithm in order to improve the performance of the system.

One way to improve the efficiency of a branch-and-bound algorithm is to use problem-specific knowledge to guide the progress of the algorithm. This is the approach adopted in this thesis.

Chapter 2

Prior Work

2.1. CAD/DA Work

The RT-level logic synthesis system of [Hafer 81] was developed as part of the CMU-DA (Carnegie-Mellon University - Design Automation) project.

2.1.1. The CMU-DA Project

The CMU-DA project is a major research effort directed at developing a technology-relative structured design aid to help designers explore a large number of alternative implementations for a given digital system. The overall structure proposed for the CMU-DA system is shown in Figure 2-1. The behavioural description of a design is written in the ISPS hardware description language and translated into a parse tree which is then converted into a data flow description called the value trace (VT). The nodes of the VT correspond to operations on data values, and the edges represent the flow of these values between operations. After the design style selector chooses the most suitable design style, the partitioner groups operations from the abstract design representation into control steps. Tradeoffs between the data and control parts are made at this level. A data/memory allocator decides the number and type of functional modules

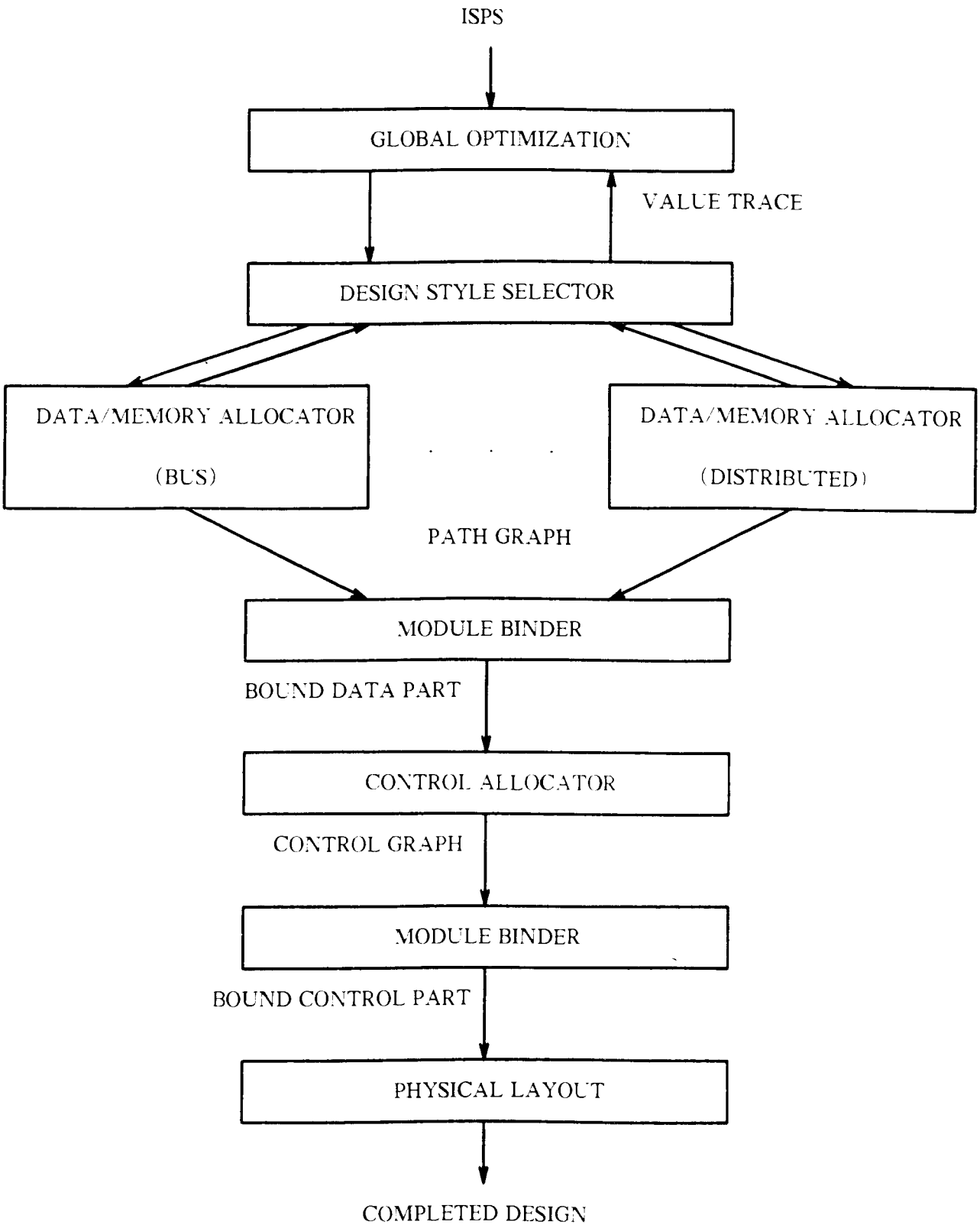

Figure 2-1: CMU-DA System Overview

(operators, registers, data paths and switching functions) needed to implement the data part of the design. A control allocator generates a sequential state machine to control the data part produced by the DM allocator. The module binder then selects the physical modules from the module set library to implement the data and control parts. Finally, a physical DA subsystem handles the physical layout for the proposed implementation and prepares engineering documentation. The interested reader is referred to [Thomas 83] or [Thomas 86] for additional material and references.

2.1.2. RT-Level Logic Synthesis System

In the context of Figure 2-1, the system of [Hafer 81] performs the function of data/memory allocation and control step partitioning simultaneously. The system formalizes the design problem using a set of algebraic relations expressing the constraints on the design and uses constrained optimization techniques to solve the problem. It expresses the data path synthesis problem as a set of algebraic-relation constraints and solves it as an MILP problem. The algebraic relations that make up the constraint set encompass the behaviour the design must support and the performance constraints it must satisfy. The relations are derived from the VT data flow representation.

The VT representation expresses the original RT-level behavioural specification in terms of operators and values. For example, Figure 2-2 is a simple two line ISPS behavioural description specifying two data transfers and Figure 2-3 is the

```
calc :=

  ( A _(AI+1)*(BI+1) next
    B _CI*(BI+1) )
```

Figure 2-2: Example ISPS behavioural description: calc

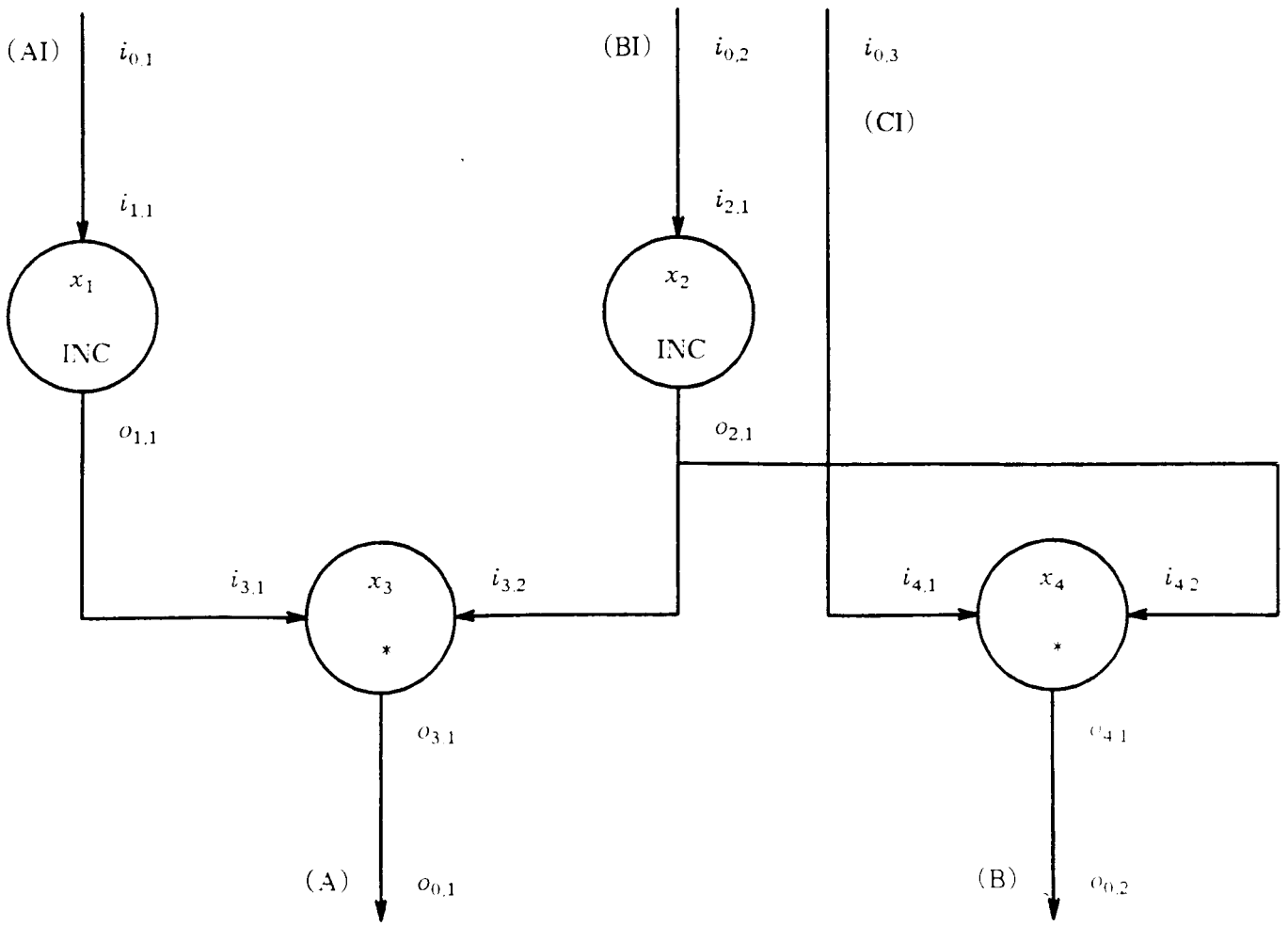

Figure 2-3: VT representation of calc

corresponding VT representation. Figure 2-3 also illustrates the naming convention for components of the VT representation. Operation nodes are labeled x_1, x_2, *etc.*, and x_a indicates an operation in general. The inputs to operation x_a are labeled $i_{a.b}$, and the outputs $o_{a.c}$, where the subscripts b and c distinguish individual values. By convention, a 0 (zero) operation subscript indicates the external environment, so $i_{0.c}$ is an input from the external environment, and $o_{0.b}$ is an output to the external environment.

To write algebraic relations describing the proper timing and sequencing properties required for correct behaviour, the system defines variables to represent certain critical times. These are associated with the operations and values of the VT, and are defined in Table 2-1.

To construct an implementation, the available hardware elements must be known. Two types of hardware elements are considered:

1. Operators labeled f_d (*e.g.*, f_1, f_2).
2. Storage elements labeled s_e (*e.g.*, s_1, s_2).

Hardware elements having both operation and storage capabilities are not allowed. The system needs to know certain timing values associated with the hardware elements available. Table 2-2 shows the set of time delays that are associated with hardware elements. For any specific element, these values are assumed to be available from data sheets or similar sources. For each VT operation x_a, the

Table 2-1: Timing Variables for the Synthesis Model

$T_{IA}(i_{a,b})$	Time when the value required by input $i_{a,b}$ of operation x_a is available for use in the computation.
$T_{XS}(x_a)$	Time when the computation of operation x_a actually begins.
$T_{OA}(O_a)$	Time when the output values O_a computed by operation x_a are available at the outputs of the operator performing the operation.
$T_{XR}(x_a)$	Time when the output values O_a of operation x_a are no longer required as input values for another operation, and thus the time that execution of the operation can cease.
$T_{IR}(I_a)$	Time when the input values for operation x_a are no longer required.
$T_{OR}(o_{a,c})$	Time when output value $o_{a,c}$ is no longer required as an input value for another operation.
$T_{SS}(o_{a,c})$	Time when the storage element assigned to store output value $o_{a,c}$ is clocked.
$T_{SA}(o_{a,c})$	Time when the value $o_{a,c}$ is available at the output of the storage element assigned to store it.
$T_{SR}(o_{a,c})$	Time when the stored copy of value $o_{a,c}$ is no longer required as an input value for an operation.

Table 2-2: Hardware Timing Values

$D_{FP}(f_d)$	Propagation delay time of operator f_d from the appearance of the input value(s) at the operator input(s) to the appearance of the output value(s) at the operator output(s).
$D_{SS}(s_e)$	Setup time at the data input of storage element s_e; data at the input to storage element s_e must be valid for at least this long prior to a transition at the clock input of the storage element.
$D_{SH}(s_e)$	Hold time at the data input of storage element s_e; data at the input of storage element s_e must remain valid for at least this long after the transition at the clock input of the storage element.
$D_{SP}(s_e)$	Propagation delay time of storage element s_e from the transition at the clock input to the appearance of the value at the storage element output.

designer has to specify F_a, the set of operators available to implement the operation, and for each output value $o_{a,c}$, (s)he has to specify $S_{a,c}$, the set of storage elements available to store the value.

The system introduces three types of binary variables (σ, ρ, δ) into the model relations to allow the inclusion of implementation decisions. These variables are used to represent the mapping of the operations and values of the VT onto the operators and storage elements which compose the implementation, and to specify how operation inputs are accessed, as shown in Table 2-3.

Table 2-3: Binary Variables for the Synthesis Model

$\sigma_{d,a}$	Specifies the operation to operator mapping. $\sigma_{d,a}=1$ indicates that operator f_d will implement operation x_a.
$\rho_{e,a,c}$	Specifies the output value to storage element mapping. $\rho_{e,a,c}=1$ indicates that storage element s_e will be used to store output value $o_{a,c}$.
$\delta_{a,b}$	Specifies how the value required by input $i_{a,b}$ is accessed. $\delta_{a,b}=1$ indicates the stored copy of the value is accessed.

The model consists of three types of algebraic relations:

1. Relations which model the execution of an operation by an operator:

   Some relations in this category express necessary conditions for a correct implementation. *e.g.*, one and only one operator should be selected to implement an operation:

$$\sum_{d|f_d \epsilon F_a} \sigma_{d,a} = 1.$$

   Other relations express the relationships between various timing variables based on Figure 2-4, *e.g.*, the outputs of an operator are valid after a propagation delay:

$$T_{OA}(O_a) = T_{XS}(x_a) + \sum_{d|f_d \epsilon F_a} \sigma_{d,a} D_{FP}(f_d).$$

2. Relations which model storing a value in a storage element:

   These relations express the relationships between various timing variables based on Figure 2-5, *e.g.*, the outputs of a storage element are valid after a propagation delay:

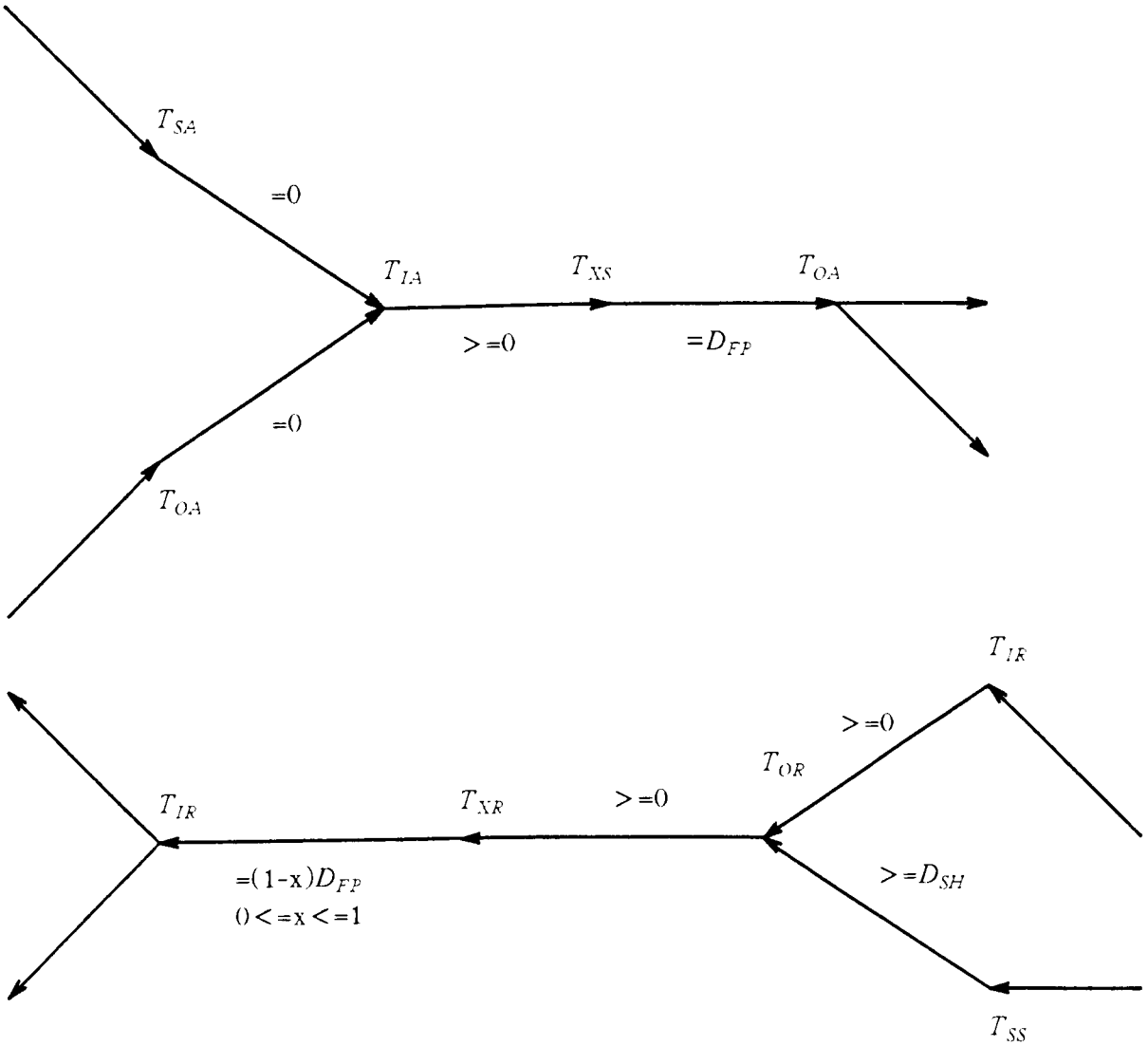

Figure 2-4: Timing relations for operation execution

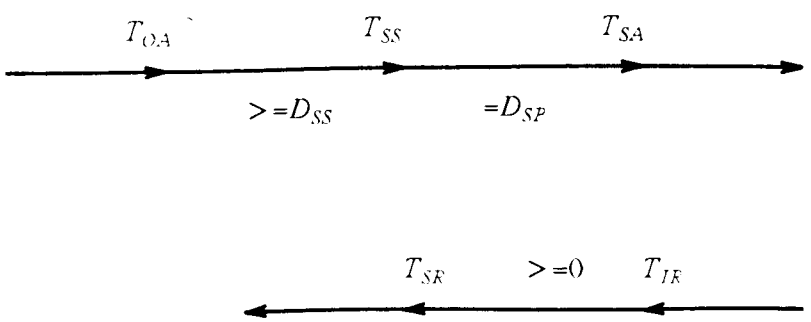

Figure 2-5: Timing relations for storing a value

$$T_{SA}(o_{a,c}) = T_{SS}(o_{a,c}) + \sum_{e|s_e \epsilon S_{a,c}} \rho_{e,a,c} D_{SP}(s_e).$$

3. Relations which ensure that hardware resources are shared correctly:

   These relations are necessary to prevent overlaps in the scheduling of events on hardware elements.

Thus, the system formally models digital systems by algebraic relations. The system finds a simultaneous solution of the algebraic relations as a constrained optimization problem and produces a globally optimal design with respect to the objective function. The set of relations is first linearized and then solved as an MILP problem, yielding both a design for the data part and a timing specification which details how the behavioural actions are to be executed on the data part.

2.2. MILP Work

There has been a lot of research effort aimed at improving the effectiveness of branch-and-bound algorithms. As we know, the branch-and-bound technique involves searching through a tree whose leaves are the candidate solutions to the problem. The algorithm reaches the optimal solution faster if it searches through the appropriate branches first. So, the criteria used for branching in the algorithm plays a very important role in the success of the algorithm. A survey of integer programming computer codes [Land 79] shows that a wide variety of heuristic ideas and rules has been used to select the variable to branch upon in various commercial codes. One of the rules which is fairly common is based on the idea of priorities.

It has been observed [Mitra 73] that branching upon variables in order of their importance, in some sense, can accelerate the progress of the algorithm. For example, some large scale hierarchical planning models involving binary variables, which could not be solved with any standard techniques, were solved when the variables were ordered by importance, *i.e.*, advantage was taken of the hierarchical structure of variables [Johnson 85]. As is obvious, to organize variables in the order of importance we require problem-specific knowledge. So, it seems that using knowledge about the problem to guide the branching process can enhance the speed of the algorithm.

Chapter 3

The Approach and The Design Strategies

The synthesis model requires many relations and variables to formally specify the constraints and objectives for a design. As a matter of fact, if n is a composite estimate of the number of operators and values in the VT, and h is a composite estimate of the number of hardware elements available for the implementation, then the number of algebraic relations grows as $O(hn^2)$, the number of continuous variables grows as $O(n)$ and the number of binary variables grows as $O(hn^2)$ [Hafer 81]. The BANDBX program is able to produce optimal results for small designs, but with very large solution times. The growth rate of the constraint system makes unspecialized MILP impractical for non-trivial designs.

To reduce the computational time required by the synthesis system, it is mandatory to devise an efficient branch-and-bound technique which, when incorporated into BANDBX, will make it faster.

3.1. The Approach

Before we outline the approach to be taken, let us have a quick discussion of the branch-and-bound technique. It involves the following four major steps:

STEP 1: Solve the LP relaxation for a bound.

STEP 2: Try to fathom the current subproblem due to the bound or penalties.

STEP 3: Pick a fractional-valued binary variable and force it to 0 and 1, creating two subproblems.

STEP 4: Solve each subproblem in turn.

The phrase 'fractional-valued binary variable' means a variable that assumes a non-integer value between 0 and 1 in the solution to the current subproblem, but must assume a binary value in a feasible solution to the given MILP problem. Steps 1, 2, and 4 are fairly straightforward and there is not much one could do about them. Step 3 seems to be crucial as far as efficiency of the technique goes because it involves a choice. At a point in time, there will usually be more than one fractional-valued binary variable, and which one to pick for forcing is a decision to be made based on certain criteria. Of course, the more effective the criteria used, the better the performance of the algorithm will be. As discussed earlier, one type of criterion used is based on the idea of organizing the binary variables in the order of importance. This is the approach undertaken in this research.

Now, the question is how one goes about ordering the variables by importance. The answer is that one has to know about the roles of the various variables in the problem being considered. If we look at the synthesis model, the binary variables were introduced into the model to represent implementation decisions. Forcing a binary variable either way implies that a design decision is being made, one way or the other. Thus, ordering binary variables by importance is equivalent to ordering design decisions by importance. This is where the design knowledge has to be used. A human designer makes various design decisions in certain order, *i.e.*, (s)he makes what (s)he considers more important decisions first. The designer usually follows an order of decisions dictated by a dynamic design strategy. The exact design strategy used by a human designer is not very well understood and is certainly complex. It is therefore worthwhile to investigate first the performance gains which can be obtained from simplified design strategies. The following are some simplified design strategies worth considering:

1. Storage elements first.
2. Operators first.
3. Critical path first.

The research investigates the effectiveness of each of these strategies in guiding step 3 of the general branch-and-bound algorithm discussed on page 16. A few more artificial strategies are investigated for the sake of completeness.

3.2. Discussion of the Design Strategies

3.2.1. Generalized Design Strategy Used by People

The question of how people do designs is not very well answered. The design strategy used by a human designer is certainly complex and dynamic. The process of design is not a formalized one; it involves a heuristic approach. The implementation is constructed using a collection of independent rules and procedures which are applied on a case-by-case basis until the designer is satisfied that the design constraints have been met. The rules and procedures are applied in different orders for different examples. For a given design example, the order is not predetermined. It is determined dynamically in the design process. The next rule to be applied is dependent on how the design has progressed so far. Eventually, the designer stops when (s)he feels that all the design constraints have been met.

The point made by the above discussion is that the process of design is really complex and it is hard to exactly outline the way people do designs. One basic idea which is very relevant in this context is the idea of "focusing attention". The whole process of design seems to be revolving around this idea. In any reasonable size design, there are so many details involved that it is almost impossible for a human designer to keep track of the overall global view of the design at all times. Therefore, people try to break a bigger design into smaller designs. At a given time, they focus attention on a certain part of the design. When it is done, the

designer moves on to another part. The order in which various parts of the design are considered is decided by the designer based on his idea of the importance of various parts. The fact that the designer's idea of the importance of various parts is dynamic contributes to the complexity of the design strategy. The next part to be considered is dependent on how the design has progressed so far.

Some of the considerations in deciding the importance of various parts are visible architectural values, functionality, and performance of the design. In many design situations, there are certain architectural values visible to the outside world and it might be a good idea to work on these first, *e.g.*, in a processor design, the designer might decide first on the storage elements representing the architectural registers of the processor. In some other situations, the correct functionality of certain parts might be more important, or the performance of the overall system, say, in terms of the execution time. The important point is that none of the considerations are used in isolation in a given situation. All the considerations are entangled in the decision function of a human designer and so (s)he ends up doing a bit of this and a bit of that. *E.g.*, one scenario could be where a human designer picks up a part of the data path which (s)he thinks is crucial to the performance of the overall system, then decides on some operators and storage elements to be used in this part, then realizes that the rest of the decisions regarding this part could be made more efficiently if the design of another part of the data path were known because of the interactions between the two parts. So

(s)he shifts focus, designs that part, comes back to finish the design of this part, then picks up some other less important part, and so on. So, the whole process or the strategy is very entangled in terms of the order of decisions. The general design strategy used by people being so complex, let us consider some of its simplifications. They are discussed in the next subsection.

3.2.2. Simplified Design Strategies

The various design strategies investigated for the guidance of the MILP solution process are:

1. Storage elements first.
2. Operators first.
3. Critical path first.

These are, in some sense, simplifications of the more general strategy discussed in the previous subsection.

3.2.2.1. Storage Elements First

This strategy is based on the idea that the decisions regarding visible architectural values are more important. The strategy says that the decisions regarding storage elements should be made first. The decisions as to whether or not a given value in the VT description is to be stored and, if yes, what storage element is to be used to store it, are made first and only then are the decisions regarding operators considered. This is a very simplified approach, however, it is

used in practice sometimes. The effectiveness of this simplified approach in guiding the branch-and-bound procedure is investigated.

In the context of the synthesis model, the strategy implies that the binary variables related to storage decisions should be given higher priority compared to those related to operator decisions. In other words, the variables of type ρ and type δ are considered more important than the variables of type σ in the branch-variable selection process. The strategy by itself does not dictate the order of importance between type ρ and type δ. So, two strategies corresponding to the following permutations are investigated:

1. ρ, δ, σ
2. δ, ρ, σ

3.2.2.2. Operators First

This strategy tries to get the correct functionality first. In this strategy, decisions regarding operators are made first, *i.e.*, decisions as to what operators are to be used and what operators are to be shared. Only after operators are decided are storage elements considered. This strategy is less commonly used in practice. However, it represents an exact counterpoint to the first strategy and thus is of interest in this research.

In the context of the synthesis model, the variables of type σ are considered more important than the variables of type δ and type ρ. Again, two strategies corresponding to the following permutations are investigated:

1. σ, δ, ρ

2. σ, ρ, δ

3.2.2.3. Critical Path First

This strategy says that first the attention should be focussed on the path which is crucial to the performance of the overall design in terms of the execution time. A critical path in a VT description is the one which has the longest timing. Clearly this path determines the minimum total execution time and thus is critical to the performance of the data part. Hence, the decisions regarding the allocation of hardware to perform operations on the critical path of the data part are more important. In the critical path first strategy, these decisions are made first and then the decisions about other paths are considered. This is a fairly commonly used strategy in practice and therefore is investigated.

In the context of the synthesis model, of course, the binary variables related to the critical path would be considered more important than the others. We can incorporate some more ideas within the general idea of critical path strategy. The variables related to the critical path can be ordered within themselves based on a certain strategy, *e.g.*, Storage Elements First or Operators First. The following seven different orderings of variables within the critical path are investigated (the variables related to the non-critical part are not ordered within themselves):

1. Simple Critical Path Strategy: In this case, there is no ordering within the critical path. All the binary variables are divided into two groups. The critical path group has higher priority than the other one.

2. $\rho\delta\sigma$ Critical Path Strategy: Here, the variables within the critical path are divided into three groups, each group representing a type (σ, ρ, or δ). Priorities are dictated by the ρ, δ, σ order.

3. $\delta\rho\sigma$ Critical Path Strategy: Again, the variables within the critical path are divided into three groups and priorities are dictated by the δ, ρ, σ order.

4. $\sigma\delta\rho$ Critical Path Strategy: Here, the priorities within the critical path are dictated by the σ, δ, ρ order.

5. $\sigma\rho\delta$ Critical Path Strategy: Here, the priorities within the critical path are in the σ, ρ, δ order.

6. Forward Propagation Critical Path Strategy: Any particular data path in the VT representation has a direction associated with it, the direction in which the data (or value) propagates along the path. The binary variables related to a path can be associated with different points on the path. So, we can order the variables associated with the critical path by traversing the path in the direction of data propagation. The order thus generated dictates the priorities in this strategy.

7. Reverse Propagation Critical Path Strategy: The order within the critical path here is exactly the reverse of that in forward propagation, *i.e.*, we traverse the path in the direction opposite to that of data propagation.

3.2.3. Some Artificial Strategies

As we saw in the previous subsection, we are going to investigate different permutations and combinations based on certain simplified ideas. We are considering four different permutations for the δ, ρ, σ ordering. We did not consider the following two permutations:

1. ρ, σ, δ

2. δ, σ, ρ

They do not exactly correspond to any of the strategies discussed in the previous subsection and so we call them artificial strategies. For the sake of completeness, we investigate these too.

Again, for the sake of completeness, we combine these two permutations with the critical path idea and investigate the following two strategies (with obvious definitions):

1. $\rho\sigma\delta$ Critical Path Strategy.
2. $\delta\sigma\rho$ Critical Path Strategy.

So, overall we are investigating sixteen different strategies to guide the branch-and-bound process. For a quick reference, they are listed in Table 3-1, containing an abbreviated as well as an explanatory name for each strategy.

Table 3-1: Strategies to Guide the Branch-and-Bound

NG	No guidance (original BANDBX program).
$\rho\delta\sigma$S	Simple ρ, δ, σ ordering.
$\delta\rho\sigma$S	Simple δ, ρ, σ ordering.
$\sigma\delta\rho$S	Simple σ, δ, ρ ordering.
$\sigma\rho\delta$S	Simple σ, ρ, δ ordering.
$\rho\sigma\delta$S	Simple ρ, σ, δ ordering.
$\delta\sigma\rho$S	Simple δ, σ, ρ ordering.
SCP	Simple Critical Path strategy.
$\rho\delta\sigma$CP	ρ, δ, σ ordering within the Critical Path.
$\delta\rho\sigma$CP	δ, ρ, σ ordering within the Critical Path.
$\sigma\delta\rho$CP	σ, δ, ρ ordering within the Critical Path.
$\sigma\rho\delta$CP	σ, ρ, δ ordering within the Critical Path.
$\rho\sigma\delta$CP	ρ, σ, δ ordering within the Critical Path.
$\delta\sigma\rho$CP	δ, σ, ρ ordering within the Critical Path.
FPCP	Forward Propagation Critical Path Strategy.
RPCP	Reverse Propagation Critical Path Strategy.

Chapter 4

Experiments and Results

To investigate the effectiveness of the strategies discussed in the previous chapter in guiding the branch-and-bound, we solved a few synthesis examples using each strategy. The BANDBX program did not have the flexibility to allow these strategies to be incorporated into the branch-variable selection process. To implement the strategies, some modifications were made to the BANDBX program. A few words about the BANDBX program and the modifications are in order.

4.1. The BANDBX Program

The BANDBX program is an enumeration code for pure and mixed zero-one linear programming problems. It is based upon a branch-and-bound algorithm. After the linear program defined at a given subproblem has been solved, the solution is checked for possible fathoming by infeasibility, bounds, integrality, and penalties. If the subproblem cannot be fathomed, the code then proceeds to either fix all monotone variables (fractional-valued binary variables that can assume only one value in successor subproblems due to feasibility or penalties) or, if no monotone variables are present, to branch on a fractional-valued binary variable.

Two penalties are calculated for each fractional-valued binary variable, one by

forcing it to 0 and the other by forcing it to 1. The choice of branching variable and direction is made by selecting the largest among all penalties and branching on the corresponding variable in the direction opposite to that yielding the maximum penalty. As is obvious, there is no consideration, whatsoever, of the importance of variables in the BANDBX code. Appropriate modifications were made and a few subroutines were written to allow the inclusion of importance of variables in the branch-variable selection process. Then each of the strategies in Table 3-1 was incorporated into the program to dictate importance of the variables.

4.2. Measure of Effectiveness of a Strategy

Now, the question is how to evaluate the effectiveness of different strategies. The BANDBX program compiles and reports an extensive set of statistics detailing the solution process. The following pieces of information are particularly relevant to our experiments:

1. Number of the subproblem at which the optimal solution was found (SPOPT#).
2. Total number of subproblems solved (TOTSP#).
3. Total number of simplex pivots performed to solve the linear relaxations (TSIMP#).

The three numbers give a good indication of how much time the branch-and-bound process took. The magnitude of the numbers certainly reflects upon how effective the strategy was in pruning the branch-and-bound tree. We will use these numbers in evaluating the effectiveness of strategies.

4.3. Synthesis Examples

The following three examples were solved:

1. Criss.Cross Example
2. Logic Example
3. Critical Path Example

Results regarding the time taken by BANDBX to solve the problems, when guided by various strategies, are given here. Detailed analysis and discussion of the results follows in the next chapter.

4.3.1. Criss.Cross Example

The ISPS behavioural description and VT representation for the example are shown in Figure 4-1 and Figure 4-2 respectively.

```
Criss.Cross :=

BEGIN

  ** Carriers **
  t1<0:15> .
  t2<0:15> .
   a<0:15> .
   b<0:15> .

  ** Activity **
  action :=
  ( t1 _a+b next
    t2 _a-b next
    a _t1+t2 next
    b _t1-t2 )

END
```

Figure 4-1: Criss.Cross ISPS Behavioral Description

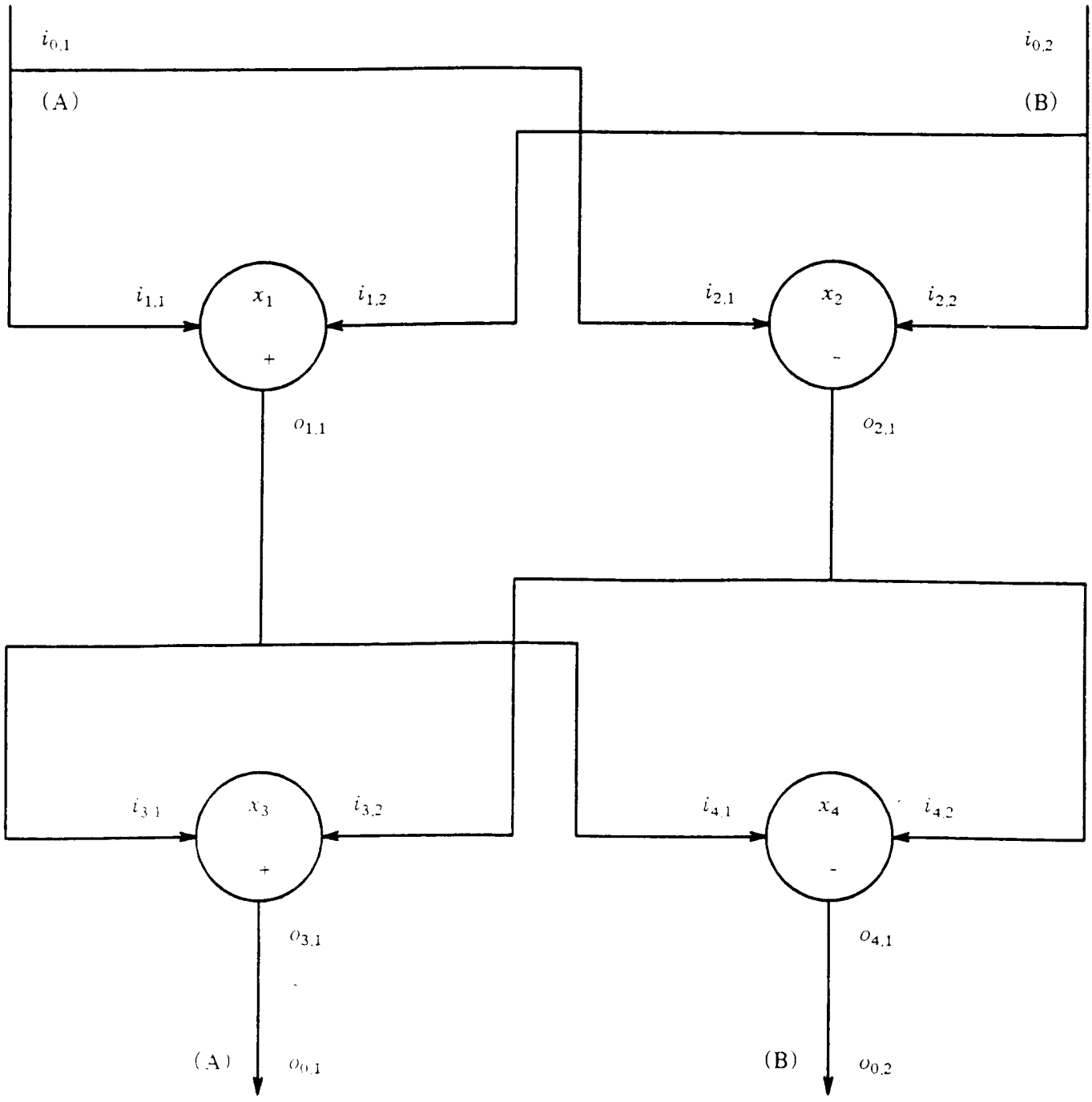

Figure 4-2: Criss.Cross VT representation

The form of the implementation has been somewhat arbitrarily restricted, as shown in Figure 4-3.

$S_{0,1} = \{s_1\}$
$S_{0,2} = \{s_2\}$
$S_{1,1} = \{s_1, s_2, s_3\}$
$S_{2,1} = \{s_1, s_2, s_3\}$
$S_{3,1} = \{s_1\}$
$S_{4,1} = \{s_2\}$
$F_1 = \{f_1, f_5\}$
$F_2 = \{f_3, f_5, f_6\}$
$F_3 = \{f_1, f_2, f_5, f_6\}$
$F_4 = \{f_3, f_4, f_5, f_6\}$

$T_{OA}(i_{0,1}) = 0$
$T_{OA}(i_{0,2}) = 0$
$T_{OR}(i_{0,1}) \leqslant 100ns$
$T_{OR}(i_{0,2}) \leqslant 100ns$
$T_{SS}(i_{0,1}) = T_{SS}(i_{0,2})$
$\delta_{0,1} = 1$
$\delta_{0,2} = 1$
$\rho_{1,3,1} = 1$
$\rho_{2,4,1} = 1$

Figure 4-3: Restrictions on Implementation of Criss.Cross

The details of the hardware set available for implementation are given in Table 4-1. The period during which the inputs I_0 are valid is restricted to the interval 0 to 100 ns., and we have assumed a single control signal to latch the inputs. As shown in Figure 4-3, we have required the output values O_0 to be accessed from register outputs, and forced the values $o_{3,1}$ and $o_{4,1}$ to be stored, by setting the variables $\delta_{0,1}$, $\delta_{0,2}$, $\rho_{1,3,1}$, and $\rho_{2,4,1}$ to 1.

The synthesis model incorporating the restrictions mentioned was developed. The model consists of 112 equations involving 72 variables of which 35 are binary variables. The model was solved for the following three implementations:

1. Implementation optimizing the cost: Here, it was assumed that the performance requirements were that the outputs had to be valid at the time t=800 ns. and remain valid until the time t=900 ns. The model was solved, with cost as the objective function, using each of the

Table 4-1: Hardware Elements for Criss.Cross Implementation

storage	bits	D_{SS}	D_{SH}	D_{SP}	cost
s_1	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_2	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_3	<16>	20 ns.	0 ns.	27 ns.	$8.10
operator	**bits**	**function**		D_{FP}	**cost**
f_1	<16>	+		70 ns.	$14.20
f_2	<16>	+		70 ns.	$14.20
f_3	<16>	-		85 ns.	$24.18
f_4	<16>	-		85 ns.	$24.18
f_5	<16>	ALU		107 ns.	$19.00
f_6	<16>	ALU		107 ns.	$19.00

strategies listed in Table 3-1. Statistics regarding the time taken by BANDBX for different strategies are listed in Table A-1.

2. Implementation optimizing the performance: The model was solved, with output availability time as the objective function, using each of the strategies. Statistics regarding the time taken by BANDBX are listed in Table A-2.

3. Implementation optimizing the cost, with stringent performance requirements: Here, the performance requirements were that both outputs be available in under 500 ns., and remain valid until 600 ns. The model was modified to include the requirements and then solved with cost as the objective function. Statistics about the solution time are given in Table A-3.

Criss.Cross, having such a symmetric VT representation, does not contain a real critical path which can be considered crucial to the performance of the design.

However, usually a '-' operation takes a little longer than a '+' operation (also indicated by D_{FP} values in Table 4-1) and therefore we have considered the path including $i_{0.2}.x_2.x_4.o_{0.2}$ to be the critical path in the guidance strategies.

4.3.2. Logic Example

The ISPS behavioural description and VT representation for this example are shown in Figure 4-4 and Figure 4-5 respectively.

```
Logic :=

BEGIN

** Carriers **
A<0:63> ,
B<0:15> ,
C<0:15> ,

** Activity **
action :=
( B _(C AND (B OR A<48:63>)) XOR (B AND (C OR A<8:23>)) )

END
```

Figure 4-4: Logic ISPS Behavioral Description

In Figure 4-5, operations x_1 and x_4 are field extraction operations which produce as outputs a subfield of the value given as the input. Here, operation x_1 produces A<48:63> and operation x_4 produces A<8:23>.

The restrictions we will place on the implementation are shown in Figure 4-6 and the details of the hardware set are shown in Table 4-2. Again, we assume that the inputs I_0 are available only during the interval 0 to 100 ns, and that only one signal is available to latch the inputs into storage elements. To ensure

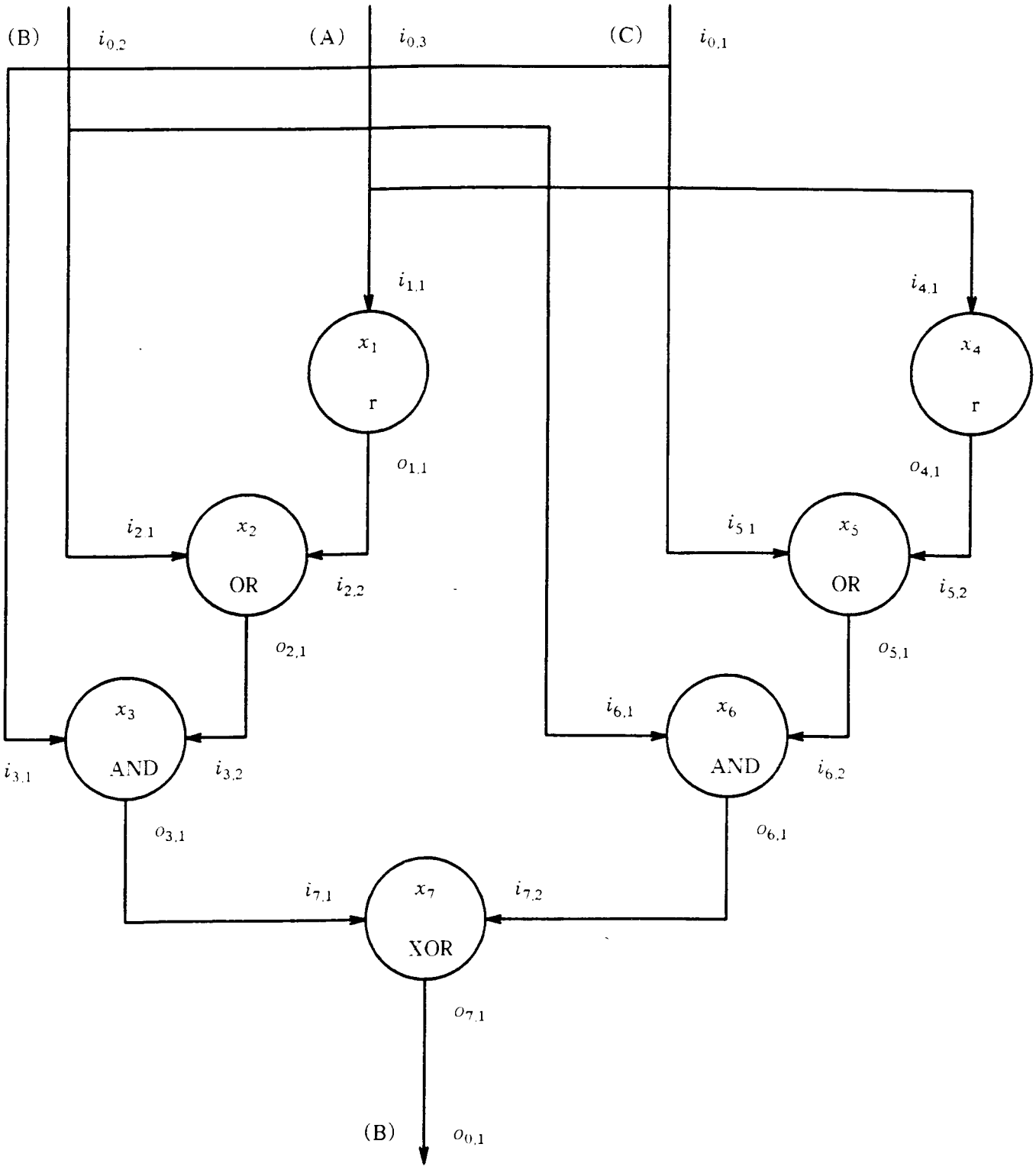

Figure 4-5: Logic VT representation

$S_{0.1} = \{s_1\}$
$S_{0.2} = \{s_2\}$
$S_{0.3} = \{s_3\}$
$S_{2.1} = \{s_2, s_3\}$
$S_{3.1} = \{s_1, s_2, s_3\}$
$S_{5.1} = \{s_1, s_3\}$
$S_{6.1} = \{s_1, s_2, s_3\}$
$S_{7.1} = \{s_1, s_2, s_3\}$
$F_2 = \{f_1\}$
$F_3 = \{f_1\}$
$F_5 = \{f_1, f_2\}$
$F_6 = \{f_1, f_2\}$
$F_7 = \{f_1\}$

$T_{OA}(i_{0.1}) = 0$
$T_{OA}(i_{0.2}) = 0$
$T_{OA}(i_{0.3}) = 0$
$T_{OR}(i_{0.1}) \leqslant 100ns$
$T_{OR}(i_{0.2}) \leqslant 100ns$
$T_{OR}(i_{0.3}) \leqslant 100ns$
$T_{SS}(i_{0.1}) = T_{SS}(i_{0.2}) = T_{SS}(i_{0.3})$

Figure 4-6: Restrictions on Implementation of Logic

Table 4-2: Hardware Elements for Logic Implementation

storage	bits	D_{SS}	D_{SH}	D_{SP}	cost
s_1	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_2	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_3	<64>	20 ns.	0 ns.	27 ns.	$32.40
operator	**bits**	**function**		D_{FP}	**cost**
f_1	<16>	ALU		48 ns.	$19.00
f_2	<16>	ALU		48 ns.	$19.00

satisfactory performance, we require that the period during which the output $o_{0.1}$ is valid is not less that 100 ns. No operations are required to implement operations x_1 and x_4, since these operations are performed by simply connecting a data path to the proper bits of the value $i_{0.3}$.

The synthesis model incorporating the restrictions was developed. It consists of 150 equations involving 85 variables of which 38 are binary variables. The model was solved for the following two implementations:

1. Implementation optimizing the cost: The model was solved with cost as the objective function and the BANDBX statistics are given in Table A-4.

2. Implementation optimizing the performance: Here output availability time was the objective function. The statistics are listed in Table A-5.

Logic too, having such a symmetric VT representation, does not contain a real critical path. For the lack of better choice, we have considered the path including $i_{0.1}.x_5.x_6.x_7.o_{0.1}$ to be the critical path.

4.3.3. Critical Path (CPath) Example

As we saw, the previous two examples (Criss.Cross and Logic) did not have real critical paths. Intuitively, one would expect any strategy related to the idea of critical path to be more effective if there were a real critical path in the design. To investigate this intuition, we solved an example whose VT representation is not symmetric and has a critical path embedded in it.

The ISPS behavioural description and VT representation for this example are shown in Figure 4-7 and Figure 4-8 respectively. The restrictions we will place on the implementation are shown in Figure 4-9 and the details of the hardware set are shown in Table 4-3. Again, the period during which the inputs I_0 are valid is

```
CPath :=

BEGIN

  ** Carriers **
  t1<0:15> ,
  t2<0:15> ,
  t3<0:15> ,
   a<0:15> ,
   b<0:15> ,

  ** Activity **
  action :=
  ( t1 _a+a next
    t2 _b+b next
    t3 _t2+b next
    a _t1+t3 )

END
```

Figure 4-7: CPath ISPS Behavioral Description

restricted to the interval 0 to 100 ns., and we have assumed a single control signal to latch the inputs. To ensure satisfactory performance, we require that the period during which the output $o_{0.1}$ is valid is not less that 100 ns.

The synthesis model incorporating the restrictions consists of 152 equations involving 94 variables of which 53 are binary. The model was solved for the following three implementations:

1. Implementation optimizing the cost: The model was solved with cost as the objective function and the statistics are given in Table A-6.

2. Implementation optimizing the performance: The model was solved with output availability time as the objective function. The statistics are listed in Table A-7.

3. Implementation optimizing the cost, with stringent performance

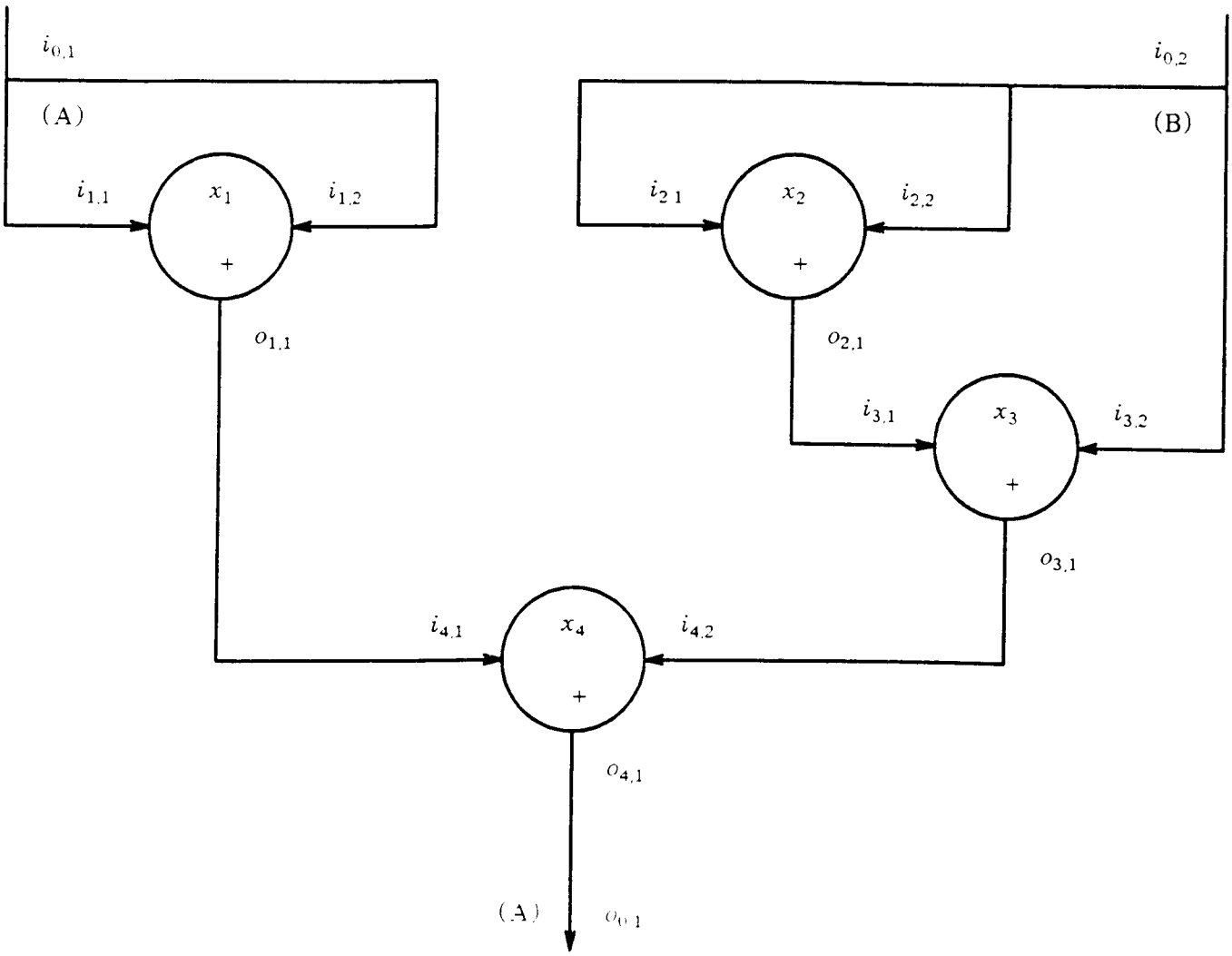

Figure 4-8: CPath VT representation

$S_{0.1} = \{s_1, s_2, s_3\}$
$S_{0.2} = \{s_1\}$
$S_{1.1} = \{s_1, s_2, s_3\}$
$S_{2.1} = \{s_2\}$
$S_{3.1} = \{s_1, s_2, s_3\}$
$S_{4.1} = \{s_1, s_2, s_3\}$
$F_1 = \{f_1, f_5\}$
$F_2 = \{f_1, f_2, f_5, f_6\}$
$F_3 = \{f_1, f_2, f_3, f_5, f_6, f_7\}$
$F_4 = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$

$T_{OA}(i_{0.1}) = 0$
$T_{OA}(i_{0.2}) = 0$
$T_{OR}(i_{0.1}) \leqslant 100ns$
$T_{OR}(i_{0.2}) \leqslant 100ns$
$T_{SS}(i_{0.1}) = T_{SS}(i_{0.2})$

Figure 4-9: Restrictions on Implementation of CPath

requirements: Here, the performance requirement was that the output be available in under 480 ns. The model was modified to include the requirement and then solved with cost as the objective function. Statistics about the solution time are given in Table A-8.

Obviously, the critical path used for guidance in this example was the path including $i_{0.2}, x_2, x_3, x_4, o_{0.1}$.

Table 4-3: Hardware Elements for CPath Implementation

storage	bits	D_{SS}	D_{SH}	D_{SP}	cost
s_1	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_2	<16>	20 ns.	0 ns.	27 ns.	$8.10
s_3	<16>	20 ns.	0 ns.	27 ns.	$8.10
operator	bits	function		D_{FP}	cost
f_1	<16>	+		50 ns.	$30.00
f_2	<16>	+		50 ns.	$30.00
f_3	<16>	+		50 ns.	$30.00
f_4	<16>	+		50 ns.	$30.00
f_5	<16>	+		100 ns.	$10.00
f_6	<16>	+		100 ns.	$10.00
f_7	<16>	+		100 ns.	$10.00
f_8	<16>	+		100 ns.	$10.00

Chapter 5

Observations and Conclusions

Of the numbers recorded for the synthesis examples, the total number of subproblems solved (TOTSP#) is a good representative of the time taken by BANDBX to solve the problem. It reflects the effectiveness of the strategy used quite well (the smaller the number of subproblems solved, the better the strategy for pruning the branch-and-bound tree). To get a better idea of the performance of the strategies with respect to the unguided BANDBX program, TOTSP# for each implementation solution is normalized with respect to the TOTSP# for the NG strategy. The normalized TOTSP#'s are listed in Table 5-1. The 'NOSF' entries (No Optimal Solution Found) in the table indicate the failure of BANDBX to find an optimal solution when guided by the strategy. The normalized TOTSP#'s are averaged over the various implementation examples for each strategy. Of course, 'NOSF' entries are ignored in the averaging process. The average normalized number of subproblems solved is thus calculated for each strategy and then plotted in a bar chart shown in Figure 5-1.

Table 5-1: Normalized Number of Subproblems Solved (1 of 2)

Strategy	Criss.Cross Cost	Criss.Cross Cost-Time	Criss.Cross Time	Logic Cost
NG	1.00	1.00	1.00	1.00
$\sigma\delta\rho$S	0.67	0.47	0.42	1.15
$\delta\sigma\rho$S	0.92	0.80	0.25	1.17
$\delta\rho\sigma$S	0.99	0.98	0.26	1.16
$\sigma\rho\delta$S	2.58	1.39	NOSF	2.30
$\rho\sigma\delta$S	2.57	1.72	NOSF	2.38
$\rho\delta\sigma$S	4.54	2.90	NOSF	2.44
SCP	0.67	0.64	0.80	1.09
$\sigma\delta\rho$CP	0.66	0.59	0.73	1.27
$\delta\sigma\rho$CP	1.16	0.70	0.73	1.29
$\delta\rho\sigma$CP	1.32	0.80	0.73	1.33
$\sigma\rho\delta$CP	1.31	1.03	1.12	1.78
$\rho\sigma\delta$CP	1.23	0.78	1.12	1.80
$\rho\delta\sigma$CP	1.78	0.97	1.12	1.79
FPCP	1.48	0.96	1.12	NOSF
RPCP	0.56	0.59	0.73	1.58

Table 5-1: Normalized Number of Subproblems Solved (2 of 2)

Logic Time	CPath Cost	CPath Cost-Time	CPath Time	Average
1.00	1.00	1.00	1.00	1.00
0.20	0.29	0.36	0.09	0.46
0.23	0.27	0.41	0.04	0.51
0.33	0.29	0.45	0.03	0.56
NOSF	NOSF	NOSF	NOSF	2.09
NOSF	NOSF	NOSF	NOSF	2.22
0.89	NOSF	NOSF	NOSF	2.69
0.58	0.84	0.51	0.33	0.68
NOSF	0.34	NOSF	0.08	0.61
0.37	0.33	NOSF	0.05	0.66
NOSF	0.37	NOSF	0.05	0.77
NOSF	1.05	0.64	NOSF	1.16
NOSF	NOSF	NOSF	NOSF	1.23
NOSF	NOSF	NOSF	NOSF	1.42
NOSF	1.24	NOSF	NOSF	1.20
0.05	NOSF	0.52	0.11	0.59

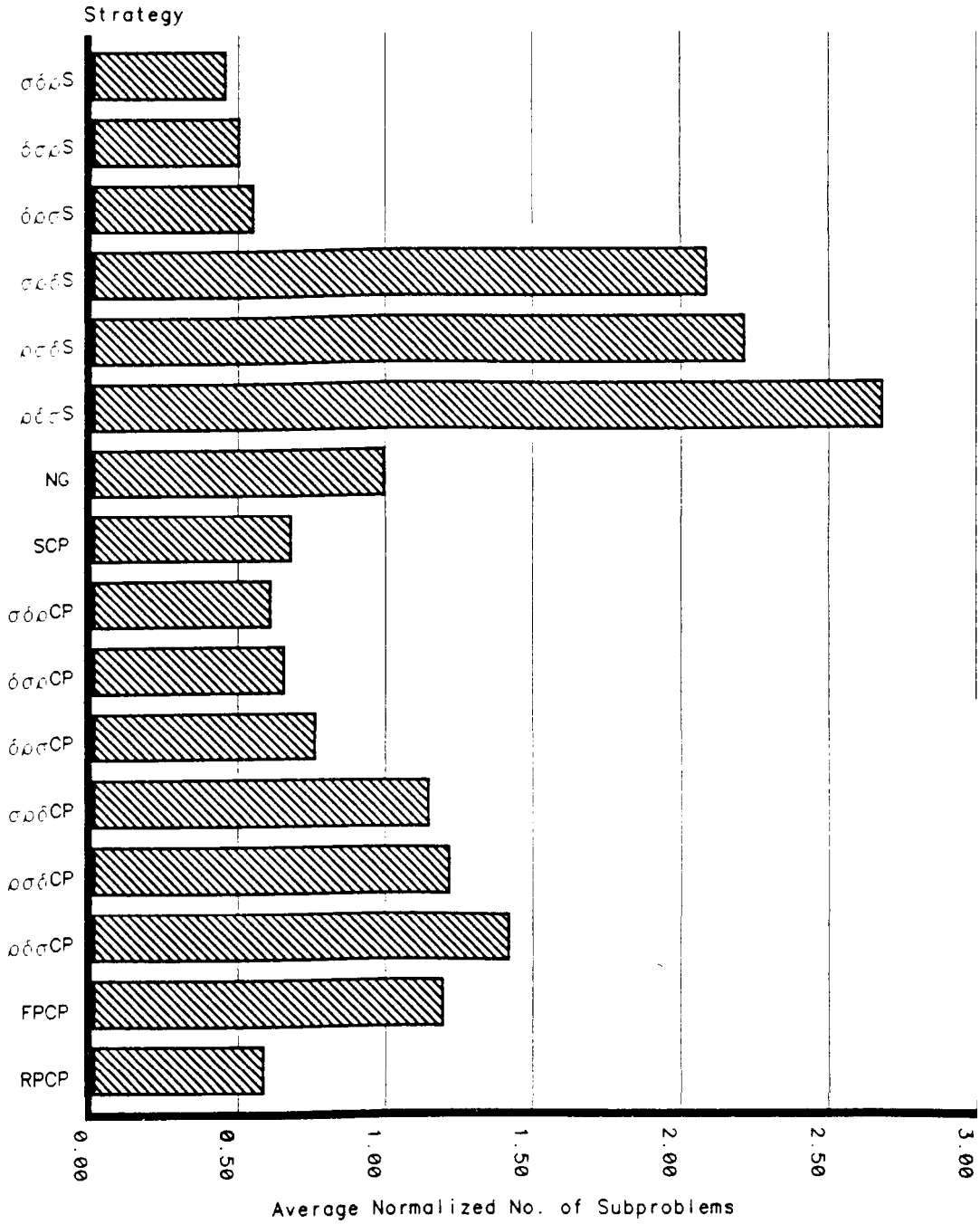

Figure 5-1: Comparison of Strategies

5.1. Observations

5.1.1. General Observations

We can make the following observations from Figure 5-1:

- The simple ordering strategies can be divided into two classes:

    1. Strategies assigning higher importance to variables of type δ than to those of type ρ (δ, ρ order): $\sigma\delta\rho$S, $\delta\sigma\rho$S and $\delta\rho\sigma$S strategies tend to improve the performance of BANDBX by a factor of two on an average.

    2. Strategies assigning higher importance to variables of type ρ than to those of type δ (ρ, δ order): $\sigma\rho\delta$S, $\rho\sigma\delta$S and $\rho\delta\sigma$S strategies tend to degrade the performance of BANDBX by a factor of more than two on an average.

- Among the critical path strategies, SCP, $\sigma\delta\rho$CP, $\delta\sigma\rho$CP, $\delta\rho\sigma$CP and RPCP lead to an improvement in the performance of BANDBX, whereas $\sigma\rho\delta$CP, $\rho\sigma\delta$CP, $\rho\delta\sigma$CP and FPCP deteriorate the performance. So, here also, δ,ρ ordering gives an improvement while ρ,δ leads to a degradation.

- The strategies that use type information (σ, ρ or δ) of the variables in deciding their importance, can be divided into four classes based on their performance:

    1. $\sigma\delta\rho$S, $\delta\sigma\rho$S and $\delta\rho\sigma$S

    2. $\sigma\rho\delta$S, $\rho\sigma\delta$S and $\rho\delta\sigma$S

3. $\sigma\delta\rho$CP, $\delta\sigma\rho$CP and $\delta\rho\sigma$CP

4. $\sigma\rho\delta$CP, $\rho\sigma\delta$CP and $\rho\delta\sigma$CP

The interesting observation is that within each class, the performance deteriorates as the importance of the variables of type σ is reduced, *e.g.*, the performance of $\sigma\rho\delta$CP is better than that of $\rho\sigma\delta$CP, which in turn performs better than $\rho\delta\sigma$CP.

5.1.2. More Specific Observations

The observations in the previous subsection are based on the average performance shown in Figure 5-1. However, some of the strategies failed to find an optimal solution in certain cases which does not show up at all in the figure. To get a more detailed idea, let us look into Table 5-1. We can make the following observations:

- Among the simple ordering strategies, the strategies following δ,ρ order have always been able to find an optimal solution and have always improved the performance of BANDBX, except in the case of Logic Cost Optimization. Even in that case, the performance degradation is very little and the three δ,ρ order strategies certainly do much better than the three ρ,δ order strategies. So, overall $\sigma\delta\rho$S, $\delta\sigma\rho$S, and $\delta\rho\sigma$S strategies seem to be very effective and reliable.

- The ρ,δ order simple ordering strategies fail to find an optimal solution in many cases. In the cases where they do find an optimal solution, the performance of BANDBX is fairly consistently degraded, except that $\rho\delta\sigma$S has improved the performance in the case of Logic Time Optimization. Even in that case the improvement is not high enough to

be taken seriously. Overall, $\sigma\rho\delta$S, $\rho\sigma\delta$S and $\rho\delta\sigma$S strategies seem to be neither effective nor reliable.

- The SCP strategy has always been able to find an optimal solution and has always led to an improvement in the performance, except in the case of Logic Cost Optimization. Even in that case, the performance degradation is low enough to be ignored. Overall, the SCP strategy seems to be fairly effective and reliable.

- The $\sigma\delta\rho$CP, $\delta\sigma\rho$CP and $\delta\rho\sigma$CP strategies seem to perform acceptably on an average basis. However, they failed to find an optimal solution in a few cases and also deteriorated performance in a few cases. Overall, they don't seem to be reliable.

- The $\sigma\rho\delta$CP, $\rho\sigma\delta$CP and $\rho\delta\sigma$CP strategies have failed to find an optimal solution in many cases and have generally led to a degradation in the performance of BANDBX. They have improved the performance in a few cases which can be dismissed as exceptions. Overall, the strategies seem to be neither effective nor reliable.

- The FPCP strategy has failed to find an optimal solution in many cases and has generally led to a degradation in performance, except in the case of Criss.Cross Cost-Time tradeoff, where the performance gain is too little to be taken seriously. Overall, FPCP seems to be neither effective nor reliable.

- RPCP seems to be quite effective on an average basis. It does very well in all but two cases. It fails completely in the case of CPath Cost

Optimization and it leads to a significant deterioration in performance in the case of Logic Cost Optimization. It would seem that RPCP is very effective but not fully reliable. Particularly, it seems that it is very effective and reliable in solving the synthesis model for time (performance) optimization.

- The trend that the performance of BANDBX deteriorates as the importance attached to the variables of type σ is reduced seems to be more closely followed in the cases where cost optimization is involved.

Overall, the strategies $\sigma\delta\rho S$, $\delta\sigma\rho S$, and $\delta\rho\sigma S$ seem to be most effective and reliable. SCP is also quite reliable, but it is not as effective in terms of the performance gain. It seems that $\sigma\delta\rho S$ would do best for cost optimization. However, when time optimization is involved, it seems that RPCP also can be used very effectively and reliably.

5.2. Discussion and Some Explanations

5.2.1. Simple Ordering Strategies

From the observations, it is quite conclusive that assigning a higher priority to δ variables than that assigned to ρ variables leads to very significant performance gains. Let us try to understand why. It does make sense after a little thought. δ variables specify how the values required by the various operator inputs are accessed, *e.g.*, $\delta_{a,b}$ indicates whether the value required by input $i_{a,b}$ is accessed directly from the output of the operator producing the value or from the stored

copy of the value. ρ variables specify the output value to storage element mapping, *e.g.*, $\rho_{e,a,c}$ indicates if storage element s_e is used to store output value $o_{a,c}$. So, δ variables dictate if an output value will have a stored copy (if there is no operator input accessing the stored copy of a particular output value, then there is no reason to keep a stored copy of the value), whereas ρ variables decide which storage element, if any, will be used to store the output value. It makes sense to decide first if a stored copy is required, and then decide the actual storage element to be used. In other words, giving higher importance to δ variables than that of ρ variables is an intuitively sound approach and will tend to cut down on backtracking in the branch-and-bound process.

As far as the effect of the importance attached to σ variables is concerned, the results are not conclusive. However, giving more importance to σ variables seems to have a positive effect on the performance of BANDBX, especially when cost optimization is involved. There are two possible explanations for this behaviour:

- σ variables represent operator usage decisions, as opposed to ρ variables which represent storage usage decisions. As the average cost of the operators is usually higher than that of the storage elements (certainly so in our synthesis examples), the decisions about operator usage have a larger impact on the cost of the implementation. If storage decisions are made first, the tendency will be to avoid using storage elements at all because that will lead to a cheaper partial implementation. However, this implies that more operators will have to be used when operator decisions are being made later (not much

sharing of operators will be possible due to unavailability of stored copies of the values). So, the overall cost of the implementation goes up. Instead, if we make operator decisions first, we will tend to use a minimum number of operators (maximum sharing) as that will lead to a cheaper partial implementation. Of course, later we will have to add more storage elements to the implementation in order to make sharing of operators possible, but since the average cost of the storage elements is less, the overall cost of the implementation will still be less. Hence, it seems that making operator decisions more important will tend to produce cheaper implementations early in the branch-and-bound exploration and thus will produce better bounds. Therefore, making σ variables more important should certainly be helpful, especially when cost optimization is involved.

- We have already concluded and justified that δ decisions should be considered before ρ decisions. Now, the question is how to make δ decisions, *i.e.*, how to decide the access mechanism for the operator inputs. Before we try to decide how to access a particular operator input, it might be helpful to decide what operator to use to do the particular operation. In general, the knowledge about operation to operator mapping should be useful in the process of δ decisions. Particularly, the knowledge about sharing of operators could be helpful in deciding if a particular operator input should be accessed from the stored copy of the value. So, it seems that σ, δ, ρ is the most intuitive order and will lead to minimum amount of backtracking.

5.2.2. Critical Path Strategies

The more sophisticated idea of critical path does show some promise as indicated by the consistency and reliability of the SCP strategy. There is more flexibility in terms of the choices available for implementation of the non-critical part. Critical path implementation, being crucial to the overall performance of the design, does not have that much flexibility. The fact that the decisions related to the more crucial part of the design are being made first does tend to reduce backtracking so as to improve the BANDBX performance. However, the data gathered for this thesis indicates the improvement may not be as much as one would intuitively expect.

The most interesting observation about the critical path strategies is the contrast between the performances of RPCP and FPCP. More than anything else, it has to do with the fact that in RPCP a pseudo δ,ρ ordering within the critical path is followed whereas in FPCP there is a pseudo ρ,δ ordering. Pseudo δ,ρ ordering means that the δ variables that dictate whether or not a stored copy of the output value related to a particular ρ variable is needed, are considered before the ρ variable. *E.g.*, in CPath synthesis example (Figure 4-8), RPCP considers how the inputs for x_3 should be accessed before it considers if and where $o_{2.1}$ should be stored; FPCP follows exactly the opposite order of decisions. As is obvious, the only reason why a stored copy of $o_{2.1}$ might be required is that x_3 wants to access it. It seems that RPCP is particularly effective in time optimization because

the decisions which directly affect the output availability time are made first (along the critical path. RPCP starts at the output and works its way up to the input).

5.2.3. Comparison

Overall, the simple ordering strategies seem to have done better than the critical path strategies in the examples considered. The idea of critical path being more intelligent, one would expect otherwise. The synthesis examples we have considered are not large and complex in the sense of having a real critical path. Logic and Criss.Cross do not have a critical path at all, and although CPath has a critical path, it does not have much of a non-critical part. It is plausible that these examples will not exhibit the real power of critical path strategies. Our conjecture is that the idea of critical path will prove more and more effective as we solve larger and more complex examples having distinguishable critical paths and non-critical parts. The absence of an automated synthesis model generator limited us to the examples considered. The equations for the examples were hand-written. Manual generation of the model for anything larger becomes infeasible.

5.3. Suggestions for Further Research

There are two major directions for further research:

1. Development of static guidance strategies.
2. Development of dynamic guidance strategies.

5.3.1. Static Guidance Strategies

The approach adopted in this thesis is static in the sense that the importance of the variables is fixed throughout the branch-and-bound process. We never evaluate the effect of the previous design decision before considering the next decision, *i.e.*, we do not look at the partially constructed implementation before deciding the next variable to branch upon. Along this approach, the following ideas can be investigated (some of the ideas can be investigated very easily if we have an automated synthesis model generator):

- Construct a few large and complex examples having real critical paths and solve them using the strategies considered in this thesis to resolve the conjecture that the critical path strategies will be more effective for larger examples.

- Construct a few examples where the average cost of the storage elements is more than that of the operators and solve them to investigate if assigning higher priority to σ variables is effective just because of the higher cost of the operators.

- Order the three types of variables based on the number of variables in each type group. The smallest group gets the highest priority. Investigate the strategy particularly for time optimization. Some of the results in our examples give a very weak indication that this might be effective.

- Investigate some combinations of the strategies considered in this thesis,

e.g., in a simple ordering strategy, divide each category of variables into two groups, one belonging to the critical path and the other to the non-critical part, and thus have six priority levels instead of just three.

- Investigate critical path strategies that also order variables within the non-critical part.

5.3.2. Dynamic Guidance Strategies

As opposed to the static approach, this approach can change the importance of the variables at an intermediate stage. The approach will actually simulate the generalized design strategy discussed in Section 3.2.1. This approach may evaluate the effect of the design decision just made and redefine the importance of the rest of the variables depending on the partial implementation constructed so far. In its extreme form, it will look at the partial implementation at each branch-variable selection step and then decide the next variable to branch upon. Eventually it takes the form of an expert system having design knowledge built into it, which guides the branch-and-bound solution process for the synthesis model. This approach is worth investigating. The knowledge about the effectiveness of the various static strategies can be used in the dynamic approach decision process.

Appendix A

BANDBX Statistics

Table A-1: BANDBX Statistics for Criss.Cross Cost Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	80	93	1617
$\rho\delta\sigma$S	378	422	10793
$\delta\rho\sigma$S	36	92	2353
$\sigma\delta\rho$S	59	62	1247
$\sigma\rho\delta$S	134	240	5593
$\rho\sigma\delta$S	194	239	5884
$\delta\sigma\rho$S	36	86	2250
SCP	50	62	1256
$\rho\delta\sigma$CP	136	166	4484
$\delta\rho\sigma$CP	38	123	3843
$\sigma\delta\rho$CP	49	61	1254
$\sigma\rho\delta$CP	45	122	3032
$\rho\sigma\delta$CP	87	114	2850
$\delta\sigma\rho$CP	38	108	3399
FPCP	52	138	3339
RPCP	31	52	1199

Table A-2: BANDBX Statistics for Criss.Cross Time Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	45	93	1956
$\rho\delta\sigma$S	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$S	24	24	452
$\sigma\delta\rho$S	34	39	707
$\sigma\rho\delta$S	Failed, Numerical Inaccuracy.		
$\rho\sigma\delta$S	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$S	23	23	451
SCP	57	74	1345
$\rho\delta\sigma$CP	69	104	1907
$\delta\rho\sigma$CP	32	68	1265
$\sigma\delta\rho$CP	32	68	1265
$\sigma\rho\delta$CP	69	104	1907
$\rho\sigma\delta$CP	69	104	1907
$\delta\sigma\rho$CP	32	68	1265
FPCP	69	104	1907
RPCP	32	68	1265

Table A-3: BANDBX Statistics for Criss.Cross Cost-Time Tradeoff

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	152	230	5862
$\rho\delta\sigma$S	64	668	18233
$\delta\rho\sigma$S	222	226	5936
$\sigma\delta\rho$S	83	107	2855
$\sigma\rho\delta$S	302	320	8111
$\rho\sigma\delta$S	354	395	10193
$\delta\sigma\rho$S	181	185	5000
SCP	111	147	4096
$\rho\delta\sigma$CP	49	223	5861
$\delta\rho\sigma$CP	142	184	4900
$\sigma\delta\rho$CP	103	136	3797
$\sigma\rho\delta$CP	208	236	5816
$\rho\sigma\delta$CP	168	180	4621
$\delta\sigma\rho$CP	133	161	4375
FPCP	195	221	5527
RPCP	126	136	3625

Table A-4: BANDBX Statistics for Logic Cost Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	2774	3144	93118
$\rho\delta\sigma$S	7390	7661	216803
$\delta\rho\sigma$S	2048	3650	114639
$\sigma\delta\rho$S	2019	3628	114588
$\sigma\rho\delta$S	6945	7216	203556
$\rho\sigma\delta$S	7213	7484	213178
$\delta\sigma\rho$S	2051	3680	115971
SCP	3101	3423	98239
$\rho\delta\sigma$CP	4616	5632	160979
$\delta\rho\sigma$CP	2899	4190	124213
$\sigma\delta\rho$CP	2722	4000	117480
$\sigma\rho\delta$CP	4459	5610	160745
$\rho\sigma\delta$CP	4430	5658	161726
$\delta\sigma\rho$CP	2767	4042	118630
FPCP	Failed, Numerical Inaccuracy.		
RPCP	3699	4953	145538

Table A-5: BANDBX Statistics for Logic Time Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	2787	2787	74622
$\rho\delta\sigma$S	490	2475	65128
$\delta\rho\sigma$S	919	919	27185
$\sigma\delta\rho$S	569	569	14927
$\sigma\rho\delta$S	Failed. Storage Limit Exceeded.		
$\rho\sigma\delta$S	Failed. Storage Limit Exceeded.		
$\delta\sigma\rho$S	632	632	18741
SCP	1607	1607	44126
$\rho\delta\sigma$CP	Failed. Numerical Inaccuracy.		
$\delta\rho\sigma$CP	Failed. Numerical Inaccuracy.		
$\sigma\delta\rho$CP	Failed. Numerical Inaccuracy.		
$\sigma\rho\delta$CP	Failed. Numerical Inaccuracy.		
$\rho\sigma\delta$CP	Failed. Numerical Inaccuracy.		
$\delta\sigma\rho$CP	1033	1033	29360
FPCP	Failed. Numerical Inaccuracy.		
RPCP	142	142	2839

Table A-6: BANDBX Statistics for CPath Cost Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	805	1076	34402
$\rho\delta\sigma$S	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$S	130	311	10077
$\sigma\delta\rho$S	135	312	10110
$\sigma\rho\delta$S	Failed, Numerical Inaccuracy.		
$\rho\sigma\delta$S	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$S	132	291	9607
SCP	586	904	31473
$\rho\delta\sigma$CP	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$CP	264	399	13214
$\sigma\delta\rho$CP	261	369	12403
$\sigma\rho\delta$CP	1098	1135	41294
$\rho\sigma\delta$CP	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$CP	253	352	12004
FPCP	368	1338	46232
RPCP	Failed, Numerical Inaccuracy.		

Table A-7: BANDBX Statistics for CPath Time Optimization

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	315	2252	68276
$\rho\delta\sigma$S	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$S	56	60	1862
$\sigma\delta\rho$S	41	192	6268
$\sigma\rho\delta$S	Failed, Numerical Inaccuracy.		
$\rho\sigma\delta$S	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$S	88	93	2552
SCP	217	754	22921
$\rho\delta\sigma$CP	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$CP	118	123	3334
$\sigma\delta\rho$CP	34	182	6073
$\sigma\rho\delta$CP	Failed, Numerical Inaccuracy.		
$\rho\sigma\delta$CP	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$CP	112	117	3028
FPCP	Failed, Numerical Inaccuracy.		
RPCP	170	257	8177

Table A-8: BANDBX Statistics for CPath Cost-Time Tradeoff

Strategy	SPOPT#	TOTSP#	TSIMP#
NG	1305	3475	106655
$\rho\delta\sigma$S	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$S	1074	1550	45503
$\sigma\delta\rho$S	767	1241	38358
$\sigma\rho\delta$S	Failed, Numerical Inaccuracy.		
$\rho\sigma\delta$S	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$S	1015	1440	44686
SCP	856	1778	57231
$\rho\delta\sigma$CP	Failed, Numerical Inaccuracy.		
$\delta\rho\sigma$CP	Failed, Numerical Inaccuracy.		
$\sigma\delta\rho$CP	Failed, Numerical Inaccuracy.		
$\sigma\rho\delta$CP	52	2222	77060
$\rho\sigma\delta$CP	Failed, Numerical Inaccuracy.		
$\delta\sigma\rho$CP	Failed, Numerical Inaccuracy.		
FPCP	Failed, Numerical Inaccuracy.		
RPCP	780	1796	58662

References

[Hafer 81] Hafer, L.
Automated Data-Memory Synthesis : A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic.
PhD thesis, Dept. of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pa., May, 1981.
Also available from the Design Research Center, Carnegie-Mellon University, as Technical Report DRC-02-05-81.

[Johnson 85] Johnson, E., Kostreva, M., Suhl, U.
Solving 0-1 Integer Programming Problems Arising from Large Scale Planning Models.
Operations Research 33(4):803-819, July-August, 1985.

[Land 79] Land, A., Powell, S.
Computer Codes for Problems of Integer Programming.
Annals of Discrete Mathematics 5:221-269, 1979.

[Martin 78] Martin, C.
BANDBX: An Enumeration Code for Pure and Mixed Zero-One Programming Problems
Industrial and Systems Engineering Dept., Ohio State University, 1978.

[Mitra 73] Mitra, G.
Investigation of Some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs.
Mathematical Programming 4(2):155-170, April, 1973.

[Thomas 83] Thomas, D., Hitchcock, C., Kowalski, T., Rajan, J., Walker, R.
Automatic Data Path Synthesis.
Computer 16(12):59-70, December, 1983.

[Thomas 86] Thomas, D.
Automatic Data Path Synthesis.
Advances in CAD for VLSI. Volume 6.*Design Methodologies.*
North-Holland, Amsterdam, 1986, Chapter 13.