System-Level Synthesis Techniques With Emphasis On Partitioning And Design Planning

Kayhan Küçükçakar

CENG Technical Report: 91-27

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
(Electrical Engineering)

(Copyright October 1991)

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4476

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

Kayhan Kucukcakar

under the direction of his........ Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

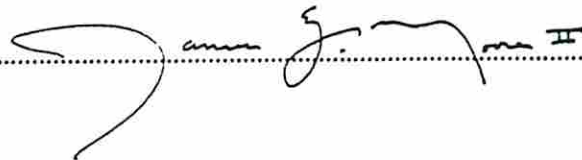
DOCTOR OF PHILOSOPHY

Dean of Graduate Studies

Date October 14, 1991

DISSERTATION COMMITTEE

Chairperson



Dedication

To my family

my parents, my sister, and my wife.

Acknowledgements

I would like to thank my advisor, Alice Parker for her encouragement, guidance, support and friendship. She was always there to listen to and to help me with research and personal problems. She took care of everything she could so that I only needed to worry about my research. Without her, this thesis would not exist. Her good sense of humor made my years at USC go by quickly. It is my privilege to have known her and have worked with her.

I would like to thank Melvin Breuer for introducing me to Design Automation and being on my thesis committee. I would also like to thank James Moore for being on my thesis committee, and Sarma Sastry and Gerard Medioni for being on my guidance committee. Their comments were most valuable.

I thank my parents and my sister for their continuous love, support and sacrifice. It has not been easy to be far away from them during my years here. I gratefully dedicate this thesis to them.

I thank my wife, Ela for her love, support and understanding. Her patience made it possible for me to get my Ph.D. this soon. I also dedicate this thesis to her.

I thank my colleagues Mitch Mlinar, Shiv Prakash, and Charles Njinda for their friendship. They made the long hours I spent at USC bearable. I also thank my other colleagues in the DA group; Rajiv Jain, Jagannath Raghavendran, Jorge Seidel, Jen-Pin Weng, and Pravil Gupta.

There are also friends within the DA group, in our department and outside who helped, supported, shared, and cared. Their enhancement to my life during these years is remembered.

I would like to thank Türk Eğitim Vakfı (Turkish Education Foundation) for their scholarship during the first two years of my studies in the U.S.

Finally, I gratefully acknowledge the financial support from the Department of Air Force, the Department of Army, and the Department of Navy under contract N00039-87-C-0194 and Defense Advanced Research Projects Agency under contract N00014-87-K-0861 (monitored by the Office of Naval Research) and under contract JFBI90092 (monitored by the Federal Bureau of Investigation). The views and conclusions contained in this thesis are of those of the author and should not be interpreted as necessarily representing the official policies. either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Contents

Dedication ii

Acknowledgements iii

List Of Figures xi

List Of Tables xiv

Abstract xv

1 Introduction **1**
- 1.1 Background 1
- 1.2 Behavioral Synthesis 3
- 1.3 The Conventional Synthesis Process 3
  - 1.3.1 The ADAM Synthesis System 8
- 1.4 Motivation 10
- 1.5 Problem Approach 12
- 1.6 Examples 15
- 1.7 The library 16
- 1.8 Thesis Organization 19

2 Related Research **21**
- 2.1 Behavioral Synthesis 21
  - 2.1.1 Behavior Specification 21
  - 2.1.2 Data Representation 23
  - 2.1.3 Domain-dependent Data Path Synthesis 23
    - 2.1.3.1 Digital Signal Processing 23
    - 2.1.3.2 Microprocessor Design 24
    - 2.1.3.3 Control Dominated Designs 24
  - 2.1.4 General Purpose Data Path Synthesis 25
    - 2.1.4.1 Mathematical Programming 25
    - 2.1.4.2 Scheduling 26

2.1.4.3 Allocation, Module Assignment, Interconnect Allocation . . . 26
2.2 Prediction Methods . . . 28
2.3 Partitioning . . . 31
2.3.1 Partitioning of Graphs . . . 31
2.3.2 Partitioning in Microprocessor Design . . . 31
2.3.3 Partitioning of Logic Circuits . . . 32
2.3.4 Partitioning of RT-Level Circuits . . . 32
2.3.5 Partitioning of Behavioral Specifications . . . 33

3 Data Path Synthesis **35**
3.1 Introduction . . . 35
3.2 Overview of MABAL . . . 36
3.2.1 Problem Statement . . . 36
3.2.2 Features of MABAL . . . 39
3.2.3 Assumptions . . . 42
3.2.4 Inputs . . . 42
3.2.5 Outputs . . . 43
3.3 Theoretical Basis for Tradeoffs . . . 44
3.3.1 Multiplexer versus bus driver tradeoff . . . 44
3.3.2 Resource sharing versus resource duplication tradeoff . . . 47
3.3.3 An upper bound on the interconnect area . . . 48
3.4 Overview of MABAL Operation . . . 51
3.4.1 How MABAL performs synthesis . . . 51
3.4.2 MABAL Algorithm . . . 52
3.4.2.1 Example . . . 55
3.5 Optimality Considerations . . . 59
3.6 Complexity Considerations . . . 60
3.6.1 Experimental Complexity Analysis . . . 61
3.7 Experimental Results . . . 61
3.7.1 AR Filter . . . 63
3.7.2 FIR Filter . . . 64
3.7.3 Differential Equation Solver . . . 65
3.7.4 Importing Other Synthesis Methods into MABAL . . . 66
3.8 Limitations of MABAL . . . 67
3.9 A Data Path Tradeoff Study . . . 69
3.9.1 Results and Discussion . . . 71
3.10 Summary . . . 80

4 Behavioral Area-Delay Predictions **82**
4.1 Introduction . . . 82

4.12 Feasibility Analysis 123
4.13 Synthesizing the predicted designs 123
4.14 Advantages of Behavioral Area-Delay Predictions (BAD) 124
4.15 Limitations of BAD 125
4.16 Summary 127

5 Validation of Behavioral Area-Delay Predictions 128
5.1 AR Lattice Filter 130
5.1.1 PLA Estimations 132
5.1.2 Clock Cycle Time Estimation 133
5.2 Elliptic Wave Filter 143
5.3 FIR Filter 143
5.4 Feasibility Analysis Example 149
5.5 Summary 150

6 Behavioral Partitioning of Digital System Specifications 151
6.1 Introduction 151
6.2 Overview of CHOP 153
6.3 Assumptions 155
6.4 Inputs 155
6.4.1 Clocking Scheme 156
6.5 Partitioning Approach 157
6.5.1 An example for the partitioning model 159
6.5.1.1 Data Transfer Tasks 161
6.5.1.2 The implementation 163
6.5.2 Predicting partition implementations 164
6.5.2.1 Further elimination of predicted designs 166
6.5.3 Constructive predictions for the global design 170
6.5.3.1 Selecting an Initiation Interval 172
6.5.4 System Integration Overhead 174
6.5.4.1 Preprocessing for Task Scheduling 176
6.5.4.2 Task Scheduling 177
6.5.4.3 System Delay Predictions 178
6.5.4.4 Predictions for Data Transfer Modules 179
6.5.4.5 Pin Multiplexing Prediction 181
6.5.4.6 Wiring Estimation 182
6.5.4.7 Clock Cycle Time Estimation 183
6.6 Partitioning Modification 183
6.7 Feasibility Analysis 185
6.8 Synthesizing the partitioned designs 186
6.9 Complexity Analysis 186

4.1.1 Overview 83
4.2 A Unified Representation of Prediction Results 85
4.3 Assumptions 89
4.3.1 Operation to operator mapping 89
4.3.2 Clocking and Control 89
4.3.3 Non-deterministic Delays 90
4.3.4 Loops 90
4.3.5 Resynchronization 90
4.3.6 Timing Constraints 91
4.4 The Prediction Mechanism 91
4.5 Operator Estimation 92
4.5.1 Pipelined Design Style 94
4.5.1.1 Theoretical Bounds 94
4.5.1.2 Operator Allocation Estimation 95
4.5.2 Non-Pipelined Design Style 95
4.5.2.1 Theoretical Bounds 96
4.5.2.2 Operator Allocation Estimation 96
4.5.3 Operator Timing Characteristics 97
4.5.3.1 Non-pipelined Designs 98
4.5.3.2 Pipelined Designs 99
4.6 Register Estimation 102
4.6.1 Notation 102
4.6.2 Determination of graph width 103
4.6.3 Register Area Estimation 104
4.6.4 Register Delay Estimation 105
4.7 Interconnect Estimation 105
4.7.1 Upper Bounds on the Multiplexer Area 107
4.7.2 Multiplexer Area Estimation 107
4.7.3 Multiplexer Delay Estimation 108
4.8 Control Estimation 110
4.8.1 Estimating abstract PLA characteristics 110
4.8.2 PLA Area Estimation 112
4.8.3 PLA Delay Estimation 113
4.9 Wiring Estimation 113
4.9.1 Wiring Area Estimation 113
4.9.2 Wiring Delay Estimation 115
4.9.2.1 Delay Estimations for a single wire 116
4.9.2.2 Estimations of delays introduced into the clock cycle 117
4.10 The effects of the clock cycle time on design characteristics 120
4.11 Run-time complexity of the predictions 121

6.9.1 Exhaustive Search Technique 1 187
6.9.2 Exhaustive Search Technique 2 188
6.9.3 Heuristic Search Technique 188
6.10 Considering design costs 189
6.10.1 Background 189
6.10.2 How to use costs in synthesis 189
6.11 Extensions to use off-the-shelf data path parts 190
6.12 Limitations of CHOP 191
6.13 Summary 194

7 Partitioning Experiments and Results **195**
7.1 Validation of partitioning predictions 195
7.1.1 AR Filter 197
7.1.2 FIR Filter 198
7.2 Partitioning evaluations 199
7.2.1 The AR filter 201
7.2.2 The FIR filter 203
7.2.3 Multiple partitions per chip 205
7.3 Comparison of search methods 206
7.3.1 Constraining the search performed by CHOP 207
7.4 Non-uniform partitioning in CHOP 209
7.5 A simple automatic 2-way partitioning 210
7.6 Discussion of Behavioral Partitioning Issues 212
7.6.1 Parallelism 212
7.6.2 Partition Granularity 212
7.6.3 Non-linearity of moves in the design space 213
7.6.4 Utilization 213
7.6.5 Off-chip Communication 214
7.6.6 Clock Cycle Times 215

8 Conclusion and Future Research **217**
8.1 Contributions 217
8.2 Future Research 218

Reference List **220**

Appendix A
A Sample MABAL Output 232

Appendix B
A Sample BAD Output 240

Appendix C
A Sample CHOP Output 242

List Of Figures

1.1 A behavioral description as a dataflow graph 4
1.2 A schedule for the behavior in Figure 1.1 5
1.3 An RTL implementation of the behavior in Figure 1.1 6
1.4 Synthesis subtasks in a generic synthesis process 7
1.5 The current ADAM Synthesis System 9
1.6 System-level Synthesis Process 13
1.7 The Design Space Search in System-level Synthesis 15
1.8 The FIR Filter Dataflow Graph 16
1.9 The AR Filter Dataflow Graph 17
1.10 The Elliptic Wave Filter Dataflow Graph 18

3.1 A possible design MABAL might produce. 40
3.2 Two possible styles for the data steering logic 45
3.3 An existing partial design 56
3.4 Adding two multiplexer inputs 56
3.5 Converting to a single bus 56
3.6 Adding a bus driver 57
3.7 Adding a multiplexer 58
3.8 Adding a functional module and a multiplexer 58
3.9 Adding a functional module and a bus driver 58
3.10 Run-time Complexity Analysis for MABAL 62
3.11 The dataflow graph used in the tradeoff experiment 70
3.12 The schedule produced by MAHA for the dataflow graph shown in Figure 3.11 71
3.13 Selected designs generated using library 1 74
3.14 Selected designs generated using library 2 75
3.15 Selected designs generated using library 3 76
3.16 Proposed model for functional area versus steering logic area . . . 77
3.17 A minimal functional resource design 78
3.18 A non-minimal functional resource design 79
3.19 The results of the experiments with PLEST 80

4.1 The PERT model used in predictions 86
4.2 The probability of feasibility as the area under a probability density curve 88
4.3 The prediction generation in BAD 92
4.4 The heuristic for pipeline length estimation 101
4.5 Possible graph shapes considered 103
4.6 Graph width estimation technique 104
4.7 Register Estimation Algorithm 106
4.8 Operator chaining in a schedule 109
4.9 Multiplexer estimation algorithm 111
4.10 The interpolation algorithm used to estimate PLEST results . . . 116

5.1 Operator area for the non-pipelined AR Filter 131
5.2 Operator + register area for the non-pipelined AR Filter 132
5.3 The RTL area for the non-pipelined AR Filter 133
5.4 RTL + PLA area for the non-pipelined AR Filter 134
5.5 The number of 2-points nets for the non-pipelined AR Filter . . . 135
5.6 Area break-down for non-pipelined the AR Filter 135
5.7 Operator area for the pipelined AR Filter 136
5.8 Operator + register area for the pipelined AR Filter 136
5.9 RTL area for the pipelined AR Filter 137
5.10 RTL + PLA area for the pipelined AR Filter 137
5.11 Area break-down for the pipelined AR Filter 138
5.12 The number of 2-points nets for the pipelined AR Filter 138
5.13 The number of stages in the pipeline for the AR Filter 139
5.14 Comparisons of layout area for two methods (Non-pipelined) . . . 139
5.15 Comparisons of layout area for two methods (pipelined) 140
5.16 Area break-down for the non-pipelined Elliptic Filter 144
5.17 Area break-down for the non-pipelined Elliptic Filter 144
5.18 Area break-down for the pipelined Elliptic Filter 145
5.19 The number of 2-points nets for the non-pipelined Elliptic Filter . 145
5.20 The number of 2-points nets for the pipelined Elliptic Filter . . . 146
5.21 The number of stages in the pipeline for the Elliptic Filter 146
5.22 Area break-down for the non-pipelined FIR Filter 147
5.23 Area break-down for the pipelined FIR Filter 147
5.24 The number of 2-point nets for the pipelined FIR Filter 148
5.25 The number of 2-point nets for the non-pipelined FIR Filter . . . 148
5.26 All designs predicted for the AR Filter 149
5.27 Feasible Predicted Designs for the AR Filter 150

6.1 The block diagram of the USC (Unified System Construction) Project 154

6.2 The overall operation of CHOP 158
6.3 An example partitioning 160
6.4 The task graph for the example partitioning 162
6.5 The architectural building blocks 164
6.6 The final implementation of the example partitioning 165
6.7 Selection of partition implementations 170
6.8 The exhaustive search techniques for finding the global design . . 173
6.9 The simplified outline for the iterative heuristic 175
6.10 The Modes of Operation for Data Transfer Tasks 179
6.11 The partitioning with dependency loop 191
6.12 The partitioning without dependency loop 192

7.1 A 2-way partitioning of the AR Filter 197
7.2 A 2-way partitioning of the FIR Filter 198
7.3 A 2-way partitioning of the AR Filter 201
7.4 A 3-way partitioning of the AR Filter 202
7.5 A 3-way, 2-chip partitioning of the AR Filter 202
7.6 A 2-way partitioning of the FIR Filter 204
7.7 A 3-way partitioning of the FIR Filter 204
7.8 A 4-way, 3-chip partitioning of the FIR Filter 204
7.9 A simple automatic 2-way partitioning technique. 211

List Of Tables

1.1 The library used in the experiments 19

3.1 Pipelined AR Filter Results 63
3.2 Non-pipelined AR Filter Results 64
3.3 Comparisons for Pipelined FIR Filter 65
3.4 Comparisons for Differential Equation Solver 66
3.5 Comparisons of results from [PK89a] and MABAL 67
3.6 Libraries used for experiments in Section 3.9 69
3.7 Statistical Data on Bus (bus driver) Usage 73

5.1 Estimated and Synthesized PLA Characteristics for the AR Filter 140
5.2 Estimated clock cycle times for non-pipelined implementations of the AR Filter 141
5.3 Estimated clock cycle times for pipelined implementations of the AR Filter 142

7.1 Estimated and Synthesized Design Characteristics of the 2-way partitioning of the AR Filter shown in Figure 7.1 198
7.2 Estimated and Synthesized Design Characteristics of the 2-way partitioning of the FIR Filter shown in Figure 7.2 199
7.3 MOSIS standard chip packages 200
7.4 AR Filter Partitioning Evaluations 203
7.5 FIR Filter Partitioning Evaluations 205
7.6 Statistical data on AR filter evaluation results in Table 7.4 207
7.7 Statistical data on FIR filter evaluation results in Table 7.5 208
7.8 Statistical data on a constrained run using the AR Filter 209
7.9 Different cost 2-chip implementations of the AR Filter using MOSIS chip packages 210
7.10 Evaluations for varying clock cycle times for partitioning shown in 7.3 216

Abstract

Behavioral synthesis is currently performed by serial execution of several synthesis tasks which, if performed separately, may have conflicting goals. Since even the solutions to the individual simpler synthesis tasks are exponential in complexity, the solution methods are generally heuristic, do not have a global understanding of the overall problem, and focus on very specific optimizations. Producing good designs within such an environment generally involves a fair amount of design iteration.

In this thesis, system-level synthesis techniques are presented to partition the behavioral specifications onto multiple chips, to guide the synthesis process, to reduce the amount of iterations in the design, and to search for as many alternative implementations as possible. These system-level synthesis techniques utilize comprehensive behavioral area-delay predictions which span from design style selection to layout and attempt to meet hard area, throughput, delay, pin-count constraints.

The behavioral partitioning methodology presented is aimed at predicting area-delay characteristics of synthesized multi-chip designs from behavioral descriptions. The prediction mechanism consists of several prediction techniques, each corresponding to a synthesis activity (e.g. module selection, operator, register and multiplexer allocation). Behavioral partitioning and prediction techniques have been tested using examples from the literature.

A data path synthesis program (MABAL) which was written to enable the ADAM Synthesis System to produce RTL designs is also included here.

Chapter 1

Introduction

1.1 Background

With the decreasing cost of VLSI chips, Application Specific Integrated Circuits (ASICs) have emerged as one of the fastest growing markets. ICs are being used in more diverse applications than ever before. As the life cycles of products are rapidly decreasing and the market pressure is being reflected in making the IC design process quicker, ASIC designers find themselves with several needs. These include but are not limited to

- reducing the design turnaround time,
- exploring alternative design implementations,
- reducing high costs of non-recurring engineering, and
- reducing the *design cost per product* as the production volume per design is decreasing.

The design automation industry has responded to these needs by introducing tools for layout generation, placement and routing, design capture, simulation, module generation, automatic test pattern generation and recently logic synthesis. As the market pressure constantly forces the development of higher-level design tools, several behavioral-level synthesis techniques which can generate

fairly acceptable designs are being proposed. The other benefits of widening the formalism to cover the whole design process include easier verification of the design before the fabrication phase, easier test generation to detect manufacturing faults, and self documentation during the design process.

The behavioral synthesis as well as the lower-level design automation tools both suffer from the fact that the techniques used generally do not have a *global* understanding of the overall design process because of the high complexity of the synthesis problem. Most synthesis techniques perform optimizations for very specific problems without having any notion of what the effects of such optimizations will be on the global design characteristics. The effects of such techniques become much more apparent at the architectural (system) level where the impact of individual design decisions on the final design characteristics can be much greater than lower-level decisions.

When individual design tools are focused on specific optimizations, an iterative overall design process is needed in order to be able to produce good quality designs. This iteration is manually controlled and generally performed on a trial basis. Clearly, moving to higher levels of abstraction speeds up the one-pass design process. But, on the other hand, when the synthesis starts at the behavioral or system level, the number of alternative designs which can be produced increases drastically. As a result, the total time for the iterated design process is not necessarily reduced, even though the one-pass design time is reduced.

Although higher-level design tools enable us to search quickly for alternative implementations of designs, in order to be able to design ASIC systems effectively with acceptable design time and quality using design automation tools, more emphasis on the system-level issues is highly needed. With a better understanding of *system-level issues*, the design process can be controlled effectively and the *design quality* can be increased while simultaneously reducing the *design time.*

1.2 Behavioral Synthesis

Behavioral synthesis is a process which produces a register-transfer design from a purely abstract behavioral description. The behavioral description only specifies the intended functionality of the design and does not specify how the implementation is to be performed.

In very simple terms, behavioral synthesis maps the operations in a specification onto time steps and onto hardware while satisfying a set of constraints and optimizing a goal.

The behavioral description can be represented by dataflow and control flow graphs. An example dataflow graph with partial control constructs is shown in Figure 1.1. One of the mappings performed during the behavioral synthesis is the mapping of the operations of the dataflow graph onto time steps as shown in Figure 1.2. This mapping, which is called scheduling, orders and discretizes the execution of operations. Another mapping performed is a mapping from operations onto actual hardware modules. This mapping is generally associated with the creation of the necessary interconnect. At this stage, a Register Transfer Level (RTL) design is produced as shown in Figure 1.3.

Most behavioral synthesis approaches stop at this stage and pass the RTL design to the lower level design automation tools for further optimization and the layout generation.

1.3 The Conventional Synthesis Process

The ultimate behavioral synthesis tool would take the behavioral description, the design library, a set of constraints (area, timing, architecture), a goal and would directly produce a feasible design (a layout) with little or no intervention.

However, since the synthesis task is very complicated, it is currently broken into several subtasks which are generally executed in a sequential manner. Figure 1.4 shows a detailed view of a generic synthesis process. The exact ordering

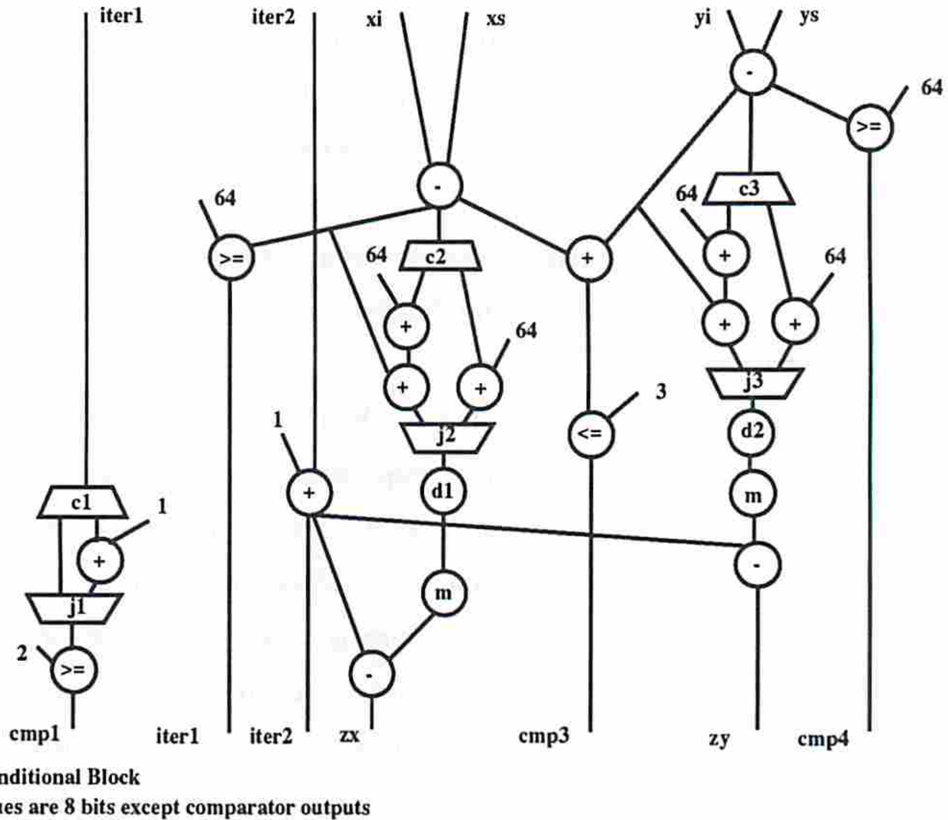


Figure 1.1: A behavioral description as a dataflow graph

of the synthesis tasks is not yet universally agreed on. Although new techniques reported in the literature attempt to simultaneously solve multiple subtasks, in practice, only a very limited number of these subtasks are handled simultaneously due to the complexity of the subtasks. The subtasks that the synthesis process is broken into generally have conflicting goals when they are performed independently. Therefore, an appreciable amount of iteration may be necessary to produce good quality results. The feedback arcs in Figure 1.4 show the possible ways of iterating. Currently, the iteration is manually controlled and performed on a trial basis. When synthesis subtasks are executed in sequence, there are always some assumptions made on the final characteristics of the design being

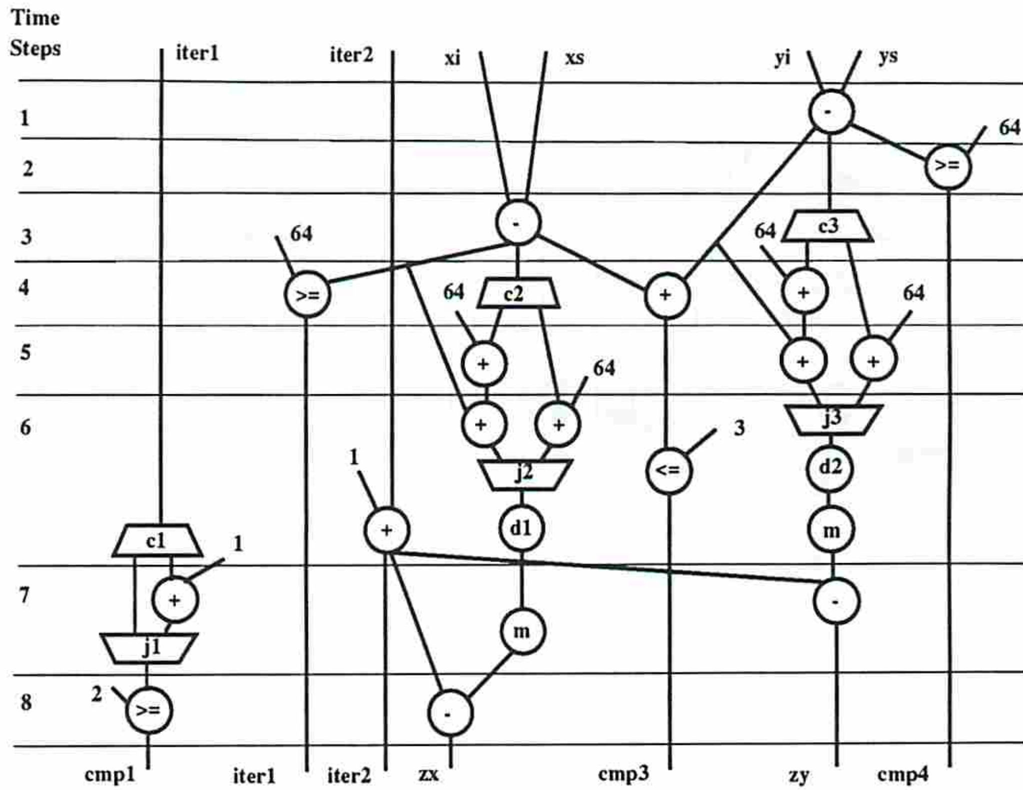


Figure 1.2: A schedule for the behavior in Figure 1.1

synthesized. The assumptions do not necessarily hold when the design is completely synthesized. For example, the operator area is assumed to be dominant factor in the final area of designs in many approaches. Therefore, each operator is generally shared to the fullest extent possible in the scheduling process. But, in a digital design, the buses and multiplexers required to interconnect functional units and registers may have a first-order effect on hardware cost (silicon area). Furthermore, the number and types of interconnections required are heavily dependent on the number of functional units and registers in the design, as well as the assignment of operations and values to these units. Thus, interconnect costs must be considered at the same time that other high-level design decisions are

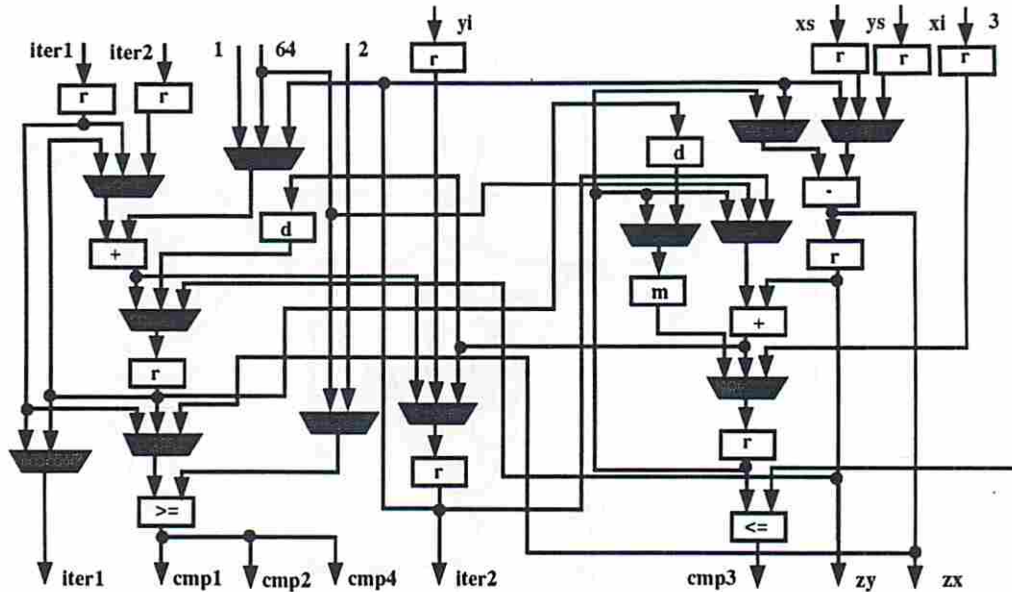


Figure 1.3: An RTL implementation of the behavior in Figure 1.1

being made in order to produce globally good designs while iterating less in the overall design process.

Unfortunately, the results of conventional behavioral synthesis are generally dependent on the way the behavior is specified. That is, the behavioral description is actually used for reasons other than only specifying the behavior, e.g. the abstraction level. By using high-level transformations, the abstraction in the synthesis process as well as some constraints on the behavior can be changed. In some schools of thought, high-level transformations are more or less associated with the behavior rather than the synthesis process [Mli91], while in others, they are considered as integral parts of the behavioral synthesis process [RCHP91]. Examples of such high-level transformations are

- **Operation composition** in which several operations are clustered into complex operations [Sno78, WT89, Mli91].

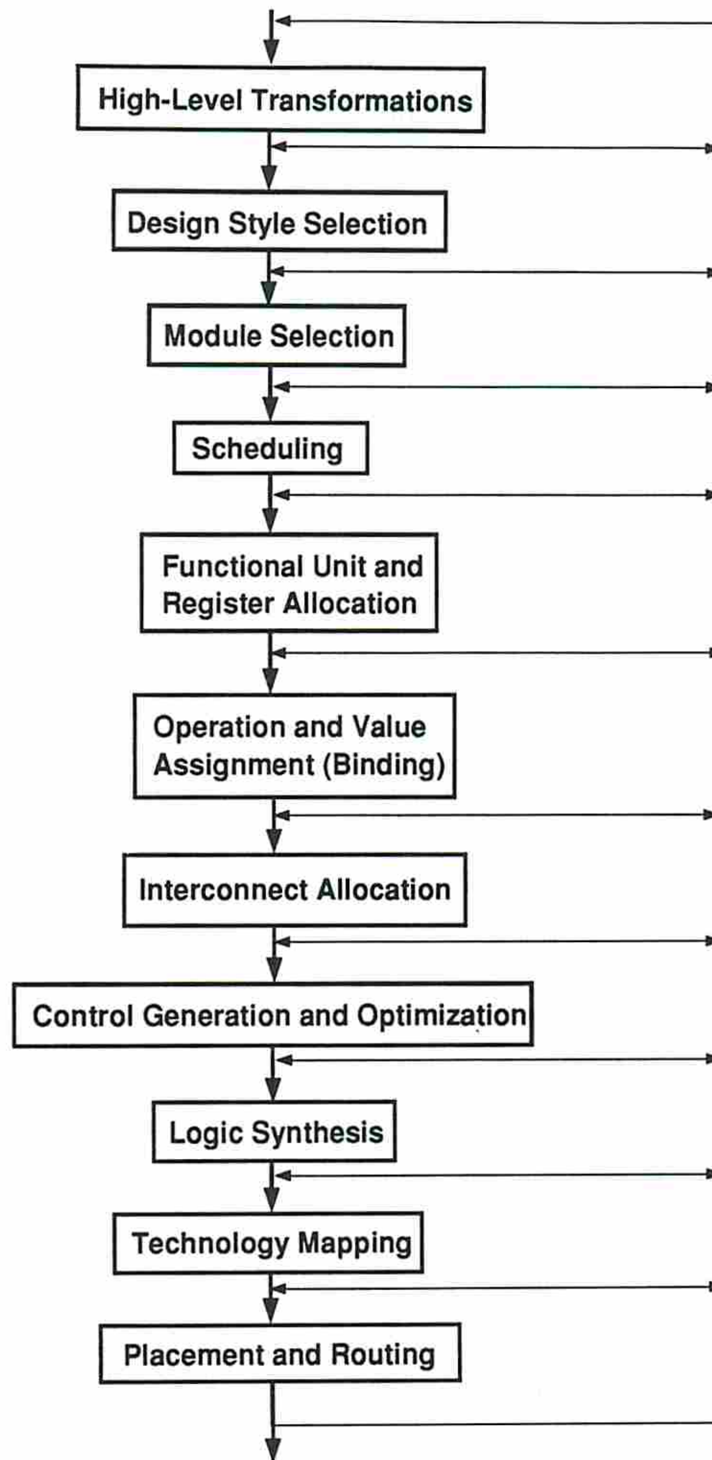


Figure 1.4: Synthesis subtasks in a generic synthesis process

- **Operation decomposition** in which complex operations are broken into simpler operations, e.g. a multiply can be broken into shift and add operations [Sno78, WT89, Mli91].

- **Bitwidth tradeoffs** in which operations with long bitwidths are broken into operations with shorter bitwidths, e.g. a 16-bit multiplication can be implemented by 4-bit multiplications and additions [Mli91].

- **Tree height reduction:** The behavior can be described in a way to introduce unnecessary data dependencies which can be optimized by well-known compiler optimization techniques such as tree height reduction using associative and/or distributive natures of the operations [Tri87, HC89].

1.3.1 The ADAM Synthesis System

Figure 1.5 shows the current ADAM Synthesis System. In the ADAM Synthesis System, the behavior is currently described in VHDL [VHD88]. A commercial parser from CLSI is used to parse the VHDL description and another program is used to extract the data from the CLSI database and to store it in the EVE database [Che90].

The design style (pipelined or non-pipelined) is decided manually, followed by module selection. SLIMOS is a program which performs module selection for the pipelined design style [JPP88]. Module selection is a process of deciding which operation in the behavioral description is to be implemented by which type of operator (e.g. additions may be performed using either ripple-carry or carry lookahead adders). There is no module selection program in the ADAM System for the non-pipelined design style, however, SLIMOS has also performed favorably for the non-pipelined design style.

The scheduling task in the ADAM System is performed by MAHA [PPM86] or Sehwa [PP88], non-pipelined and pipelined schedulers, respectively. While MAHA produces a non-overlapping (non-pipelined) schedule, Sehwa produces a

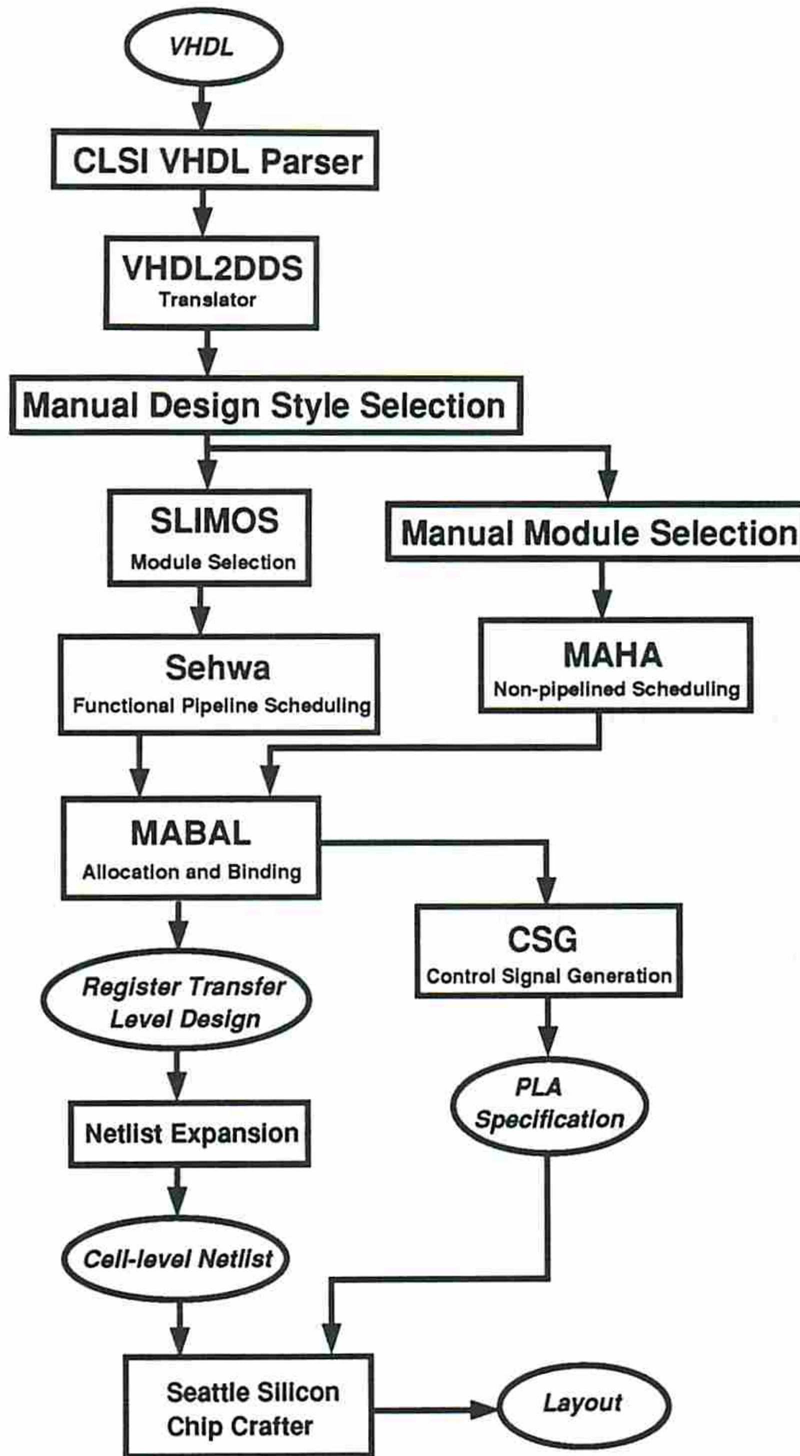


Figure 1.5: The current ADAM Synthesis System

schedule employing functional pipelining in which there are no physical stage boundaries.

After either type of scheduling is completed, MABAL [KP90e] performs the functional resource allocation, the register allocation, the assignments for operations and values, and the interconnect synthesis, thereby producing an RTL design.

The RTL design along with the behavior is then passed to CSG [Wen89] to produce the controller specification which can be optimized by general purpose control optimization techniques.

The RTL design and the controller specification are currently fed to the Seattle Silicon Chip Crafter System for placement and routing to produce the final layout.

1.4 Motivation

The synthesis process is so complicated that there is not yet a complete formal model to describe it. Therefore, the synthesis process is performed by sequentially executing tools which correspond to the subtasks shown in Figure 1.4. These tools generally solve very restricted instances of the problems which subtask names suggest. For example, rather than finding an answer to "*What is the best way to schedule the behavior so that the resulting design would have the feasible area and delay characteristics?* ", these tools answer questions like "*What is the schedule with the minimal number operators of each type, given the number of time steps in the schedule?* ". Clearly, to solve the first problem several instances of the second problem need to be solved. Even though these subtasks address very simplistic optimizations, they are known to be exponential in complexity.

The synthesis process can be viewed as a huge design decision tree, where leaves represent alternative designs[1.1]. Then, any form of synthesis method can

[1.1] These leaves also constitute the so-called discrete design space.

be represented as a search on this decision tree. Synthesizing the optimal design corresponds to a complete search of this decision tree.

While the search is being performed on the decision tree some decision paths taken are detected to lead to infeasible designs. In such cases, some of the decisions should be undone and a different choice of decisions should be followed. Unfortunately, any such detection is generally not available at an early step in the design process. This is mainly due to the fact that there may not be enough information about the design characteristics unless a complete decision path is taken. This is further complicated by the fact that subtasks do have conflicting goals. That is, we can not simply perform independent optimizations (even optimizations with optimal results) for each subtask in sequence and expect that the final design would be well optimized. As a result, to synthesize a good design an excessive amount of iteration may be required and this iteration is not known to have any regular patterns.

The complexity of the synthesis task forces us to limit the search to very limited parts of the discrete design space. But, there is no known measure for absolute or relative *goodness* of a design except the final design characteristics. This makes it very hard to prune the search at an early stage.

Early behavioral synthesis research had the objective of producing single-chip VLSI implementations from behavioral specifications. In time, the sizes of implementable VLSI chips have increased drastically, but the average size of designs has also increased as well, keeping many implementations still multi-chip. Almost all of the behavioral synthesis research is based on single-chip design assumptions. With the decreasing feature sizes, functional delays have become comparable to or less than off-chip delays. The assumption that designs can be synthesized as single-chip designs and can then be partitioned onto multiple chips without too much penalty on timing does not hold anymore. After a design is synthesized as a single-chip design, it may not be possible to partition the design onto multiple chips while satisfying the constraints. Even if it may be possible

to partition some designs after synthesis while satisfying the constraints, it is likely that such multi-chip implementations would be inferior since the synthesis tools have performed optimizations assuming a single-chip implementation as the target design. Therefore, either behavioral synthesis techniques should be changed to handle multi-chip design synthesis or the partitioning has to be performed prior to the behavioral synthesis while taking into account multi-chip and system-level design issues.

To synthesize good quality multi-chip designs, partitioning before behavioral synthesis has to take into account several system-level issues in addition to the behavioral synthesis problems. Balancing the data processing among the chips, balancing the data processing and data transfers, and the memory hierarchy design become important aspects of the design process which are considered at a higher level than behavioral synthesis.

Given the flexibility of the behavioral synthesis and the vast abstraction space, it is very hard to make higher and higher level decisions in the early stages of the design process where there is not enough information on how the design would actually be implemented.

1.5 Problem Approach

When there is a huge design space to be searched, one well-known method to handle this type of complexity is to identify what constitutes the first-order effects and employ this knowledge to decrease the search involved in the design process. This type of knowledge is generally acquired by designers over periods of time with experience, but can also be obtained by accurate prediction techniques.

In this thesis, using accurate prediction techniques to guide the design process is presented. Even if there are very fast and very accurate prediction techniques, it would still be almost impossible to search the entire design decision tree due to its complexity. To overcome this problem, a different search tree is presented,

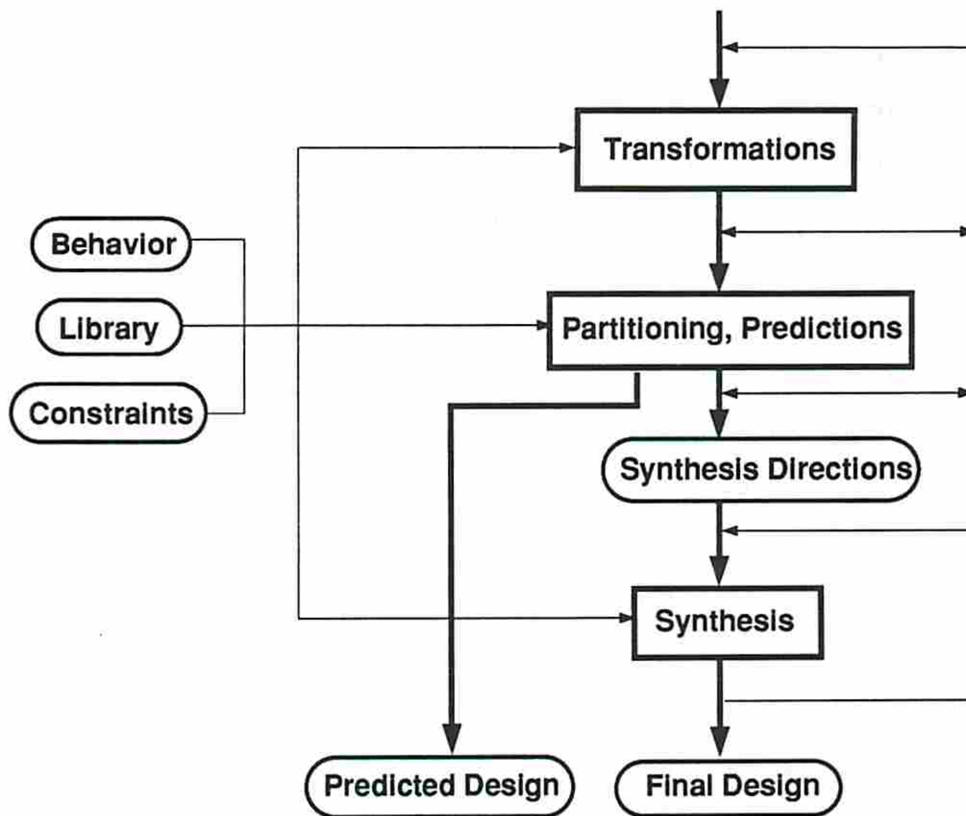


Figure 1.6: System-level Synthesis Process

which models what the actual synthesis tools can do, not what can be done in an idealistic system. This model is explained in more detail in Chapter 4.

The proposed method for system-level synthesis is shown in Figure 1.6. The first task in system-level synthesis is to evaluate several high-level transformations and find the transformed behavior which is predicted to result in the best implementation with respect to the given library and constraints. The predictor also provides directions on how to reach the predicted implementation of the transformed behavior. By following these directions, the behavioral synthesis system can produce designs in close proximity to the predicted design. The prediction techniques are inherently less sensitive to the variations caused by local optimizations. Therefore, the design search space can easily be reduced to a subspace using predictions as shown in Figure 1.7, followed by extensive local optimizations by the synthesis tools within that subspace. The process outlined in Figure 1.6 is orders of magnitude faster than the conventional ad-hoc methods for synthesis and allows searching a larger design space to produce better quality designs in less design time.

Partitioning of a design onto multiple chips is performed before the behavioral synthesis process. Using accurate predictions, the behavior is partitioned onto multiple chips in such a way that the synthesized multi-chip design will likely be feasible with respect to the given constraints. The synthesis directions from predictions are again followed to synthesize each partition and the necessary hardware for inter-chip communications, data transfers and data buffering.

System-level issues are taken into account during the prediction phase of the partitioning process. While predicting the multi-chip implementation, the prediction techniques consider several system-level tradeoffs on how to implement individual partitions and how to integrate them.

High-level transformations can be applied before the partitioning. In this way, the effects of transformations on the global design can be predicted. After a partitioning is decided, further evaluation of transformations can be performed

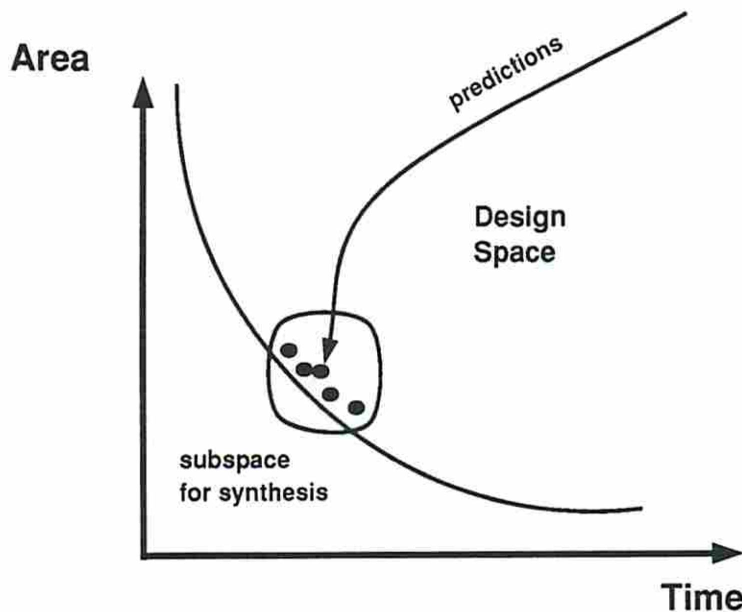


Figure 1.7: The Design Space Search in System-level Synthesis

before the synthesis process to find out if any partition-specific optimizations are favorable.

1.6 Examples

Some design descriptions will be used throughout this thesis. These design descriptions are relatively small and are used for demonstration of ideas.

A 16 point FIR Filter which is shown in Figure 1.8 is taken from [PP88] and subsequently used in the literature: [PK89b], [HCCd89], and [PK89a].

The AR Lattice Filter Systolic Array Element shown in Figure 1.9 is taken from [Kun84] and is subsequently used in [JPP87], [JKMP89], [KP91], and [PGH91].

Another design description is the fifth order elliptic wave filter and is shown in Figure 1.10. It is taken from Kung et al. [KWK85] and was chosen to be one of

the benchmark designs for high-level synthesis at the 1988 High-level Synthesis Workshop. It is one of the most commonly used design descriptions in the high-level synthesis literature.

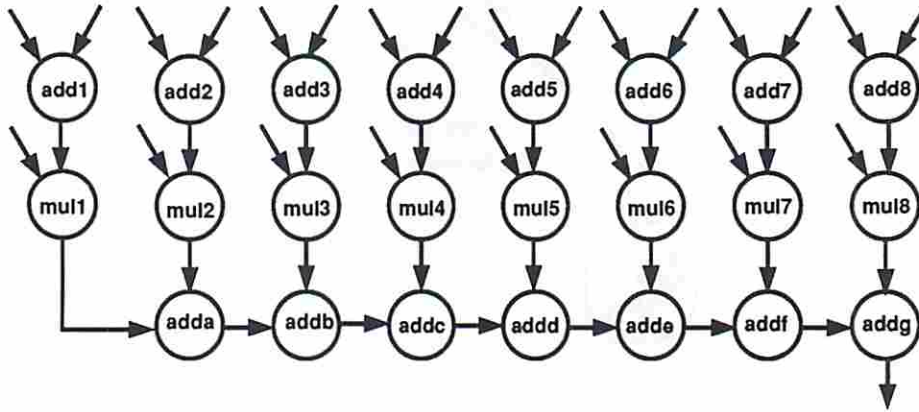


Figure 1.8: The FIR Filter Dataflow Graph

1.7 The library

A single design library shown in Table 1.1 was used throughout the thesis to compare results from different synthesis systems and to compare prediction results to the synthesized design characteristics except the data path tradeoff studies discussed in Chapter 3 in which a separate library was used.

Kurdahi generated several layouts for different adder and multiplier implementations using the MP2D placement and routing program [Kur87]. MP2D used standard cells from the RCA CADDAS 3 micron CMOS Library. An abstract library was created from these layouts and used by Jain [Jai89] and Mlinar [Mli91]. The library shown in Table 1.1 is similar to the libraries used by Jain and Mlinar. This library is taken from [JKMP89] with the exception of adder characteristics which are taken directly from [Kur87].

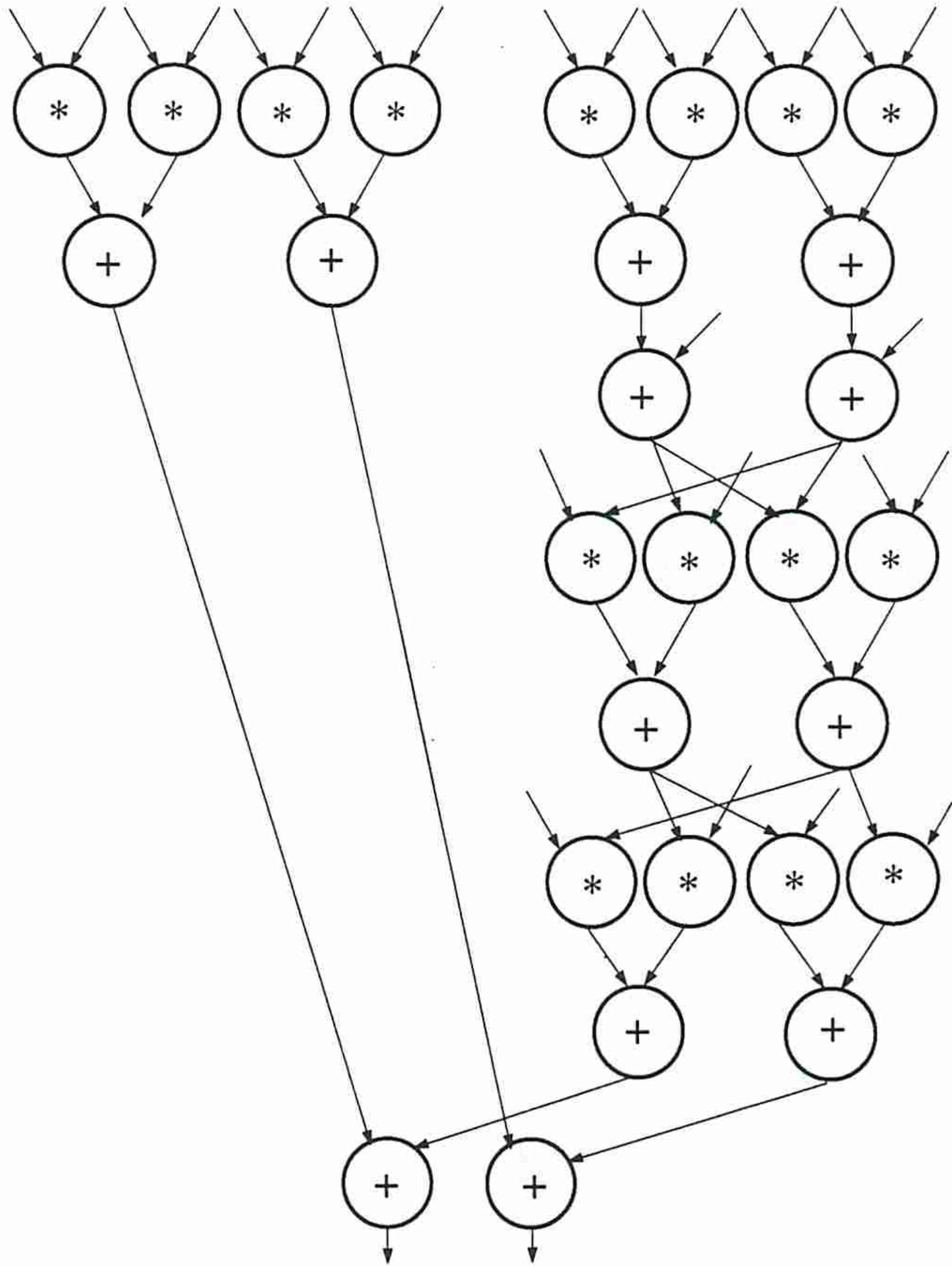


Figure 1.9: The AR Filter Dataflow Graph

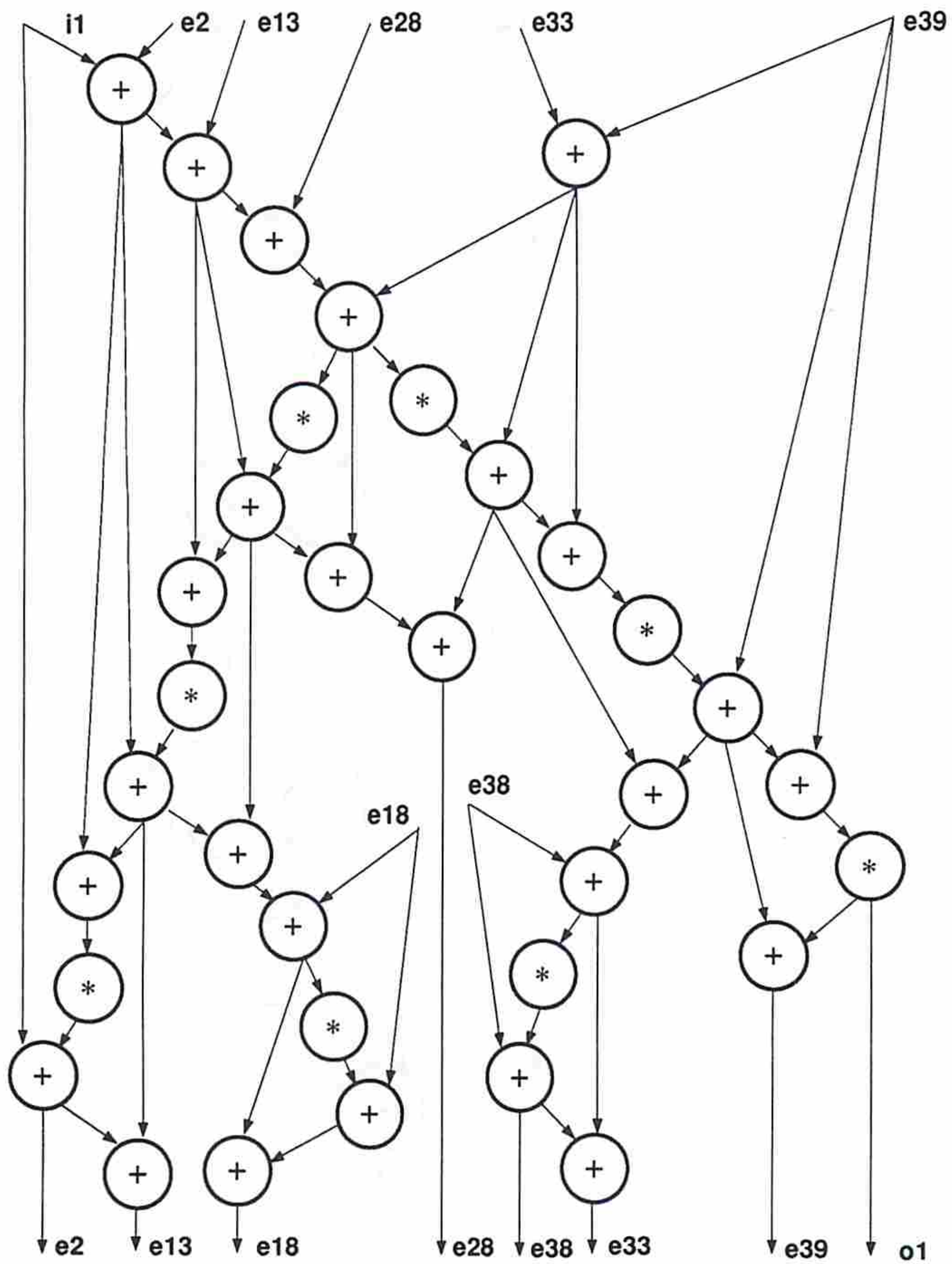


Figure 1.10: The Elliptic Wave Filter Dataflow Graph

Module Name	Type	Bit Width	Area mil^2	Delay ns
add1	Addition	16	4200	34
add2		16	2880	53
add3		16	1200	151
mul1	Multiplication	16	49000	375
mul2		16	9800	2950
mul3		16	7100	7370
register	Register	1	31	5
mux	2:1 Multiplexer	1	18	4

Table 1.1: The library used in the experiments

1.8 Thesis Organization

The research presented in this thesis naturally falls into three major groups; data path synthesis, behavioral predictions, and behavioral partitioning. Each research effort on these topics has been implemented as a stand-alone prototype although the primary purpose of the data path synthesis research was to produce validation tools for research at the system level.

Chapter 2 describes the related research. In accordance with the organization of the thesis, the related research for behavioral synthesis, predictions, and partitioning are presented.

Prior to this research, the ADAM Synthesis System had pipelined and non-pipelined scheduling programs, but there was no pathway to produce RTL designs from behavioral specifications. The research in data path synthesis and the implementation of the prototype software MABAL has enabled the ADAM Synthesis System to produce RTL designs.

The first part of Chapter 3 describes the data path synthesis research and the features of the prototype software, MABAL. Several experiments have been performed to compare MABAL results to the published results of other data path synthesis programs/systems. The second part of Chapter 3 describes a data path

tradeoff study performed using MABAL. In this section, several observations from this study are presented. These observations have been very useful to understand several issues in the behavioral synthesis process, and were extremely useful for the derivation of the prediction techniques. The data path synthesis work and the tradeoff study were reported in [KP89a, KP90d, KP90e].

Chapter 4 describes the research on the behavioral area-delay prediction techniques and the prototype software BAD. BAD combines theoretical, heuristic, and statistical prediction techniques around a statistical environment and is the first behavioral predictor which spans from design style selection to layout. This chapter also discusses how current prediction techniques evolved from earlier prediction research. The early work on BAD was reported in [Küç90, KP90a, PKPW91].

Chapter 5 presents the experimental results performed to validate the prediction techniques presented in Chapter 4.

Chapter 6 describes the behavioral partitioning research. The prototype interactive partitioner, CHOP is also presented. CHOP uses prediction techniques from BAD to help the designer to interactively partition behavioral specifications while considering system-level issues. The early work on CHOP was reported in [KP90c, KP90b, KP91, PKPW91].

Chapter 7 presents the experimental results for the behavioral partitioning research. The first set of experiments were performed to validate CHOP. The rest of the chapter presents experiments performed using CHOP to investigate system-level design issues.

Finally, Chapter 8 summarizes the conclusions of the research in the thesis and discusses future research problems.

Chapter 2

Related Research

In this chapter a number of related research efforts will be described in three major groups, in accordance with the organization of the thesis; behavioral synthesis, predictions, and partitioning.

2.1 Behavioral Synthesis

Behavioral synthesis has attracted a lot of interest in the past ten years. There have been numerous approaches to the problem. In this section, only examples of such approaches will be mentioned. A more detailed and comprehensive study of behavioral synthesis and related synthesis topics can be found in [MPC88, MPC90].

2.1.1 Behavior Specification

Several languages have been used to specify behavioral descriptions as input to high-level synthesis systems. The major difficulties in selection of the behavioral specification language involve specifying the behavior without introducing any structure or timing information which over-constrain the synthesis process. But, it is also important to be able to specify timing constraints as well as partial designs. The following is a partial list of input languages used by current behavioral synthesis systems:

- VHDL [VHD88] is used by the ADAM System at USC, the University of California at IRVINE and CMU. The VHDL subset used by the ADAM System is restricted to pure behavior. Although the popularity of using VHDL as a behavioral specification language for behavioral synthesis is increasing, the major problem with VHDL is that it was intended to be a simulation language. Therefore, the semantics of the language are not suitable for synthesis. Any timing information included in the language severely over-constrains the design. It is possible to use the *actual-delay* timing constructs in VHDL to specify the timing constraints without over-constraining the design, however, this changes the semantics of the language. The current solution to this problem is to either supply the timing constraints in the form of VHDL comment lines to be parsed by an intelligent parser or to supply the timing constraints separately. VHDL is also being used by the behavioral synthesis community to describe the output from behavioral synthesis (RT-Level Designs).
- Pascal used by Trickey [Tri87].
- ISPS [Bar81] used by System Architect's Workbench (SAW) [TDW+88].
- ADA used by ELF [GK84, GBK85].
- HardwareC used by Hercules [MK88].
- SILAGE [Hil85] used by Cathedral [GRVM90] and HYPER [RP90].

There is no agreement in the synthesis community to date on how a behavioral specification language should be designed so that it would have powerful constructs to support many very different synthesis methodologies without over-constraining the behavior or without being too detailed for practical use.

2.1.2 Data Representation

The data representations used by most behavioral synthesis approaches are quite different in style and structure but, in general, control and data flow information is stored using one or two flat or hierarchical graphs.

The Value Trace (VT) [McF78] of the CMU-DA System and the Control and Data Flow Graph (CDFG) of the HAL System merge the control and data flow graphs into one graph, while on the other hand, the Design Data Structure (DDS) [KP85] of the ADAM System keeps separate graphs for control and data flow and uses bindings to relate objects in these graphs.

2.1.3 Domain-dependent Data Path Synthesis

Mostly due to the complexity of solving the whole synthesis problem in the most general sense, some systems tend to specialize on specific domains and make use of more domain-specific knowledge [MPC90]. This results in systems which perform better for certain types of problem domains than others although the techniques used in such systems could also be applicable to other problem domains.

Behavioral synthesis is performed for different architectures in mind as well as for different problem domains. Three major problem-specific domains are listed below. The general-purpose data path synthesis approaches will be reviewed separately.

2.1.3.1 Digital Signal Processing

Digital Signal Processing (DSP) is an application area which is highly suitable for automatic synthesis and optimizations. FACE [SDD+89] and the Cathedral System [GRVM90] are fine examples of such systems combining the power of general purpose synthesis techniques and optimizations with the domain-dependent knowledge.

2.1.3.2 Microprocessor Design

SUGAR [RT85, TDW+88] is a part of SAW [TDW+88]. SUGAR is aimed at synthesis of microprocessors and has extensive knowledge of the microprocessor design style. Some of the domain-specific tasks performed in SUGAR are restructuring of control flow to allow fast decoding of instructions and selecting an efficient bus style.

2.1.3.3 Control Dominated Designs

The types of optimizations frequently employed in synthesis of control-dominated designs tend to be somehow different than the domains listed above. In control-dominated designs, the scheduling may be complicated with several timing constraints and resource sharing may not be desirable due to relatively smaller functional units. On the other hand, the control generation and controller optimizations play a more important role. A well-known example of such applications is synthesizing designs communicating with the outside world via specified bus protocols which generally have a complicated and pre-specified sequence of events.

ELF [Gir84] was the first data path synthesis system which accepted multiple timing constraints. These constraints were in the form of minimum and/or maximum execution time constraints for a portion of the behavior and/or loops.

ISYN [NT86] which is a part of SAW, allows incorporating local timing constraints into the data path scheduling program's (CSTEP) priority list scheduling so that the schedule generated for the data path satisfies the given constraints.

Janus [BK87] is a program which directly synthesizes asynchronous interface transducers from event graphs. Event graphs are derived from timing graphs. Janus outputs a logic circuit specification which is then further optimized with general-purpose logic synthesis tools.

CONSPEC [HP89] is a program which generates control specifications from detailed timing graphs. CONSPEC handles minimum and maximum timing

constraints, inner loops, and conditional cases and produces an implementation-independent control state table.

2.1.4 General Purpose Data Path Synthesis

A crude classification of data path synthesis approaches can be performed by classifying them according to types of methods used by each system. However, the boundaries of this type of classification are not distinct because most systems use different classes of methods for different synthesis subtasks and fall into multiple categories. As it will be seen from the rest of the section, most data path synthesis systems use one or more of the following:

- mathematical programming approach,
- procedural heuristic approach,
- rule-based approach,
- graph-theoretic approach (clique partitioning, bipartite matching), and
- search-based approach.

2.1.4.1 Mathematical Programming

Mathematical programming for behavioral synthesis was used by Hafer and Parker [HP83] in an early system. This approach formulated a brute-force mixed-integer-linear program, and therefore suffered from long run-times. The use of mathematical programming was more or less abandoned for a period due to its run-time complexity and has recently begun to be popular again.

An integer linear programming approach for scheduling is proposed by Hwang et al. [HLH91]. ALPS brings an integer programming solution to the scheduling problem by handling several variations including multi-cycle

and pipelined operators, operator chaining, functional pipelining, loop folding, and conditional operations. ALPS can perform time-constrained, resource-constrained (not area constrained), and feasible-constrained scheduling. The module binding and interconnect allocation phases are later performed using heuristics.

Papachristou and Konuk proposed a scheduling method using hill-climbing search followed by a heuristic interconnect allocation [PK90]. In this formulation, linear programming is used as a cost function to estimate the cost of any particular partial schedule.

Another ILP approach for simultaneously scheduling and allocating functional units is given in [GE91].

2.1.4.2 Scheduling

Several heuristic solutions have been proposed to solve the scheduling subtask in behavioral synthesis. Among these methods we can easily count

- force-directed scheduling in HAL [PK89b] and SAM [CT90],
- critical path scheduling using freedoms in MAHA [PPM86],
- list scheduling in CSTEP [TDW+88],
- urgency scheduling in ELF, and
- functional pipeline scheduling using urgency scheduling in Sehwa [PP88].

2.1.4.3 Allocation, Module Assignment, Interconnect Allocation

ELF and DAA [KT83] are examples of systems using a ruled-based allocation approach. Using a rule-based approach, the knowledge from expert designers can be incorporated into the synthesis system. It is also possible to improve or change the rule set when more knowledge is available.

A clique partitioning approach is used in Facet and Emerald [TS86]. The module allocation and module assignment are formulated as a clique partitioning problem, but later solved by a heuristic clique partitioning technique rather than the exact solution mainly due to run-time complexity problems. The register allocation and value binding are also formulated and solved in the same way. But, these tasks are performed independent of each other which reduces the chances of reaching a global minimum.

Splicer [Pan88] is an example system using extensive search techniques to perform synthesis. The technique used in Splicer is based on branch-and-bound search to find the best operation assignment while generating the interconnect simultaneously. A user-specified lookahead is used to search a larger portion of the design space.

HAL and SAM both use force-directed scheduling but take different paths to complete the synthesis process. The allocation approach in HAL is rule-based, while on the other hand, SAM performs allocation together with scheduling using the force-directed concept.

EMUCS [Hit83, TDW+88] is a data path synthesis tool which tries to minimize the overall cost of the design using a procedural heuristic. EMUCS uses McFarland's greedy minimax heuristic which binds operations and values one at a time to minimize a cost function. The intuitive idea of the algorithm used in EMUCS is to give priority to decisions that may have the worst impact on the design if delayed in the incremental binding process. At each binding step, an update of a cost table containing the costs of all bindings is made. This process of updating is time consuming; however a much faster local update method can be used at the user's option, at the expense of missing some of the good designs which could be found otherwise.

IMBSL [Kna89] is a system which accepts partial structures, allows designer intervention, and constructs a layout model for detailed analysis of area and timing. IMBSL starts after the scheduling phase of the synthesis process.

LYRA and ARYL [HCLH90] formulate the operation assignment task as a bipartite graph matching problem. They solve the interconnect allocation by a greedy heuristic after register allocation and operation assignment are performed. LYRA first performs register allocation, followed by operation assignment. ARYL first performs operation assignment, followed by register allocation.

BUD [McF86] first builds a hierarchical clustering tree based on common functionality, common interconnections, and potential parallelism. Cutting this tree at different levels produce different configurations. These cuts are then used to perform resource allocation and operation assignment. A scheduling is performed for each configuration and final design characteristics are estimated using an approximate floorplan.

2.2 Prediction Methods

Although most high-level synthesis programs do not use predictions for the design characteristics yet to be determined, we cite BUD [McF86], ELF [Gir84], CHIPPE [BG90] and [WP91] as exceptions. The prediction results are used to help the synthesis programs to make design decisions. BUD estimates physical design characteristics like area and wire delays by approximate floorplans. ELF predicts the wiring area from RTL characteristics to make design decisions. CHIPPE predicts the worst case wiring delays after behavioral synthesis (without constructing a floorplan) and uses this delay information in the evaluation mechanism of its iterative synthesis process. Also, the ADAM System [JKMP89] uses predictors to guide the design search at a higher level than the synthesis programs.

A 3D scheduling approach has been proposed by Weng and Parker [WP91] which takes floorplanning into account. In the improvement phase performed after a constructive scheduling, operation re-binding and adding redundant operators are tried to reduce the wire delays. The approach generates approximate

floorplans at each step to estimate layout characteristics (layout area and wire delays).

The early work on the estimation of functional resource allocation was on the prediction of lower-bound functional area for varying time characteristics for pipelined and non-pipelined design styles by Jain and Parker [JPP87, JMP88]. This prediction research was geared to estimate the resource allocation behavior of Sehwa [PP88] and MAHA [PPM86], the pipelined and non-pipelined scheduling programs in the ADAM synthesis system. Only single-cycle operations were considered in this work. The clock cycle was chosen by the prediction method in the same way that the corresponding synthesis tool (MAHA or Sehwa) decides. Some loose theoretical lower bounds are also given for register and multiplexer area in [JPP87].

The HYPER system uses a min-max search method to perform behavioral synthesis [PR89]. The search is performed in the design space by varying the resources in the design (operators, registers, and interconnect elements). Whenever possible lower and upper bound predictions are used to set bounds on the resources, thereby constraining the design space search performed by the min-max search method. Estimations are also used to guide the transformations in the HYPER system.

An estimation technique for the functional unit allocation was recently proposed by Hagerman [Hag91]. This technique makes use of the concept of distribution graphs [PK89b] to estimate the functional unit allocation to be produced by an imperfect scheduling tool (e.g. a force-directed scheduling tool). The estimation results compare favorably to the results obtained using a force-directed scheduling tool [CT90]. This prediction technique raises a good question: *Where to draw a line between estimation and synthesis ?* Because the more processing is done in the estimation phase, the better chances for the estimation results to be closer to the synthesis results. But, on the other hand, the complexity of the estimation becomes closer to the complexity of the synthesis process. For example,

the complexity of estimation technique proposed by Hagerman is $\mathcal{O}(E+N \log N)$ where E is the number of edges in the dataflow graph and N is the number of operations in the dataflow graph. This complexity is lower than the complexity of the force-directed scheduling method. But, there exists scheduling techniques which are simpler than the force-directed scheduling and have the same type of run-time complexity as this estimation technique.

The existing methods used at USC for register [Mli91] and wiring [KP89b] area estimation, although fast enough for human interaction, are inherently too slow to be practical when called hundreds or even thousands of times by other programs.

PLEST is a program which estimates the layout area of an RTL design for the standard-cell placement and routing style implementation. PLEST considers varying aspect ratios for the layout, feed-throughs, and track density and estimates the layout area using probabilistic methods. PLEST results were verified against RCA CADDAS standard-cell layouts and were within 10% of actual layout areas [KP89b].

Another layout area estimation technique was proposed by Chen and Bushnell [CB88] for standard-cell and full custom placement and routing implementations. Although their average estimation error was reported to be 12%, some reported errors go up to 40-70% since the approach does not assume that multiple nets can share the same tracks.

Zimmerman proposed a constructive layout estimation technique based on predicting shape functions for a slicing geometry [Zim88]. A structural hierarchy as a slicing geometry and shape functions are only known at the leaf nodes. Shape functions of leaf nodes are used to predict shape functions of composite nodes and this process is hierarchically repeated to predict the overall layout area. The authors claim that they can predict the layout area to within 5 to 10% accuracy. There are no run-times reported in the paper.

In [KR91], the authors propose an layout estimation technique which uses both analytical and constructive methods. In contrast to Zimmerman's method, LAST [KR91] performs slicing up to a user specified depth and then use an analytical approach to estimate the shape function at that level. The analytical approach used is a version of PLEST. Since the method does not travel through the whole slicing tree, it has a favorable speed advantage over purely constructive methods.

2.3 Partitioning

2.3.1 Partitioning of Graphs

Partitioning graphs with costs on the edges into subgraphs of specified sizes while trying to minimize the total cost of edges cut has been a popular problem. Many heuristics have been devised (e.g. [FM82]) and successfully applied, most of which are based on Kernighan and Lin's work [KL70]. This partitioning formulation is suitable for partitioning of RTL and logic circuits. However, behavioral synthesis introduces significant and generally irregular sequential behavior into the design. This causes most final design characteristics to be a function of the sequential behavior introduced and hence a function of the structure as well as of the original behavior. Without having the results of behavioral synthesis or accurate predictions of these results, it has not been shown if one can directly correlate *sum of costs of values cut* to the pin count requirement or *weighted sum of operations in a partition* to the area of chips.

2.3.2 Partitioning in Microprocessor Design

Partitioning has been used at many stages of the microprocessor design process. Floating-point co-processors, I/O peripherals and memory management units

are among the most common partitions. A study on partitioning strategies for multi-chip VLSI microprocessors was performed [Sch84].

2.3.3 Partitioning of Logic Circuits

Kodres formulates the partitioning of logic circuits onto chips, boards, and cabinets [Kod72]. In this formulation, the device size constraints (e.g. chip or board size) and external device connection constraints are taken into account.

An automatic partitioning method for logic circuits in order to meet gate and pin count constraints of chips was described by Payne and vanCleemput [Pv82]. In this approach, a quality function was used to measure the costs of partitions, the compactness and the reliability of the overall design.

Another reason to partition large logic circuits is to reduce the design size to be processed by the logic synthesis tools, which in turn, reduces the memory and run time requirements during logic synthesis [CB87].

SLIP [BKM+89] is a software environment for system-level interactive partitioning. It is a framework providing a means for maintaining and modifying the design hierarchy for general purpose partitioning algorithms.

Saab and Rao proposed an evolution-based approach to partition logic circuits [SR89]. Their multi-way partitioning takes into account constraints (e.g. the size of each part and the number of pins) to minimize the total number of nets cut. The method takes testable and critical nets into account during partitioning. Testable nets are cut so that they are observable in the final design and critical nets are not cut so that the delay penalties due to partitioning are minimized.

2.3.4 Partitioning of RT-Level Circuits

SPARTA [Res86] is a tool which evaluates an RTL design with a spreadsheet-like approach which checks if any area, power and pin count constraints are violated. However, SPARTA is performs highly complicated calculations which are not

suitable for any row/column type of spreadsheets. Examples of such calculations include finding off-chip interconnections and calculating delays.

2.3.5 Partitioning of Behavioral Specifications

Partitioning of behavioral specifications is a process of assigning operations of a behavioral specification to partitions with the assumption that the operations in different partitions will not share hardware (e.g. operators). Partitioning of behavioral specifications can be performed with the goal of a single or multi-chip implementations.

A manual partitioning as a part of the behavioral synthesis process is described by Walker and Thomas [WT89]. Among other high-level transformations, manual partitioning as a process creation is discussed. The necessary data structures for the multi-partition implementation are automatically created so that the synthesis tools in SAW can synthesize a multi-partition design.

McFarland partitions behavioral specifications with a clustering algorithm based on a *similarity* measure [McF83]. These clustering algorithms are employed in BUD [McF86] which is used as a front-end to DAA [MK86], to perform a part of allocation and module binding phase of the data path synthesis process.

A behavioral partitioning algorithm proposed by Lagnese and Thomas [LT89, LT91] uses a similar but multi-level clustering approach to partition the behavioral specification onto multiple processes which improves the quality of single-chip designs. The approach makes use of the information about the CDFG (Control and Data Flow Graph) topology as well as behavioral synthesis to reach the best intuitive partitioning. Their results show significant area reductions for the partitioned single-chip designs, but their approach does not consider design constraints or multi-chip design issues.

Camposano and van Eijndhoven [CvE87] describes a partially manual and partially automatic partitioning method used in the Yorktown Silicon Compiler (YSC). The automatic partitioning algorithm forms clusters of operations in the

behavioral specification minimizing the number off-cluster signals while keeping the cluster sizes under a specified limit. The manual partitioning is performed by arranging operations into separate procedure calls.

A graph-theoretical behavioral partitioning effort has been reported by Gupta and De Micheli [GM90]. The designer manually finds a starting partitioning which satisfies the timing constraints, and a preliminary schedule for the design is produced, then the Kernighan-Lin algorithm or simulated annealing is used to finalize the partitioning. The operations have an abstract area corresponding to the number of literals. The authors do not consider pin-sharing or area/delay characteristics of registers, multiplexers, controllers, or wiring.

CAMAD [Pen86] system takes a different approach. An Extended Timed Petri Net model is used as the behavioral-level specification. Costs derived from the Petri Net represent the importance of the data/control connectivity and these costs are assigned to places, transitions and edges connecting them in a way to merge the partitioning of data path and control constructs. Then, the Kernighan-Lin algorithm is used to perform a divide-and-conquer partitioning of the behavioral specification.

Chapter 3

Data Path Synthesis

3.1 Introduction

As mentioned in Chapter 1, the buses and multiplexers required to interconnect functional modules and registers may have a first-order effect on hardware cost (silicon area) in a digital design. Furthermore, the number and types of interconnections required are heavily dependent on the number of functional modules and registers in the design, as well as the assignment of operations and values to these units. Thus, interconnect costs must be considered at the same time that other high-level design decisions are being made.

Synthesis of data steering structures has been demonstrated by early systems like the program by Hafer [HP78] to more recent packages with marked improvement over initial attempts. However, a number of issues remain to be addressed in order for high-level synthesis to be practical. Data path synthesis researchers are often confronted with the comment that *it may be advisable to duplicate functional logic because routing and switching costs outweigh the savings when functional resources are shared.* If this is true, a synthesis program should somehow take into account actual wiring costs when trading off interconnect complexity for functional complexity. Furthermore, a synthesis program should be comprehensive. It should not be limited to a single style of interconnect. It should include the ability to trade off between bus drivers and multiplexers

and the ability to trade off between data steering logic and functional logic. In order to support tradeoff studies it should be fast enough to support multiple iterations with designer-imposed decisions and constraints, and should be able to search different parts of the design space with different user constraints. Finally, it should provide summary information to the designer as output data so that tedious details about the design do not have to be digested and evaluated.

A design with non-optimal allocation of registers and functional modules may turn out to be cheaper overall than the design with optimal resource allocation, since either interconnect costs can have a first-order effect on total cost or a non-optimal resource may allow a better optimized, hence cheaper interconnect. The heavy resource sharing required when functional resources are scarce can cause multiplexing and bussing costs to dominate the total cost. As a result, the range of designs searched has to be totally unconstrained to get optimal results, which causes the search to be computationally intensive.

3.2 Overview of MABAL

3.2.1 Problem Statement

The basic problem Module And Bus ALlocator (MABAL) solves includes allocation of functional modules and registers, module binding and creating the interconnect. The function performed can be modeled as a mapping from the domain consisting of all scheduled operations and values to a range consisting of functional modules, registers and their interconnect consisting multiplexers, bus drivers and wires.

Definition 3.2.1.1 *Let $B(O, E, V, T)$ be the scheduled behavior where*

O is a set of operation nodes;

E is a set of directed edges;

V is a set of values;

T is a set of timing tuples

such that

1. *(O, E) forms a directed and acyclic graph;*
2. *Each $t \in T$ is a timing tuple (t_{id},s,e) where*

   t_{id} is operation (value);

   s is the start (birth) time for the operation (value); and

   e is the end (death) time for the operation/value.

In general, only timing tuples about the operations are given in the specification. Values and the timing tuples for values are extracted from $G(O, E)$. This process, even if it may seem to be straightforward, may be quite involved when conditionals are present and values can be broken into multiple instances.

Definition 3.2.1.2 *Let S(M,R,I,C) be a directed graph representing the structure onto which B(O,E,V,T) is mapped, where*

M is a set of functional modules;

R is a set of registers;

I is a set of interconnect modules (steering logic); and

C is a set of directed edges representing wires (carriers) connecting elements in M,R, and I.

Definition 3.2.1.3 *Let A(P,Q) be the structural binding (assignment) information where*

1. *$\forall p \in P$ is a 2-tuple (o,m) such that*

   $o \in O$;

$m \in M$; and

o and m have the same type.

2. *$\forall q \in Q$ is a 2-tuple (v,r) such that*

$v \in V$;

$r \in R$; and

v and r have the same bitwidth.

MABAL's mapping task can be loosely expressed as

$$MABAL : B(O, E, V, T) \longrightarrow S(M, R, I, C)$$

and to produce the structural binding information, $A(P, Q)$ to minimize an area cost function $F(M, R, I, C)$.

The more general mapping of the scheduled behavior onto the structure can not be fully defined before the RTL design is complete. During allocation and binding, not only are the best bindings from behavior (domain) to structure (range) to be found, but also the range is to be determined simultaneously. Although the lower bound on the number of functional modules can be calculated using some existing theory [Par85], prior to RTL synthesis there is very little know about the interconnect for a given design, whose cost is highly dependent on the way in which module binding is performed. A complication of the problem is that a design with the minimum allocation of functional modules and registers does not guarantee a small interconnect cost. In fact, a design with minimum number of registers and functional modules may turn out to be more expensive overall than a design with non-minimal number of registers and functional modules, since interconnect costs can have a first-order effect on total cost. The heavy resource sharing required when functional resources are scarce can cause multiplexing and bussing costs to dominate the total cost. As a result, the range of

the mapping has to be totally unconstrained to get optimal results, which causes the complexity of finding the best mapping to be computationally intensive.

3.2.2 Features of MABAL

MABAL performs functional module allocation, register allocation, operation/value binding and data steering logic allocation concurrently, with the objective of minimizing the total area (cost) by selecting between allocating more functional modules and registers or data steering logic. MABAL handles both pipelined and non-pipelined design styles and also allows operator chaining. Information about conditionals, constants and commutativity of operations is used to minimize interconnect cost and share resources. MABAL handles designs with outer loops in which values are fed back from primary outputs to primary inputs. MABAL allows the user to constrain inputs and outputs or even partially specify a design, while checking for errors in designer's inputs, and constructing interconnect which is not restricted to a single style.

Variations in input and output signals are supported by MABAL. Any input to the target design can be selectively latched or left unlatched upon user request. The input values can be constants which do not require storage, loop variables used for multiple iterations, variables sampled once at the beginning of the execution and stored until the end, and finally variables which are sampled at each iteration of a loop body. Any output from the target design can be selectively latched for a user-specified time duration. Since MABAL allows the user to decide when and how to latch inputs and outputs, the problem of interfacing several pieces of a larger design becomes simpler.

Data steering logic allocation consists of allocating multiplexers, bus drivers and carriers. There is no enforced bus style. The resulting interconnect is decided entirely by making tradeoffs between multiplexers, bus drivers, functional modules or registers subject to optional bindings and restrictions requested by the user. Unless the user restricts the use of bus drivers, multiplexers and bus

drivers may be mixed depending on the relative costs of these modules. MABAL is capable of creating highly complicated interconnect structures. To illustrate the model used by MABAL, a possible design MABAL might produce is shown in Figure 3.1.

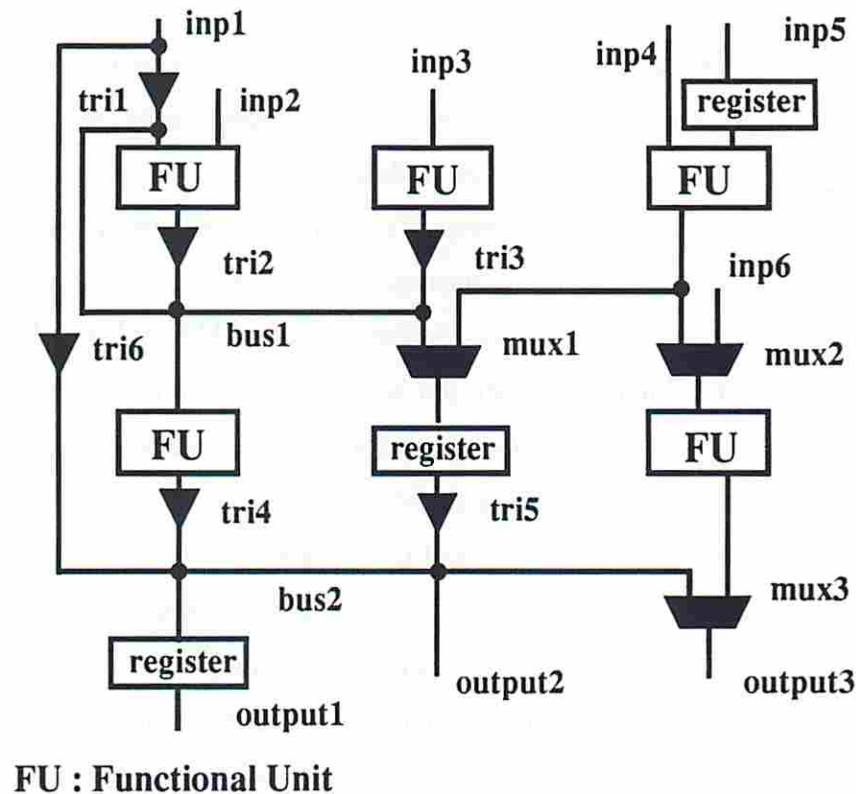


Figure 3.1: A possible design MABAL might produce.

One of the key features of MABAL is the ability to allow the designer to influence the design process, while automating the tasks that the designer does not want to control[3.1]. The user-supplied structural information is checked and, in case there are errors, corrections are made by the program during execution. If a complete design is given to MABAL, the program can be used to validate the design and correct it if necessary.

[3.1]However, MABAL does not require designer intervention to produce good results.

Depending on user-supplied structural information and parameters (e.g. initial resource allocation, bindings and restriction of bussed connections for specified ports) different parts of the design space can be explored. Since an incremental algorithm which builds on the existing design is used, every different starting point in the design space may yield a different design. Very short run times allow designers to iterate frequently, spending more time on higher-level decisions which further improve the overall quality of the final design.

As stated earlier, it might be desirable to duplicate some small functional modules and registers to save routing area. To accomplish this, a parameter which indicates a degree of tendency for MABAL to use more functional modules and registers, or more interconnect logic at individual decisions can be set by the user to override a MABAL decision. This parameter can be derived from the statistical data on the synthesized designs, the characteristics of the library, and the technology used and can be effectively used to make tradeoffs between using more functional area or interconnect area. A similar measure is also used in [HCCd89, WP91].

One of the useful features of MABAL is that it produces detailed summary information about the design constructed. This summary information includes utilization for each functional module and register as well as functional module type and global register utilizations. Bus utilizations and total hardware utilization are also reported. In addition to utilization summaries, global design characteristics are also summarized; functional area, interconnect area, total area, number of abstract multiplexers, number of abstract multiplexer inputs, total number of 2-to-1 multiplexers to implement the abstract multiplexers , number of bus drivers, and number of nets.

The summary information from MABAL is very useful to pinpoint problems with the implementation. It is very easy to find out if the schedule is good or not from the utilizations of functional modules and registers. For example, under-utilizing some of adders in a design may not be a serious problem, but

under-utilizing multipliers which can easily be 10 times larger than an adder shows a serious problem with the schedule and can easily be detected. Often, there exist several schedules with the same resource requirements and similar timing characteristics. But depending on the schedule, the number registers required to store intermediate values may vary considerably. Since MABAL is very fast, it can be used effectively to differentiate between such resource-equivalent schedules.

3.2.3 Assumptions

In our approach to this synthesis problem, the following assumptions hold in order to be consistent with the remainder of the USC ADAM system. In practice, these assumptions do not severely restrict the design space explored.

- Any operation has to be completed within one clock cycle, or the operation and the operator must be partitioned in a way that each partition can be completed within a clock cycle.
- Scheduling has to be performed prior to running MABAL.
- Module style selection has to be done prior to both MABAL and the scheduling task.
- Operations the designer desires to implement with different module types must have been given different type names even if they perform identical functions. Usually, there is more than one functional module type which can implement an operation type; the module selection program which runs prior to MABAL decides which one is better to use for each operation type.
- The final design is synchronous with a single clock.

3.2.4 Inputs

MABAL's inputs can be divided into 3 categories:

- A scheduled dataflow graph with information about conditional branches
- a set of library components which can implement all operations
- optional information in addition to the required data
  - an initial resource allocation which might be modified by the program,
  - bindings of operations and values to functional modules and registers,
  - restrictions on the use of bussed connections at given inputs of given modules, or throughout the design, and
  - a factor which modifies the relative cost of interconnect modules to functional modules and registers.

Although an entire RTL design can be specified as an input to MABAL, the designer must use the restricted interconnect model used by MABAL. Multi-level interconnections, buses, arbitrary bindings of values to time steps, multi-phase clocking schemes and complicated interconnect elements (e.g. inverting bus drivers) are not allowed. Although specifying buses to MABAL is easy, it has not been implemented yet because of the fact that the global effects of predefined buses on the algorithm used by MABAL are not yet predictable.

3.2.5 Outputs

MABAL completes data path synthesis and generates an RTL design without the controller. It specifies all functional modules, registers, multiplexers, bus drivers, inputs, outputs, connections between them and bindings. Each operation is bound to a single module, whereas each value can be bound to a number of registers if necessary. The input arrangements of multiplexers are arbitrary and can be optimized by a controller synthesis program.

3.3 Theoretical Basis for Tradeoffs

There are two types of tradeoffs performed by MABAL. The first tradeoff is between using multiplexers and bus drivers while creating the necessary interconnect to share functional modules and registers. The second tradeoff is between sharing a resource (functional module or register) and introducing a new resource into the design.

In order to perform these tradeoffs, a cost function to be used in evaluating tradeoff decisions is needed. An accurate model should take placement and routing effects into account. Unfortunately, this data is not available until after the design is completed. Therefore, the following simple RTL Area Model will be used while performing/evaluating tradeoffs.

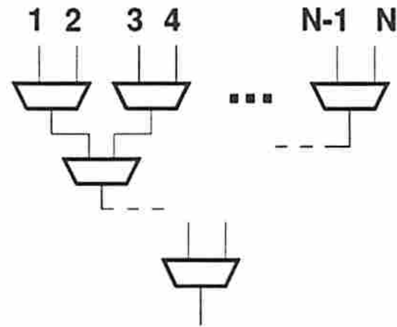
Definition 3.3.4 Uniform RTL Area Model *is a model in which the design area is only composed of area of individual modules (functional modules, registers and interconnect modules* [3.2]*) and area taken by nets and is calculated by a simple algebraic addition. Each net is assumed to occupy an area which is proportional to the equivalent number of 2-point nets for the net. An m-point net is defined to be equivalent to* $(m-1)$ *2-point nets. It is also assumed that any two design entities of the same type (e.g. adders, 2-to-1 multiplexers, or bus drivers) have the same characteristics.*

3.3.1 Multiplexer versus bus driver tradeoff

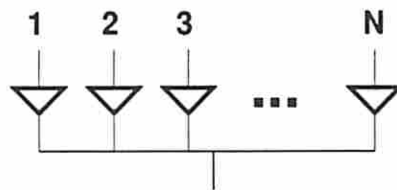
In our model it is assumed that any multiplexing network is created by nested 2-to-1 multiplexers, and bus drivers are powerful enough to drive high-fanout buses. Figure 3.2 shows how one input out of N unique inputs ($N > 1$) can be selected using either a multiplexer network or bus drivers. It is also assumed in Section 3.3 that multiplexer networks and buses do not mix. The local switching logic to share a functional module or register is implemented in whole either by a

[3.2] interconnect modules: multiplexers and tri-state drivers.

multiplexer network or a bus style interconnect. This restrictive assumption will be relaxed in MABAL since more data on the partial design is available during synthesis.



(a) Multiplexer Network



(b) Bus Style

Figure 3.2: Two possible styles for the data steering logic

Using the uniform RTL area model, the area,IC_{mux} of the multiplexing network shown in Figure 3.2 (a), is calculated as

$$IC_{mux} = (N-1)A_m + 2(N-1)A_{net} \tag{3.1}$$

where A_m is the area of a 2-to-1 multiplexer, and A_{net} is the area occupied by a 2-point net.

Again using the uniform RTL area model, the area, IC_{bus} of the bus network shown in Figure 3.2 (b), is calculated as

$$IC_{bus} = NA_{bd} + (2N-1)A_{net} \tag{3.2}$$

where A_{bd} is the area of a bus driver. Without loss of generality, we assume that each value in the design have the same bitwidth and A_m, A_{bd} and A_{net} are already adjusted for that bitwidth.

Theorem 3.3.1.1 *Given A_m, A_{bd}, A_{net}, N, and the uniform RTL area model, the multiplexing network area is less than the area of the bus style interconnect if and only if*

$$\begin{cases} A_m \leq A_{bd} \quad \textit{or} \\ A_m > A_{bd} \quad \textit{and} \quad N < \dfrac{\mathrm{A_m + A_{net}}}{\mathrm{A_m - A_{bd}}} \end{cases}$$

Proof: From Equations 3.1 and 3.2, the claim follows that

$$(N-1)A_m + 2(N-1)A_{net} < NA_{bd} + (2N-1)A_{net}$$

After a simplification process, we reach

$$N(A_m - A_{bd}) < A_m + A_{net} \tag{3.3}$$

Case 1: If $A_m > A_{bd}$, then Equation 3.3 can be rearranged as

$$N < \frac{A_m + A_{net}}{A_m - A_{bd}}$$

which is the necessary condition stated in the theorem.

Case 2: If $A_m < A_{bd}$ then Equation 3.3 can be rearranged as

$$N > \frac{A_m + A_{net}}{A_m - A_{bd}}$$

which is satisfied for all N since $N > 0$.

Case 3: If $A_m = A_{bd}$ then Equation 3.3 becomes

$$0 < A_m + A_{net}$$

which is again satisfied at all times. □

3.3.2 Resource sharing versus resource duplication tradeoff

Given a partial design, a major decision in incremental binding is whether to share an already allocated resource or introduce a new instance of the resource to perform an operation. If the interconnect tradeoff presented in the previous section is used for each input port j of the resource, the switching logic area overhead, IC_j for sharing input port j of the resource will be

$$IC_j = \min(IC_{j(mux)}, IC_{j(bus)})$$

where $IC_{j(mux)}$ and $IC_{j(bus)}$ are the areas of multiplexer and bus implementation of the switching logic for input port j of the resource. Then, the total area overhead to share the resource is

$$\sum_j \min(IC_{j(mux)}, IC_{j(bus)})$$

where j varies over the input ports of the resource.

Let A_R be the area of the resource under consideration for resource sharing. The area impact of introducing a new instance of the resource and connecting its inputs to other modules before any steering logic is needed is $A_R + \eta A_{net}$ where η is the number of inputs of the resource. If

$$\sum_j \min(IC_{j(mux)}, IC_{j(bus)}) < A_R + \eta A_{net}$$

then sharing the resource will result in smaller area. Otherwise, duplicating the resource takes less area. Of course, it is important to note that this is only a local decision whether to share an already allocated resource or introduce a new instance of the resource to perform an operation. Therefore, this type of decision making does not necessarily lead to the *best* possible design.

3.3.3 An upper bound on the interconnect area

Given the uniform RTL area model, an upper bound on the interconnect logic requirement for a special class of designs can be calculated. This class of designs consists of non-pipelined implementations of designs with no conditionals.

Without any loss of generality, we will assume in this section that all operations/values have the same bitwidth and all operation types have a single input.

Lemma 3.3.3.2 *Given the uniform RTL area model, consider the operator allocation and resource sharing for only one operation/operator type, namely type i in any non-pipelined implementation of a design specification with no conditionals. Let n_i be the number of operations of type i to be performed. Then, the upper bound interconnect requirement for these n_i operations is higher for the single-operator implementation[3.3] than any multi-operator implementation.*

Proof: We assume that we have the flexibility of using multiplexers or buses for the interconnect as described in Section 3.3.1. We will prove that the upper bound interconnect requirement for the single-operator implementation is higher than any multi-operator implementation for either interconnect scheme.

The upper bound interconnect requirement deals with the worst case scenario in which all inputs used by operators are unique. In the single-operator implementation, the operator will perform $N = n_i$ operations and will have (in the

[3.3]Single-operator implementation uses a single operator to implement these n_i operations.

worst case) N unique inputs for the interconnect. In a multi-operator implementation, each operator will perform N_j operations and there will be p operators. We will have (in the worst case) N_j unique inputs for the interconnect to each operator j subject to

$$N = n_i = \sum_{j=1}^{p} N_j \qquad\qquad p > 1$$

Case 1: Multiplexing network

For the single-operator implementation of n_i operations of type i

$$IC_{mux} = (N-1)A_m + 2(N-1)A_{net}$$

For the multi-operator implementation of n_i operations of type i

$$IC'_{mux} = \sum_{j=1}^{p}(N_j - 1)A_m + 2(N_j - 1)A_{net}$$

which can be rearranged as follows:

$$\begin{aligned} IC'_{mux} &= A_m \sum_{j=1}^{p}(N_j - 1) + A_{net} \sum_{j=1}^{p} 2(N_j - 1) \\ &= A_m[(\sum_{j=1}^{p} N_j) - p] + A_{net}[2(\sum_{j=1}^{p} N_j) - 2p] \\ &= A_m(N-p) + A_{net}\, 2(N-p) \end{aligned}$$

It is clear that IC_{mux} is larger than IC'_{mux}, and IC'_{mux} degenerates to single-operator case when $p = 1$.

Case 2: Bus style interconnect

For the single-operator implementation of n_i operations of type i

$$IC_{bus} = NA_{bd} + (2N-1)A_{net}$$

For the multi-operator implementation of n_i operations of type i

$$IC'_{bus} \sum_{j=1}^{p} N_j A_{bd} + (2N_j - 1)A_{net}$$

which can be rearranged as follows;

$$\begin{aligned} IC'_{bus} &= A_{bd}\sum_{j=1}^{p} N_j + A_{net}\sum_{j=1}^{p}(2N_j - 1) \\ &= NA_{bd} + A_{net}(2N - p) \end{aligned}$$

It is also clear that IC_{bus} is higher than IC'_{bus}. We have proved that the upper bound interconnect requirement for an operation type is higher for the the single-operator implementation than any multi-operator implementation even when it is possible to tradeoff between multiplexers and buses. □

Theorem 3.3.3.3 *Given the uniform RTL area model and the assumption that primary inputs are to be stored externally, the upper bound interconnect requirement for any non-pipelined implementation of a design specification with no conditionals,* IC_{max} *is given as*

$$IC_{max} = \sum_{i} \min((n_i - 1)A_m + 2(n_i - 1)A_{net}, n_i A_{bd} + (2n_i - 1)A_{net}) \tag{3.4}$$

where i *varies over all operation/value types.*

Proof: The proof follows from Lemma 3.3.3.2. In our synthesis model as described by Definition 3.2.1.3, all operations/values of the same type are to be implemented by the same type of operators/registers and the interconnect for registers are not different than interconnect for regular operators. Each upper bound is only a function of the resource sharing requirement of the operation/value type. Therefore, the upper bound interconnect requirement for each operation/value type can be calculated independently using Lemma 3.3.3.2. The summation of the upper bound interconnect requirements given by Lemma 3.3.3.2 for each operation/value type gives the the upper bound interconnect requirement for the whole design. □

3.4 Overview of MABAL Operation

3.4.1 How MABAL performs synthesis

MABAL performs synthesis using the following approach. A greedy, incremental algorithm is used by MABAL with some decisions reversible later. Although the approach used does not allow backtracking, the reversible decisions provide a similar, but a more limited effect of exploring a larger portion of the design space without actually searching that larger portion of the design space. In this way, the negative consequences resulting from early decisions which are not suitable for the final design are reduced. MABAL might start with no structural design or a given structural design in place. Resource allocation is incrementally performed as needed. The idea behind the main algorithm is to perform incremental binding by delaying interconnect style decisions until all bindings are set. The decisions made about the interconnect are tentative unless they are forced by the user. The connections between functional modules and registers are implicitly set by bindings but the way multiplexers and bus drivers are used is not finalized until all operations and values are bound to functional modules and registers.

During incremental binding decisions, MABAL calculates the cost of the possible bindings for an operation and chooses the best binding with respect to the partial design and tentative interconnect already in place.

3.4.2 MABAL Algorithm

The algorithm used by MABAL does not differentiate between operations and values requiring storage, between functional modules and registers, or between resource and register allocation. Any value needing storage will be assigned a storage-operation. In the case of the pipelined design style, more than one instances of a value may be alive simultaneously. Then, each of these instances are assigned a separate storage-operation. Therefore, the terms operation, hardware module and resource allocation will be used in a broader sense to cover functional and storage-operation, functional module and register, and functional module allocation and register allocation, respectively.

The main algorithm tries to minimize total cost by trading-off between module cost and interconnect logic cost for each binding decision. In order to make each binding decision, the algorithm trades off between multiplexers and bus drivers and calculates the interconnect cost introduced at each binding step. The binding cost of an operation to a module is calculated by summing the cost of additional interconnect which must exist for each port of the module. The binding cost of an operation to a module is also calculated multiple times by permuting the commutative inputs if the operation is commutative.

A pseudo language has been used below to illustrate the fundamentals of the MABAL algorithm where

- i is an operation node
- $m(i)$ is a hardware type which will be used to implement operation i.
- $m(i)^j$ is the jth instance of type $m(i)$.

- *muxcost* is the incremental cost of having a multiplexed connection given the existing partial design.
- *buscost* is the incremental cost of having a bussed connection given the existing partial design.
- *ic* is the incremental cost of an individual binding decision.
- *generosityfactor* is the degree to which MABAL uses more hardware modules rather than interconnect.

Bindtominimizecost()

Begin

Sort all i according to their time steps and within that order according to
precedence order in the dataflow graph.

For each i in the sorted order

While TRUE

For each $m(i)^j$ already existing in the design

If i is not bindable to $m(i)^j$

Continue

else

Calculate the binding cost of i to $m(i)^j$

endif

If min $(Bindingcostof(i\,to\,some\,m(i)^j)) \geq \left(\frac{module\ cost}{generosity\ factor}\right)$

then

/* This may be done at most once for each i */

Allocate a new module of type $m(i)$

Continue

else

Tentatively choose connection types for $m(i)^j$

Make a permanent decision to bind i to $m(i)^j$

Break

endif

If i has not been bound to any $m(i)^j$ by the algorithm

then

If there is manual binding for the operation

then

Remove the manual binding

Issue a warning to the user

Continue

else

Allocate a new module of type $m(i)$

Continue

End While

For each $m(i)^j$

Choose final connection types given bindings of all operations

Try to merge buses which do have conflicts

Discard unused pre-allocated modules

Remove unnecessary bus-drivers

End.

The binding cost of an operation to a module can be calculated by summing the cost of additional interconnect which must exist for each port of the module. A part of the algorithm which decides how the module should be connected to the rest of the design given some already existing bindings and a new candidate for binding can be outlined as follows:

Bindingcostof(i **to** $m(i)^j$)

Begin

For each permutation of commutable inputs

  $ic = 0$

  For each input port k of $m(i)^j$

    calculate *muxcost*, *buscost*

    If $muxcost < buscost$ **or** there is a bus conflict

      then

        choose multiplexed connection

        $ic = ic + muxcost$

      else

        choose bussed connection

        $ic = ic + buscost$

    endif

  Record the permutation corresponding to the minimum ic

For all output ports of $m(i)^j$

  A similar algorithm applies

End.

3.4.2.1 Example

In this section, we will show how the algorithm works on an example. Assume that the partial design shown in Figure 3.3 exists and the algorithm is to decide about the binding decision of an operation whose type is *foo*. We will only consider connections for the input port. Assume that the operation has an input which is coming conditionally from either R1 or R2. The algorithm must find the best binding for the operation as well as the necessary interconnect from R1 and R2. This is done by calculating binding costs for a number of possible cases as shown in Figures 3.4 through 3.9.

The multiplexers are assumed to be built as trees of 2-to-1 multiplexers and multiplexer costs are calculated based on this assumption. A_{mux}, A_{bd}, A_{foo} are the costs of a 2-to-1 multiplexer, a bus-driver and a functional module foo, respectively.

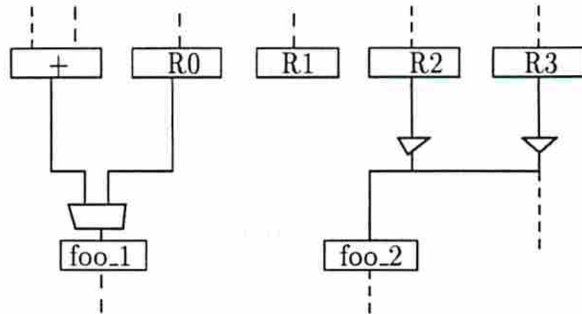
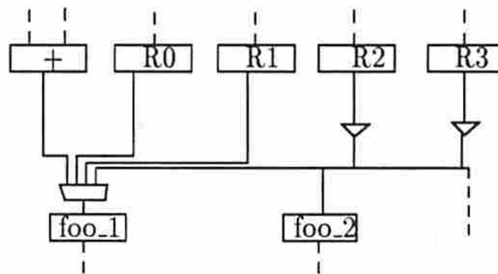
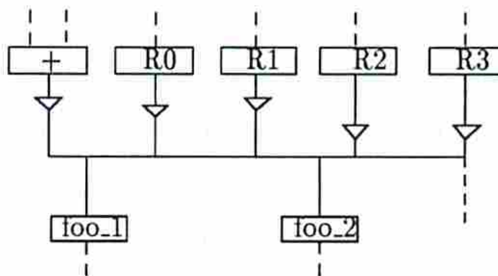


Figure 3.3: An existing partial design



Incremental Cost $= 2 * Amux$

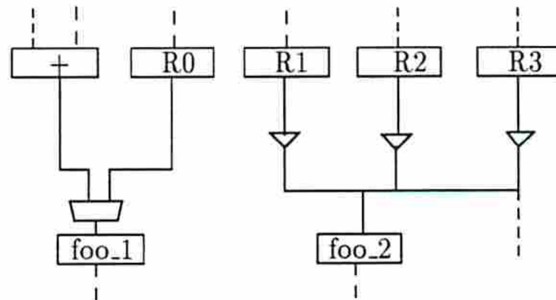
Figure 3.4: Adding two multiplexer inputs



Incremental Cost $= 3 * A_{bd} - A_{mux}$

Figure 3.5: Converting to a single bus

Since there are two existing modules which may perform the new operation, the algorithm checks the binding costs of these cases by calculating the incremental cost needed for each binding. In Figure 3.4, the binding of the new operation to module foo_1 is considered keeping the multiplexed input. Since there are no paths from R1 and R2 to foo_1, two 2-to-1 multiplexers are needed for the interconnection resulting in an incremental cost of $2 * A_{mux}$. If a bussed input is considered for the same binding as shown in Figure 3.5, the incremental cost is $3 * A_{bd} - A_{mux}$ based on the fact that there was a 2-to-1 multiplexer at the input of foo_1 already accounted for. Figures 3.6 and 3.7 show two more possible cases if the binding is to be done to module foo_2.

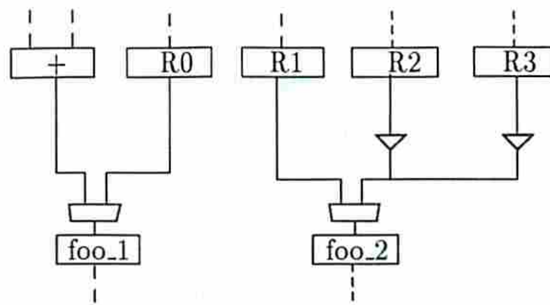


Incremental Cost = A_{bd}

Figure 3.6: Adding a bus driver

The algorithm checks the existing partial design for possible binding options and the binding costs are calculated for all candidate modules, unless there is a time conflict or there is a manual binding for the operation. For the existing partial design in Figure 3.3, the algorithm calculates the binding costs shown in Figures 3.4 through 3.7, if we assume that there are no manual bindings or time conflicts. If any of the bussed input connections create bus conflicts, then multiplexed inputs are considered for each case with conflicts.

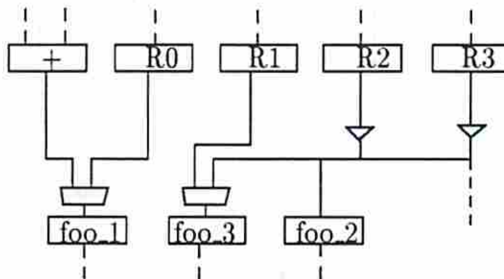
Figures 3.8 and 3.9 show the cases if the algorithm is to add module foo_3 to the design as a candidate for binding. If adding another module is feasible, then the algorithm adds another module of the same type and calculates the binding cost. Once a choice is made, each operator binding will be final but



Incremental Cost = A_{mux}

Figure 3.7: Adding a multiplexer

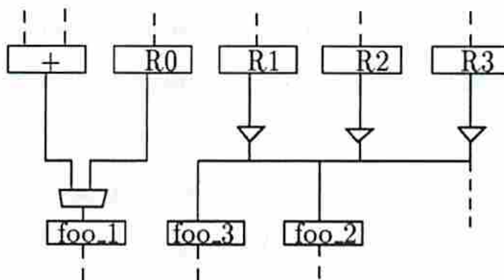
the interconnect will evolve by means of successive binding operations. The interconnect style will be finalized using the same approach after all operations are bound.



Incremental Cost = $A_{mux} + A_{foo}$

Figure 3.8: Adding a functional module and a multiplexer

One of these 6 cases will be selected depending on the minimum incremental cost, which is totally dependent on the library.



Incremental Cost = $A_{bd} + A_{foo}$

Figure 3.9: Adding a functional module and a bus driver

3.5 Optimality Considerations

MABAL is guaranteed to obtain a resource allocation with the minimum possible number of operators for unconditional graphs. For conditional graphs, the detection of mutually exclusive operations is performed with color codes given to MABAL. These color codes are generated according to Park's model [PP88]. More discussion on the detection of mutual exclusion can be found in Section 3.8.

When functional module costs are comparable to interconnect costs and the user does not require the minimum resource allocation, MABAL has yielded designs with non-minimum resource allocation and total cost less than designs with minimum resource allocation.

The MABAL algorithm treats all functional operations and values needing storage (storage-operations) alike. If one examines the binding algorithm with minimum resource requirement (generosity factor = 0) and no manual initial register allocation, it can be seen easily that the binding procedure for values is similar to the left edge track assignment algorithm [HS71]. The algorithm starts with no registers in place and does not introduce any new registers unless it has to. The ordering of value bindings is according to their birth times. In each value binding, a choice is made among possible registers based on the incremental interconnect cost. The effect of the algorithm used in MABAL is similar to the REAL [KP87] algorithm for register allocation purposes. REAL uses the left edge algorithm for register allocation. This approach is known to give optimal results only with respect to the number of registers needed, if the dataflow graph is unconditional and is free of loops but, it does not consider any interconnect issues resulting from register sharing. The main difference between REAL and MABAL for register allocation is that the REAL algorithm finds value assignments for a register (one register at a time), while on the other hand, MABAL finds the register assignment of a value (one value at a time). Although both methods have found the same register allocation (value assignments differ) in our past

experience, it is not proven yet if MABAL can always produce optimal register allocation.

3.6 Complexity Considerations

The data preparation phase which includes derivation of storage operations, levelizing the graph and sorting operations has $\mathcal{O}(n^2)$ complexity based on the assumption that the number of edges in the dataflow graph is $\mathcal{O}(n^2)$ where n is the total number of operations. MABAL performs local searches and the run-time is dependent on the design being synthesized. Because of the dynamic features of the local search method used in MABAL, it is hard to find a tight bound for the run-time complexity. Therefore, only an approximated run-time complexity analysis will be presented. The following assumptions are made to calculate the approximated run-time complexity shown in Equation 3.5:

1. The number of commutativity cases considered for each operation is assumed to be constant. The number of cases considered is usually two or less.

2. The local searches MABAL performs to find connectivity of operations, modules, and buses are assumed to take constant time. In extreme cases (highly serial designs), this assumption may not hold, but on the average, assuming a relatively large constant is enough.

The approximated run-time complexity of the MABAL algorithm is then given as

$$\mathcal{O}(n^2 + \sum_i (n_i \times o_i \times p_i)) \tag{3.5}$$

where n is $\sum_i n_i$, n_i is the number of operations of type i (including storage-operations), o_i is the number of operators of type i (including registers), and p_i is the number of ports of operation type i.

3.6.1 Experimental Complexity Analysis

An experimental run-time complexity analysis was performed using 40 of the pipelined and non-pipelined designs produced for the AR and FIR Filters. The run-times were measured and compared against the approximated run-time complexity of the basic algorithm excluding pre-processing which is $\mathcal{O}(\sum_i n_i \times o_i \times p_i)$. The data preparation phase is excluded in either case. It has to be noted again that values needing storage are modeled as storage operations and they contribute to the complexity in the same way as regular operations.

Results are organized into normalized 2-tuples (the approximated-complexity and the actual run-time). Then, these 2-tuples are ordered by the approximated complexity. The results are shown in Figure 3.10. The continuous line shows the actual run-time information and the approximated complexity is shown by asterisks.

Major deviations of actual run-times from the approximated complexity have been seen for highly parallel designs. Although, there are many choices for module binding decisions for such designs, most of these choices are pruned as soon as it is detected that resource sharing is not possible.

3.7 Experimental Results

In this section a number of examples from the literature will used to compare the results obtained from MABAL to other published results. The library and some of dataflow graphs are given in Chapter 1. Comparing results for different data path synthesis approaches is generally a hard task since different assumptions are made by each approach during the synthesis process and detailed information about these assumptions and/or partial results are not always available. Another problem is that the comparison is generally performed at RT-Level which does not provide enough detail about the final layout. Some results which may seem

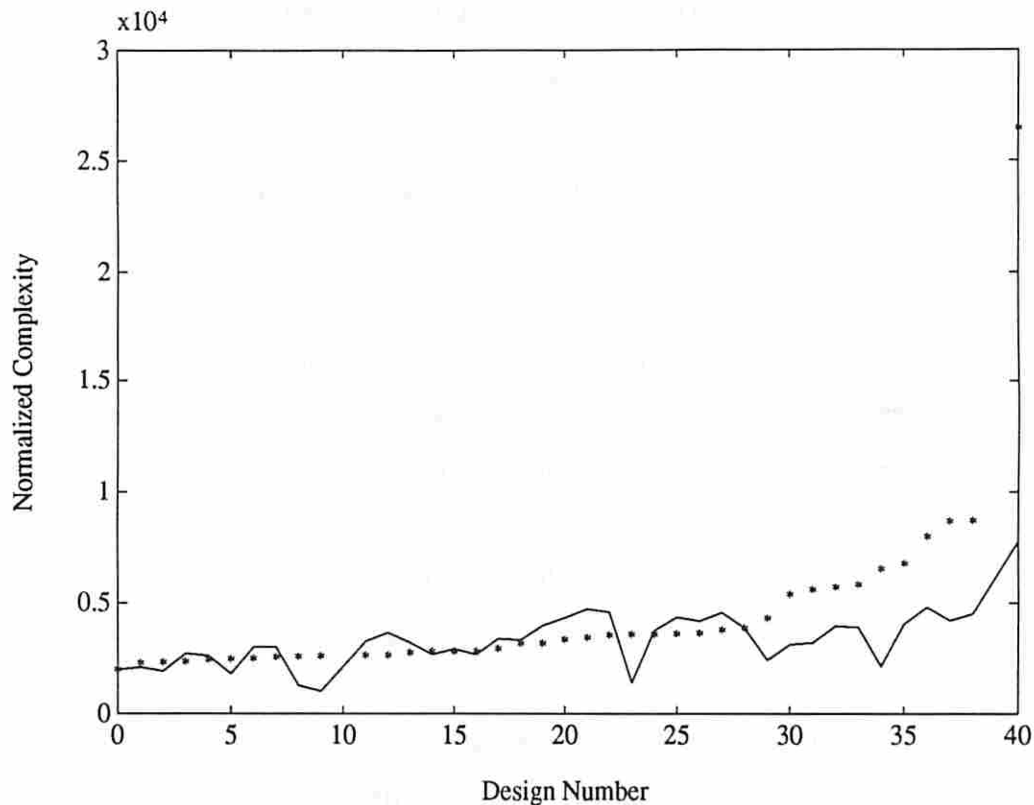


Figure 3.10: Run-time Complexity Analysis for MABAL

to be inferior at RT-Level may turn out to be better than others when layouts are actually produced.

In these experiments, the library shown in Table 1.1 with module library configuration (add3, mul2) is used for RTL area calculations. The reason for selecting this library configuration is that no single module characteristic (e.g. the multiplier area) can dominate the total area so that the comparisons are as fair as possible. We are aware of the fact that other data path synthesis tools with our library and our tool with their library could produce relatively better or worse results, but there is currently no way of obtaining such data. Reported CPU times are in seconds and MABAL CPU times are measured on a Sun Sparc 4/330.

3.7.1 AR Filter

To our knowledge, there has been no published work using the AR Filter and is comparable to our work. In this section, both pipelined and non-pipelined implementations of the AR Filter are given in Tables 3.1 and 3.2. Sehwa and MAHA schedules have been used for pipelined and non-pipelined implementations respectively. In MAHA schedules the clock cycle time is not fixed. Highly parallel MAHA schedules have more operation chaining, hence considerably longer clock cycle times. It is assumed that all inputs are already stored for the duration of the initiation interval (which means none of the inputs will be stored for non-pipelined implementations). The results are not necessarily minimum resource designs (especially for registers). MABAL was run in such a mode to directly select the cheaper decision of adding new functional resources or adding interconnect at each binding step. There was no bias to share functional resources or not to share.

II	N	+	*	Register	2:1 Mux	RTL Area	CPU Time
1	6	12	16	80	0	210880	1.17
2	7	6	8	30	22	106816	0.56
3	7	4	6	16	33	81040	0.48
4	10	3	4	30	33	67184	0.83
5	11	3	4	26	33	65200	0.70
6	14	2	3	24	42	55800	0.69
7	8	2	3	7	37	45928	0.46
8	12	2	2	7	35	35552	0.51
12	13	1	2	6	28	31840	0.68
16	19	1	1	8	37	25624	0.76

II Initiation Interval (number of clock cycles)
N Pipeline Length (number of clock cycles)

Table 3.1: Pipelined AR Filter Results

The quality of the scheduling is very important not only for the resource allocation but also for the interconnect allocation. The schedules created by MAHA and Sehwa were used by MABAL without changing the schedules except one MAHA schedule. Both MAHA and Sehwa do not consider the effect of scheduling on the interconnect. The way the time steps are created highly affects the resulting interconnect. Keeping repetitive patterns in the schedule without causing time conflicts is highly beneficial to save interconnect cost by simply sharing the wires. This is performed in the 8-step manual schedule shown in Table 3.2. In this case, paying special attention to repetitive patterns pays more than the serial/parallel tradeoff. The 8-step RTL design is smaller than the 13-step design which is more than 50% slower.

N	+	*	Register	2:1 Mux	RTL Area	CPU Time
1	12	16	2	0	172192	0.13
3	4	6	4	28	73648	0.34
4	4	4	5	33	55984	0.32
5	3	4	6	35	55856	0.41
6	3	3	5	37	46136	0.37
8†	2	2	5	28	32544	0.36
9	2	2	5	37	35136	0.47
13	1	2	7	37	34928	0.57
17	2	1	6	35	25256	0.67
18	1	1	6	32	23192	0.80

N Number of time steps
† Manual schedule

Table 3.2: Non-pipelined AR Filter Results

3.7.2 FIR Filter

The first set of comparisons will be performed against the results shown in [HCCd89]. Table 3.3 shows the comparisons of MABAL results and results

reported in [HCCd89]. These results are for a pipelined implementation of the FIR Filter with data initiation rate of 2 clock cycles. The schedule is taken from [HCCd89]. It is assumed that all operations take 1 clock cycle to execute. MABAL was instructed not to use bus drivers in order to generate comparable results. One difference in assumptions was how primary inputs were stored. In [HCCd89], it is assumed that inputs were already stored for 1 clock cycle, on the other hand, MABAL can be instructed to assume either the inputs are not stored at all or all inputs are assumed to be stored for the data initiation interval (which is 2 clock cycles for this example). This assumption changes the register and multiplexer allocation. We include all comparisons to be fair. The reported CPU time for [HCCd89] was for Sun-based Common LISP.

	*	+	Register	2:1 Mux	RTL Area	CPU Time
[HCCd89] - Case 2	4	8	42	20	75392	138.00
MABAL - Case 1	4	8	20	22	65056	0.27
MABAL - Case 1	4	8	23	19	65680	0.35
MABAL - Case 3	4	8	44	24	77536	0.67
MABAL - Case 3	4	8	48	19	78080	0.77

Case 1: Inputs were already stored for the first 2 clock cycles
Case 2: Inputs were already stored for the first clock cycle
Case 3: Inputs were not stored at all

Table 3.3: Comparisons for Pipelined FIR Filter

Another comparison of MABAL results using the FIR filter is given in section 3.7.4.

3.7.3 Differential Equation Solver

The differential equation solver is a relatively small description. The description, schedule and other specifications except the library configuration are taken from [PKG86]. All inputs are assumed to be already stored. Reported CPU

times for HAL and Splicer are given on a Xerox 1109 Lisp Machine and Sun 3/260 respectively. HAL's CPU time includes scheduling time. The reported run time for ARYL is on a VAX-11/8550.

	*	+	-	<	Reg	2:1 Mux	RTL Area	CPU Time
HAL [PKG86]	2	1	1	1	6	7	28480	140.00
Splicer [Pan88]	2	1	1	1	6	6	28192	1245.00
ARYL [HCLH90]	2	1	1	1	5	5	27120	0.04
MABAL	2	1	1	1	5	7	27696	0.10

Table 3.4: Comparisons for Differential Equation Solver

3.7.4 Importing Other Synthesis Methods into MABAL

Since MABAL is capable of building onto partially specified designs, it is possible to import other allocation and binding algorithms (with some restrictions).

In this section, an example of importing an outside operation assignment approach into MABAL will be given. Park and Kurdahi proposed a method for operation assignment and interconnect sharing [PK89a]. In this approach, all compatible paths (ordered lists functional operations and storage-operations which obey certain resource usage pattern) are extracted from the scheduled dataflow graph and a clique partitioning formulation of the problem is constructed for sharing the interconnect.

In [PK89a], the authors only list the operation assignments of functional operations for an already scheduled FIR Filter design. Register assignments for the values were not listed. The operation assignments in [PK89a] were simply specified to MABAL as pre-existing operation assignments for the functional operations. The scheduling is also taken from [PK89a]. The allocation of registers, the assignment of values to registers, and the interconnect allocation were left to MABAL.

Table 3.5 shows the comparisons of their results and MABAL results. The last row in for MABAL results in Table 3.5 were obtained without importing any operation bindings, but using the same schedule.

	Adder	Multiplier	Register	2:1 Multiplexer	RTL Area
[PK89a]	5	3	18	23	50952
MABAL †	5	3	15	23	49464
MABAL †	5	3	14	25	49544
MABAL ‡	5	3	12	33	50856

† with imported operation assignments
‡ without imported operation assignments

Table 3.5: Comparisons of results from [PK89a] and MABAL

3.8 Limitations of MABAL

In order to be compatible with the rest of the ADAM System, some assumptions were made during the design phase of the tool.

The architecture style assumed by MABAL only allows single-cycle operations/operators. Multi-cycle or pipelined operators are not allowed. This is one of the major assumptions in the ADAM Synthesis tools. This assumption is not too restrictive to search the design space because operator chaining is allowed and the clock cycle time is decided by scheduling programs to best fit the schedule. But, it occasionally causes the schedules generated by Sehwa or MAHA to be inferior.

MABAL does not handle dataflow graphs with inner loops. MABAL is actually a synthesis tool to generate pipelined data paths. It handles non-pipelined designs as a special case (the initiation interval equals the number of time steps).

Using the pipelining model, it is hard to deal with inner loops since multiple instances of the dataflow graph are executed simultaneously. Therefore, the following assumption is made for the overall design process. The inner loops are either unrolled as proposed by [PP88] or inner loops will be designed in a bottom-up fashion.

The conditional information about the dataflow graph is currently included in the dataflow graph using *Distribute-Join* node pairs as described by Park [PP88]. This model, which is being used by Sehwa, MAHA, MABAL, and CSG has its restrictions. Problems arise when conditional constructs from hardware description languages are translated into single-assignment dataflow graph representations. The Design Data Structure, DDS [KP85] has separate dataflow and control flow graphs with bindings relating dataflow and control flow objects with no representation difficulty. Unfortunately, the implementation of DDS on an accessible database happened after most data path synthesis software had been written. Therefore, conditional information may not be utilized to its fullest extent with current tools in ADAM System.

A limitation of MABAL which is not due to assumptions in the ADAM System is the interconnect style. In a general design, the interconnection hardware can consist of multi-level multiplexing. MABAL only produces one-level abstract multiplexing which can be also nested with a bus. In another words, MABAL does not share multiplexer outputs. It is preferable to share multiplexer outputs to save RTL area. But, such optimizations have to be performed in a such a way that the least possible delays should be introduced into the critical path of the design by such optimizations. Sharing multiplexer outputs may also increase the routing complexity which, in turn, may increase the routing area and wire delays. To our knowledge, there is no in-depth study yet to show whether sharing multiplexer outputs is better or not in general.

3.9 A Data Path Tradeoff Study

Library		1		2	3
Module	Bitwidth	Area	Delay ns	Area	Area
adder	8	13350	17.0	13350	5000
subtractor	8	13350	17.0	13350	5000
2-to-1 mux	1	370	3.7	370	370
bus driver	1	250	2.4	200	250
register	1	1106	5.0	1106	350
register	8	8850	5.0	8850	2800
Geq	8	8710	16.3	8710	8710
Leq	8	8885	18.3	8885	8885
divide-by-shift	8	0	0.0	0	0
memoryread	8	∞*	17.0	∞	∞
* Set to force the selection of only one memory unit					

Table 3.6: Libraries used for experiments in Section 3.9

A set of experiments have been performed to find out the applicability of certain tradeoffs for varying component characteristics. A controller specification shown in Figure 3.11 has been chosen for the tradeoff study. This example has 9 additions, 4 subtractions, 4 comparisons, 2 memory-read operations, 2 divide-by-shift operations and 32 values. The example contains 3 conditional blocks and an outer loop. A non-pipelined schedule produced by MAHA[PPM86] was used as a starting point to demonstrate some design tradeoffs. This particular schedule produced by MAHA using Library 1 is shown in Figure 3.12.

Three closely related libraries which have been used during the experiment are shown in Table 3.6. The original library (Library 1) was derived from the Texas Instruments 2 micron CMOS Standard Cell library [TI86]. The areas reported in Table 3.6 are 100 times the relative area to a 2-input nand gate. Library 2 is the same as the first library except that the cost of the bus driver has been deliberately decreased. Library 3 has the same costs for data steering modules as the first library but it has some smaller, slower functional modules. It would be

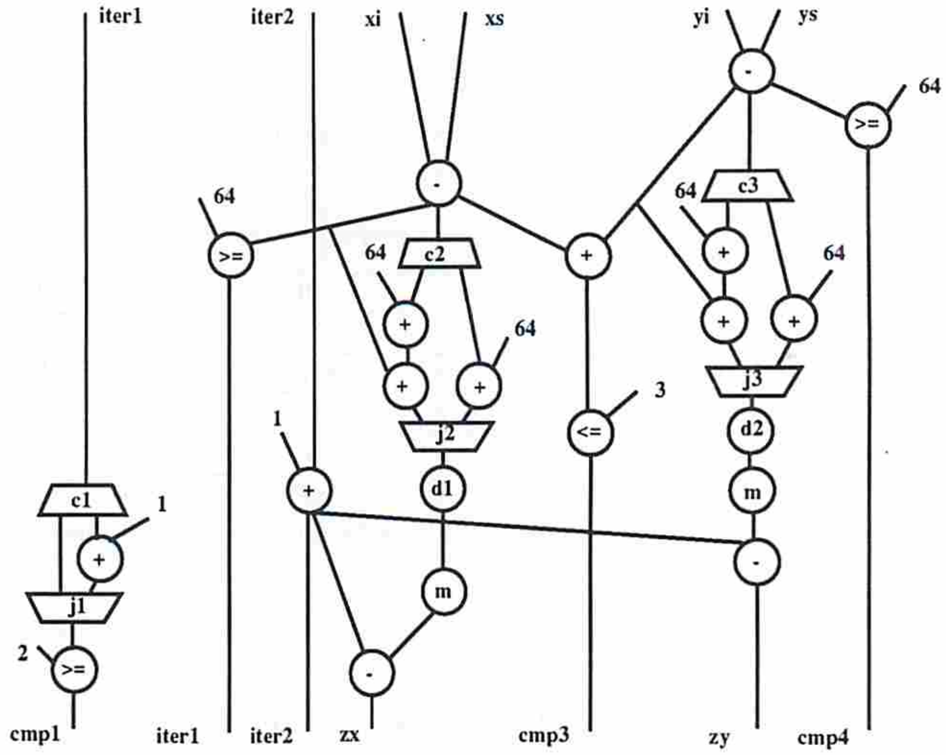


Figure 3.11: The dataflow graph used in the tradeoff experiment

unreasonable to assume that slower modules could be used without changing the delay characteristics of the design. It is obvious that the design will be slower if library 3 is used instead of library 1, if the same schedule is used. The schedule of operations was assumed to be fixed, but even minor variations in module sets might have produced different schedules. However, our primary goal here is to demonstrate the concept of how well variations of component characteristics in a library can be utilized by a data path synthesis algorithm which is sensitive to the technology. Thus, we will use the same schedule for all libraries.

The specific purpose of the experiments was to observe how varying the relative costs of modules affected tradeoffs in a data path implementation. Two particular studies focused on answering the following questions.

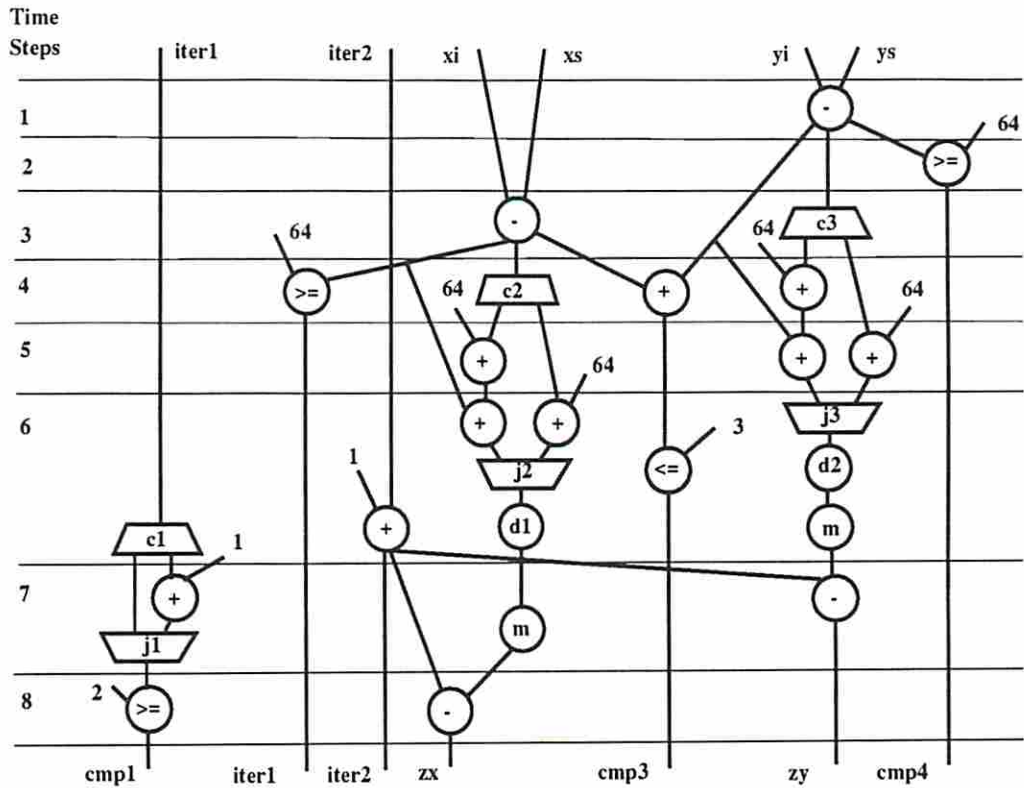


Figure 3.12: The schedule produced by MAHA for the dataflow graph shown in Figure 3.11

1. When should values be bussed and when multiplexed ?
2. When should functional resources be duplicated ?

3.9.1 Results and Discussion

In general, for designs which have little sharing, multiplexing is most cost effective, while for heavily shared designs, buses are cheaper. However, the point where the tradeoff between multiplexers and buses is made shifts depending on the relative costs of both types of modules.

A lower bound on functional resource allocation (optimality) is achieved when all resources are shared to the fullest extent possible, given the schedule. An upper bound is achieved by assigning a resource for every dataflow graph operation, independent of potential sharing. Between these two bounds, a number of designs exist, with varying bit-steering logic, depending on the amount of sharing.

The capability of trading between multiplexers and bus drivers, and between the interconnection units and functional modules may be very important when the costs of multiplexers, bus drivers and some functional modules are comparable. For library 3, the costs of modules were adjusted such that costs of additional adders, subtractors and registers were slightly higher than the worst-case multiplexer costs to share them. While this particular situation may not be realistic for arithmetic operations, sometimes small components with these characteristics are used. For example, the designs encountered in some communication applications have coding and non-arithmetic processing, and include simpler modules.

A total of 3241 designs were generated by varying initial resource allocation, library and program parameters. Not all the designs generated are unique; some designs have the same RTL structure although the bindings for operations and values differ. It took less than a day to generate the designs and to extract the data[3.4]. Results obtained from tradeoffs between bit-steering and functional resources show that there are literally thousands of good designs possible.

Results obtained indicate that the decision to trade off between buses and multiplexer steering logic is quite sensitive to the relative costs of modules employed. As is seen from Table 3.7, bus drivers are hardly used in the designs generated from library 1. But, when a 20% cheaper bus driver in library 2 is substituted, bus driver use increases drastically. It is seen from Table 3.7 that the use of buses is also related to the resource sharing. If minimum resource allocation is requested, buses are more likely to be created.

[3.4]Human analysis of the extracted results, of course, took somewhat longer.

Minimum resource	Library	Designs with buses	Total # of designs
Required	1	46	160
Not required	1	4	160
Required	2	153	158
Not required	2	118	150
Not required	3	1	1141

Table 3.7: Statistical Data on Bus (bus driver) Usage

Although, it may seem plausible to use as many functional modules as possible in the designs using library 3, the resulting designs actually have a functional area which is less than this upper-bound functional area. This is mainly due to the fact that it is sometimes possible to share hardware without creating any interconnect cost (sharing the interconnect already in place). It is also seen that there is only one design with buses in the 1141 designs generated from library 3. When sharing a resource costs as much as the shared resource itself, resource sharing obviously tends to be lower, making buses less desirable. This singular design was intentionally generated by the manipulation of user settable parameters.

When Figures 3.13, 3.14 and 3.15 are examined, the inverse relation between steering logic area and functional area can be easily seen. As one area component of the RTL area increases, the other decreases, resulting in many designs with comparable RTL areas. The designs shown with asterisks in Figures 3.13 through 3.15 are larger than others. They are included to show the bottom-right part of the design space which has been searched.

From the designs we generated from each library, we observed some general characteristics. We generated several designs varying functional resource allocation and the program parameters. Repeating this process for each library, we plotted design points which are non-inferior on the basis of functional area versus steering logic area. The distributions obtained (Figures 3.13 through 3.15) are

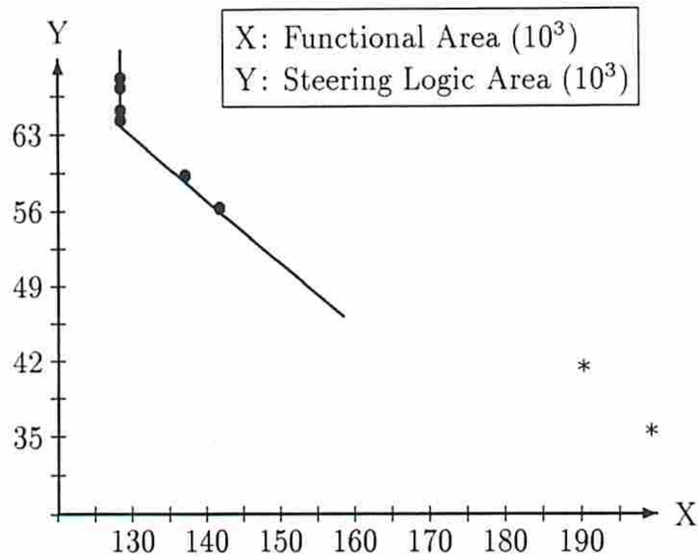


Figure 3.13: Selected designs generated using library 1

roughly similar to Figure 3.16. Actually, there are a family of designs close to each design point plotted. There also a few exceptionally good designs in the upper-left part of the design space.

Figure 3.16 tells us a number of important things. The line segment between points 1 and 2 corresponds to designs with the minimum possible steering logic area (which can be non-zero in cases like having to share memory address lines, or a value conditionally coming from multiple sources). All designs with minimum functional resource allocation lie on the line segment from point 3 and point 4. We are not generally interested any designs to the right of point 3 or above point 2 because they are more likely to be inferior. The design points between points 2 and 3 represent designs with roughly equal RTL area but different architectures. The number of designs on this line segment as well as the slope of the line segment (or the shape of the curve) can vary greatly depending on the characteristics of the library, the dataflow graph and the schedule. If no tradeoff is feasible, then this line segment shrinks to a single point. The larger the

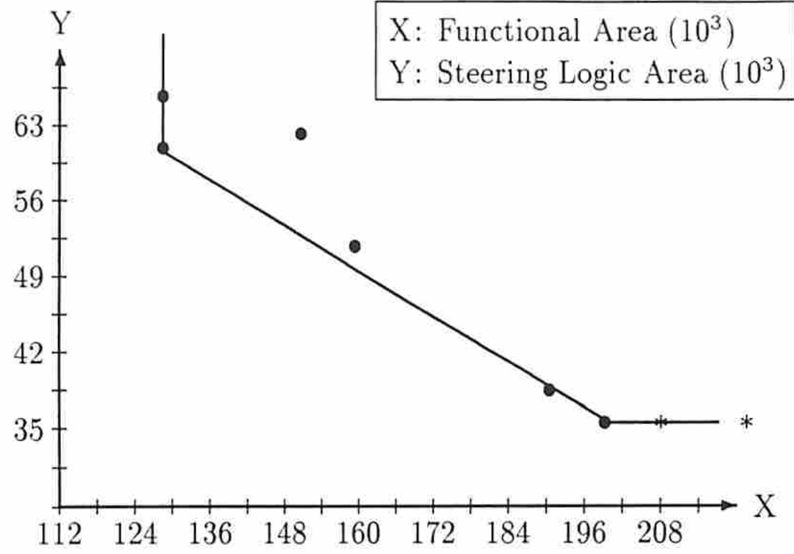


Figure 3.14: Selected designs generated using library 2

number of *tradable*[3.5] library modules and, the larger the number of operations implemented by those library modules, the higher the likelihood of having more designs in this region (ignoring the fact that some potential designs will coincide). The general approach in the synthesis community has been to generate designs in close proximity to point 3 which corresponds to minimum functional resource allocation.

Manual placement and routing of two selected RTL designs (one design which is close to point 2, another which is close to point 3) has shown that designs in close proximity to point 2 are more likely to be suitable for custom placement and routing, while on the other hand, designs which are highly shared might not be. RTL designs with minimum and non-minimum resource allocations are shown in Figures 3.17 and 3.18, respectively. The final differences between areas of designs will be due to floorplanning and routing differences which are highly

[3.5] a *tradable* module means its area is close to the area of steering logic modules needed to share the module.

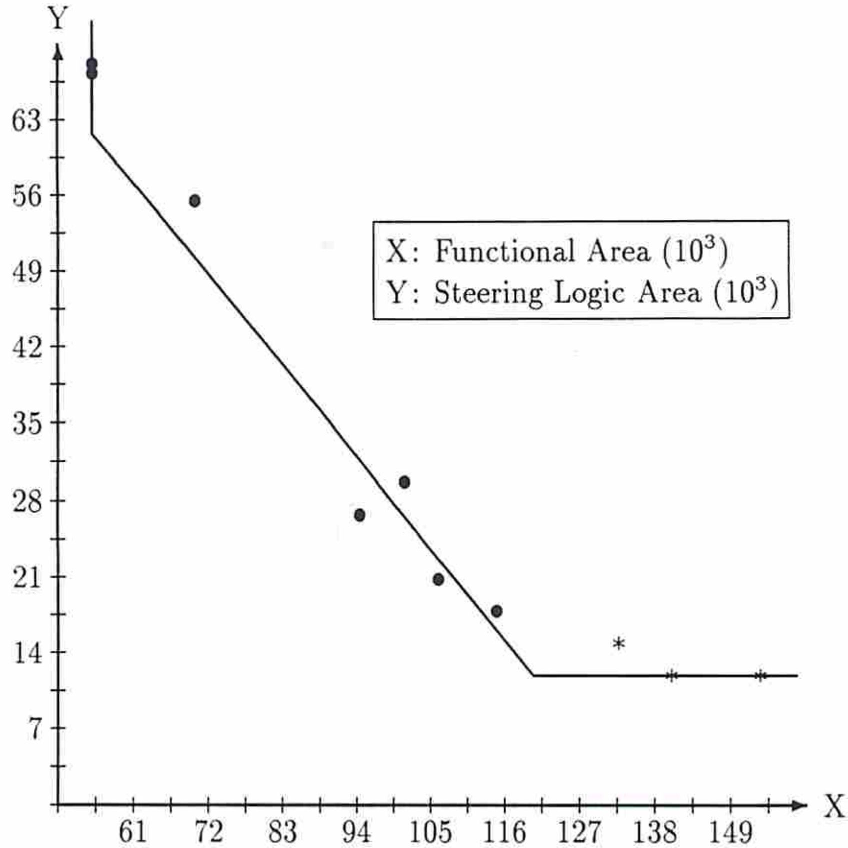


Figure 3.15: Selected designs generated using library 3

sensitive to the architectures generated. Although, floorplanning is ignored by most synthesis programs (some exceptions are [McF86, WP91]), it is the only criterion which can be used to determine the best design among the designs between points 2 and 3.

Wiring space is taken into account in the ADAM System by the PLEST predictor, which can be run immediately following MABAL. MABAL counts the equivalent number of two-point nets, which is used directly as input to PLEST, along with the RTL area. PLEST predicts the wiring space required for a standard cell layout of the design. The MABAL runs described here showed no substantial differences in two-point nets between the designs produced, except

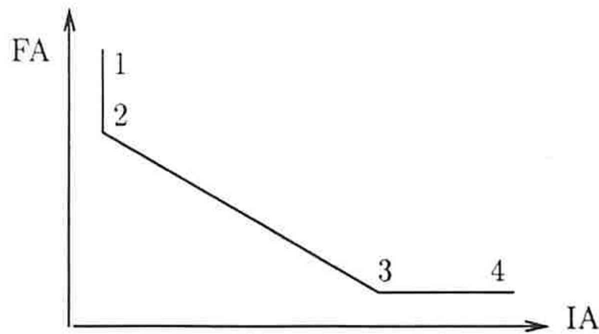


Figure 3.16: Proposed model for functional area versus steering logic area

for the nets internal to the multiplexers. The preliminary conclusion that could be drawn here is that standard cell designs with equivalent RTL area (functional modules plus data steering) and equivalent nets external to multiplexers could be expected to be of similar size, but significant use of multiplexers will result in larger designs.

To characterize the sensitivity of standard-cell layout area to the number of 2-point nets in a design, a set of experiments were performed using PLEST. In this experiment, for different RTL areas, the number of 2-point nets is varied and the estimated standard-cell layout area from PLEST is recorded. A sample of results are shown in Figure 3.19. The upper curve is for the RTL area of 300,000 mil^2 and the lower curve is for the RTL area of 100,000 mil^2. Keeping in mind that PLEST is an estimator with 10 percent accuracy, it can be easily seen that the slope of the curves in Figure 3.19 are, without any doubt, less than 1. This tells us that given any two designs with comparable number of 2-point nets, the layout area of the design with smaller RTL area is more likely to be smaller than the design with larger RTL area even though the design with larger RTL area may have fewer 2-point nets. With this result, we can expect that synthesizing designs with the objective of minimizing the RTL area and number of nets while ignoring the layout features is quite likely to be successful in producing minimal layout-area designs for the standard-cell layout style.

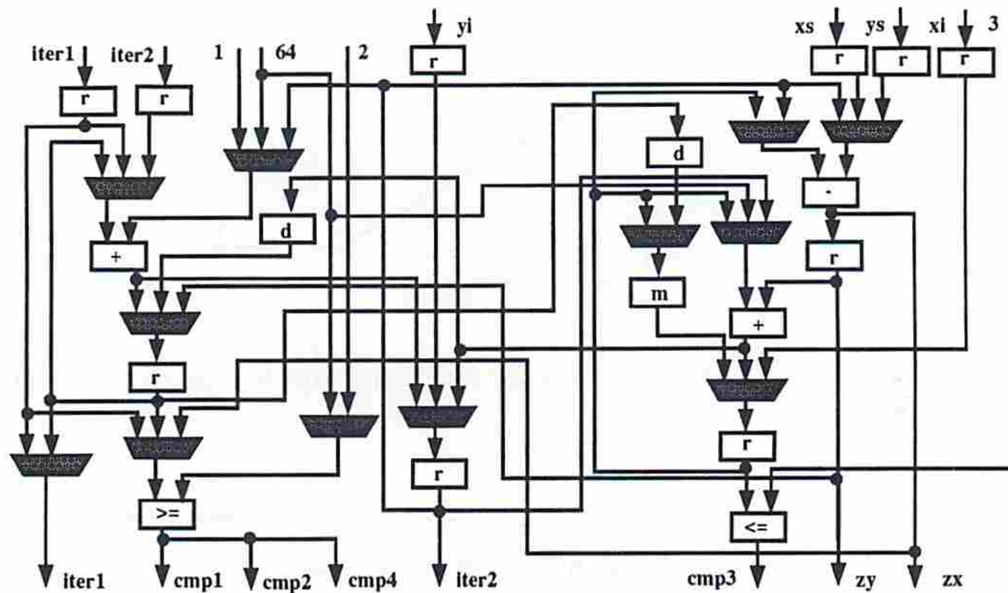


Figure 3.17: A minimal functional resource design

The effect of routing can be taken into account to some extent by incorporating the average wire length estimates into functional and steering logic modules. Since MABAL tries to minimize the overall area, the deficiency of not explicitly considering the routing area can be partially overcome. But, the explicit consideration of floor-planing during synthesis shows itself to be the most effective way to solve this problem [McF86, WP91].

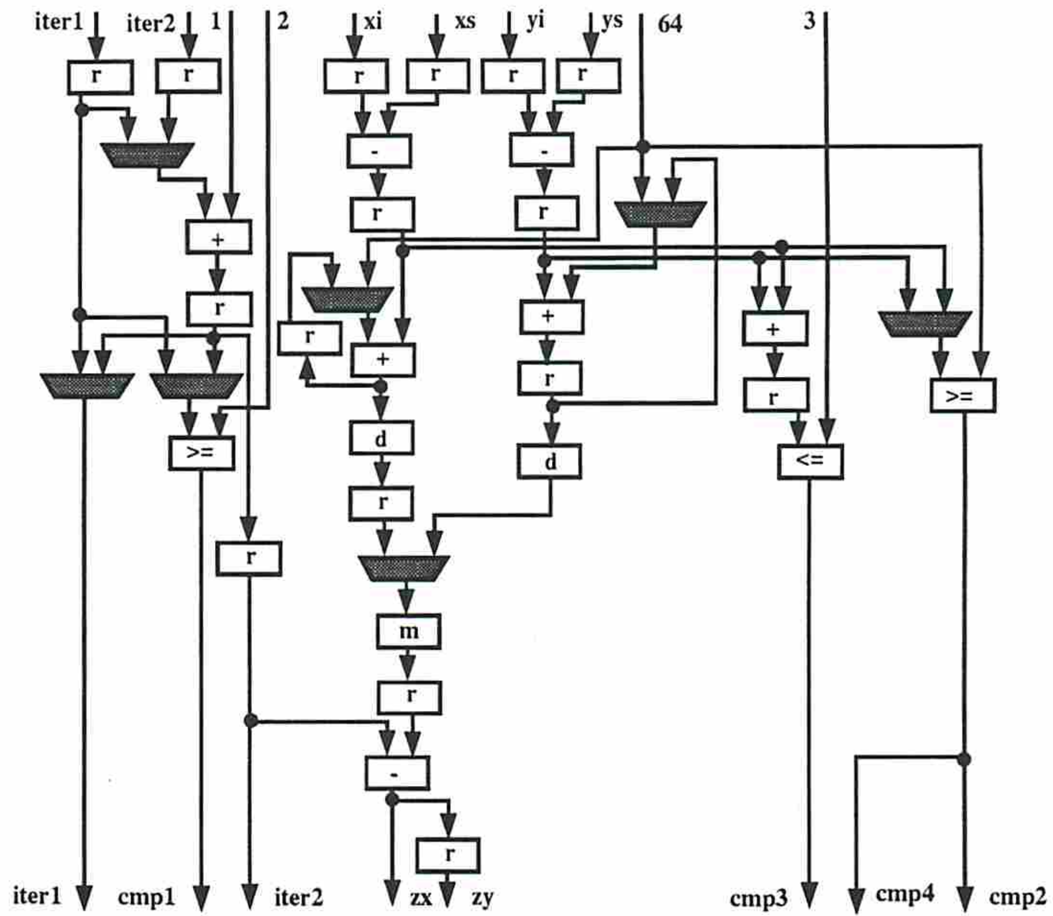


Figure 3.18: A non-minimal functional resource design

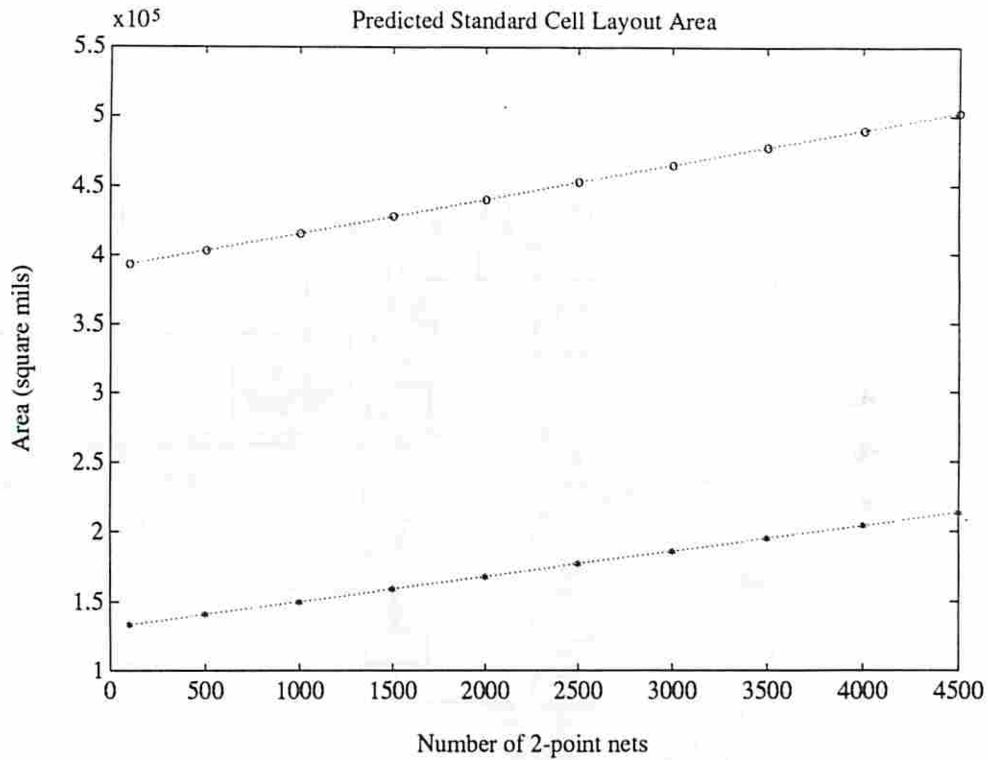


Figure 3.19: The results of the experiments with PLEST

3.10 Summary

In this chapter, MABAL, a tool which assigns operations to operators and produces the necessary interconnect for data transfers given a scheduled behavior has been presented. MABAL has shown to be very useful in quickly generating good quality RTL designs. It was also very useful in experiments to explore possible design tradeoffs with its tradeoff capabilities and speed which allowed us to search a very large design space in a short amount of time.

A set of experiments have been performed using MABAL. Broadly speaking, the results indicate that data path designs are sensitive to variations in the module library. This leads us to believe module selection should precede data path

synthesis. Furthermore, if module generators, logic synthesis or logic optimization are employed, their interaction with data path synthesis procedures must be studied carefully. Synthesis programs with a rigid design style (e.g. using multiplexers or buses exclusively) will not provide the tradeoffs shown here, and synthesis programs which do not tradeoff functional costs for data steering costs might miss some better designs. The lower-level synthesis process to be used has to be parameterized into the high-level synthesis process in order to produce better quality designs. It is also essential to have technology-sensitive algorithms which produce technology-dependent designs in order for a synthesis tool to be used with a variety of systems and applications.

Chapter 4

Behavioral Area-Delay Predictions

4.1 Introduction

As explained in Chapter 1, high-level synthesis is a very complicated process and even its subtasks are believed to be exponential in complexity. Because of this complexity, most high level synthesis programs have a very limited scope of the overall synthesis problem and they try to generate highly optimized results for relatively small synthesis subtasks. The fact that the design space to be explored is huge together with the high complexity of solving individual (and often simplified) synthesis subtasks usually results in an iterative high-level synthesis process. The basic iteration mechanism is as follows: A set of assumptions is made about the final design to be synthesized, then a sequential synthesis process is followed by making design decisions along the way. When it is found that the design decisions which have been made would not result in an acceptable design (either during the synthesis process or after the synthesis is completed), some of the assumptions and design decisions are modified and the whole synthesis process (or parts of it) is repeated.

One major difficulty which causes the synthesis process to be iterative and time consuming is the fact that some architectural assumptions made at early stages of the synthesis process do not necessarily hold when lower level details are considered. An example of this is that most scheduling approaches assume

negligible wire delays in the final design. However, designs with submicron feature sizes have wiring delays which are comparable to functional delays, a fact which can require schedules to be changed to meet constraints. This illustrates how complicated the synthesis process is; even the evaluation of most high-level design decisions generally requires a complete synthesis process.

Even though automated synthesis can speed up the design process by orders of magnitude, the growing complexity of designs together with the multitude of tradeoffs possible during synthesis make it impossible to completely search the design space, even with synthesis tools which span behavior to layout. In order to improve the quality of generated designs, the designer has to iterate several times by modifying the functional specification (high-level transformations), design constraints and design decisions. This iterative process can be quite time consuming. Fast prediction tools to estimate the impact of tentative design decisions while taking into account physical design effects can be extremely beneficial in reducing the iteration time.

4.1.1 Overview

This chapter presents a comprehensive, integrated area-delay prediction methodology which can be used to support behavioral synthesis tools. Each of these methods represents an improvement over previous prediction methods in terms of breadth, performance or accuracy. The prediction model includes functional, register, multiplexer, wiring and PLA area and delay predictions. The predictions are based solely on the behavior of the target design, a design library and on its area-performance constraints. These methods together with a unified statistical representation of prediction results can be used effectively for a variety of purposes including guiding system-level decisions, guiding a behavioral synthesis planner, quickly searching the design space and evaluating transformations.

The input for BAD, the prediction tool presented here, consists of the tentative clock cycle time, a directed acyclic dataflow graph, an operator library

(preferably with more than one operator style to implement every operation type) and constraints for total area, initiation interval and total circuit delay. The prediction results are in the form of a set of area-delay pairs for non-pipelined designs and area-delay-initiation interval triples for pipelined designs. Each pair or triple forms a point (predicted design) in the predicted design space.

BAD generates predicted designs for all possible meaningful implementations of the design. It simply enumerates all possible library configurations (each library configuration contains one operator per operation type). For each library configuration, pipelined and non-pipelined predictions are generated for each possible value of initiation interval and number of stages (in terms of clock cycles). If it is preferred not to have any pipelined data path predictions, BAD can be instructed to only produce non-pipelined data path predictions. The selection of the best feasible predicted design (satisfying design constraints) from the pool of predicted designs also solves the problem of design style selection and module selection.

BAD is also capable of deleting predicted designs which are inferior or cannot meet specified designer requirements. In this way, predicted designs which would eventually be too large or too slow can be eliminated automatically. The tool handles multi-cycle operations but can also be constrained to use only single cycle operations. In the latter case, the enumeration phase for generating possible library configurations automatically screens out selection of library operators which would be too slow for the given clock cycle.

BAD is a fast, practical tool which combines former theoretical and heuristic prediction research done at USC along with new additions, using a unified representation. Statistical properties (average and variance) for each design characteristic are derived from predictions of the lower bound, upper bound, and most likely values of the design characteristic and the overall characteristics are approximated by a normal distribution, to be described in the next section.

The overall area-delay predictions are obtained by performing several prediction subtasks (corresponding to actual synthesis subtasks) in a specific order. The initial data pool (before performing any predictions) consists of the input data given to BAD (clock cycle time, dataflow graph, design library, constraints and goals). Each prediction subtask contributes some predicted characteristics of each potential design to the data pool and uses the original input data as well as prediction results existing prior to its execution.

Predictions are performed by following the actual behavioral synthesis models, without dealing with the actual synthesis details. For example, the core task of pipelined scheduling in behavioral synthesis maps the operations of the dataflow graph into time slots and onto operators, given a dataflow graph, a library configuration, and an initiation interval. At the end of the actual scheduling step, operation assignments, the functional resource allocation and the number of stages in the pipeline are easily available. But, on the other hand, the prediction methods only estimate the functional resource allocation quantities and the number of stages in the pipeline without actually dealing with details like the assignment of operations to time slots and/or operators.

In the following sections, the statistical model used to represent prediction results is presented and the prediction methods for operators, registers, multiplexers, control and wiring are discussed.

4.2 A Unified Representation of Prediction Results

Using predictions in behavioral synthesis requires special care because of the discrete nature of the problem being solved. In addition, there are many types of prediction tools, each dealing with a different aspect of a design. The fact that it is not always possible to have predictions of the same nature complicates the problem even further. Some prediction results are in the form of lower-bound

values while others might be expected values or upper-bound values. There is usually more data available than just a single expected value or a bound and it is desirable to make use of as much data as possible.

We use a statistical representation which provides a unified environment for prediction results. The PERT modeling technique [LK66] has been chosen for the prototype software to approximate predicted design characteristics. In the PERT model every predicted point consists of three values (for each design characteristic): a lower bound, a most likely value and an upper bound. The average and the variance are approximated from these three values using the PERT technique, and standard statistical methods [Rob85] can then be applied.

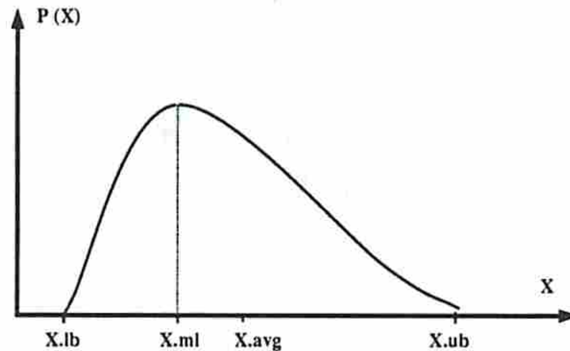


Figure 4.1: The PERT model used in predictions

Assume that PERT modeling technique is to be used for the area characteristic. *area.lb*, *area.ml*, *area.ub* are the lower-bound, most likely and upper-bound values of the characteristic, respectively. Based on these three values, *area.avg* and *area.var*, the average value for area and variance of area in the PERT model are approximated (also see Figure 4.1) as follows;

$$area.avg = \frac{area.lb + 4 \times area.ml + area.ub}{6}$$

$$area.var = \frac{(area.ub - area.lb)^2}{36}$$

The following equations show how prediction results are summed to get the total area, A:

$$\begin{aligned}
A.avg &= operator.area.avg + register.area.avg + \\
&\quad multiplexer.area.avg + control.area.avg + \\
&\quad wiring.area.avg \\
A.var &= operator.area.var + register.area.var + \\
&\quad multiplexer.area.var + control.area.var + \\
&\quad wiring.area.var
\end{aligned}$$

After the total of the prediction results for a particular design characteristic has been computed, infeasible predicted designs are eliminated as follows. In the PERT model, it is assumed that aggregate characteristics are approximated with normal distributions. The accuracy of these approximations depend on the data pool size and how well the PERT model fits the application. Given the assumption that the distribution of A can be approximated accurately by a normal distribution, then the probability of satisfying the hard area constraint A_{max} is estimated [Rob85] (see Figure 4.2):

$$Pr[A \leq A_{max}] = \int_{-\infty}^{\frac{A_{max} - A.avg}{\sqrt{A.var}}} \frac{e^{\frac{-x^2}{2}}}{\sqrt{2\pi}} dx \tag{4.1}$$

Any predicted design whose probability of satisfying a design constraint is lower than the user-specified minimum probability of feasibility for the same design constraint is eliminated. These user-specified probability figures indicate not only the severity of the constraint but also the confidence of the designer in the prediction tool. If a constraint (e.g the total circuit delay) is not very critical,

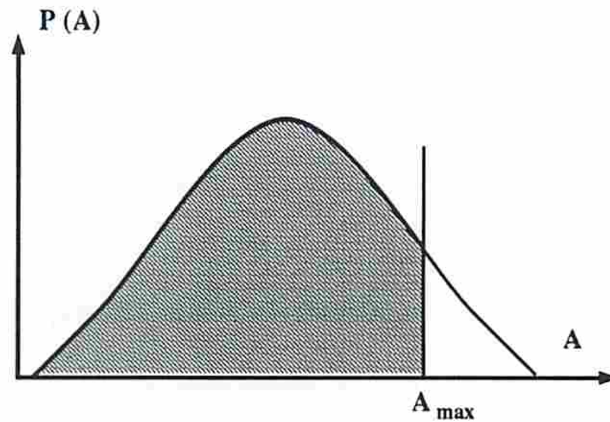


Figure 4.2: The probability of feasibility as the area under a probability density curve

then it is better to include some of designs which are predicted to violate the given constraint, but are still close to the acceptable limits. Thus the probability of satisfying a design constraint can be less than one and the design can still be accepted. Another problem is that the error in predictors may not be constant. A predictor which generally gives very accurate results may occasionally show a large deviation from the real results. Therefore, it is important not to eliminate predicted designs which violate some constraints but still stay close to the acceptable limits. The user-specified probability figure for each constraint is used as a measure of how strictly that constraint should be enforced. The right hand side of Equation 4.1 is readily available in a tabular form which makes it suitable for easy, fast calculation [Rob85].

Using PERT modeling technique to address uncertainties in prediction results is generally not sufficient. Tight bounds should be used in PERT. If very loose bounds are used, the average values calculated by PERT tend to shift away from the most likely value, resulting in inaccurate predictions.

4.3 Assumptions

This section will explain the major assumptions underlying BAD. Some of these restrictions are due to the current assumptions of the ADAM System, and some are made to reduce the complexity of the prediction.

4.3.1 Operation to operator mapping

In the current ADAM System [JKMP89] (Synthesis and Prediction Subsystems), it is assumed that all operations of a given type are to be implemented by a single operator type. ALUs are not directly supported. High-level transformations before the synthesis process may be performed to implement operations of different types on the same ALU operator. In this case, the types of operations in the design representation can be modified by the designer or pre-processing software such that these operations will have a common operation type. Alternatively, some off-critical path operations may efficiently use cheaper operators. Such operations can be assigned a different type prior to prediction. Pipelined operators to implement operations are not yet supported.

4.3.2 Clocking and Control

Most synthesis systems do not have any notion about the clocking until very late in the design phase. It is generally assumed that either a single phase clocking or 2-phase clock will be used. Another issue is to define how the control works in the design. The control and the data paths might be running in overlapped or non-overlapped mode. This makes a difference in controller design and scheduling for conditional execution. We assume non-overlapped execution of control and the data path in our architecture model.

4.3.3 Non-deterministic Delays

It is assumed that the behavioral description, the design library and the design constraints to BAD are specified using a non-deterministic model. Non-deterministic design entities (unknown or non-deterministic operator/operation delays, non-deterministic timing constraints and dynamic scheduling) are not supported. Since the system considers several tradeoffs of design entities and the fact that the system is geared neither for the most parallel nor for the most serial implementation of the behavioral description, non-deterministic design entities are too hard to handle at this time.

4.3.4 Loops

The loop timing model does not allow internal loops with non-deterministic iteration counts. It is assumed that fixed-iteration internal loops in the behavioral description are unrolled so that the resulting dataflow graph is acyclic, as proposed in [PP88] or [Tri85]. If unrolling inner loops is not realistic due to a very high iteration count, some of the techniques presented in this chapter are still applicable to handle inner loops in a bottom-up fashion or to handle inner loops by imposing constraints on the hardware [PK89b].

4.3.5 Resynchronization

It is also assumed that resynchronization (flushing the pipeline) does not occur. The major reasons for resynchronization are exceptions and conditional branches. The presence of resynchronization affects the performance. Based on the resynchronization rate, a pipeline with a slow initiation rate and short circuit delay may have a higher performance than another pipeline with a faster initiation rate but longer circuit delay. More details can be found in [Par85, PP88].

4.3.6 Timing Constraints

In the remainder of this thesis, we assume that there are no local timing constraints unless it's stated otherwise. Only global timing constraints are considered. These types of constraints are defined on

- the throughput (data initiation rate), and
- the overall system delay (delay from the time all inputs are available to the time all outputs are ready).

4.4 The Prediction Mechanism

The prediction mechanism, in general terms, mimics the overall synthesis process. Due to the difficulty in efficiently solving the overall synthesis problem as a single step, behavioral synthesis has been broken into subtasks which are solved separately. Conventional behavioral synthesis considered in this prediction tool consists of the following subtasks:

- design style selection (pipelined or non-pipelined),
- module selection (library configuration),
- scheduling,
- operation allocation,
- register allocation,
- interconnect allocation,
- control generation, and
- layout generation

BAD includes a number of separate predictors for each synthesis subtask listed above. The behavioral synthesis process is modeled as a decision tree as shown in Figure 4.3. Only some design decisions are represented in this decision tree. If there is no branching shown in Figure 4.3 for any particular synthesis subtask (e.g. control generation), then the predictor currently assumes a rigid synthesis subtask and generates predictions for well optimized results in accordance with the assumptions on the synthesis subtask.

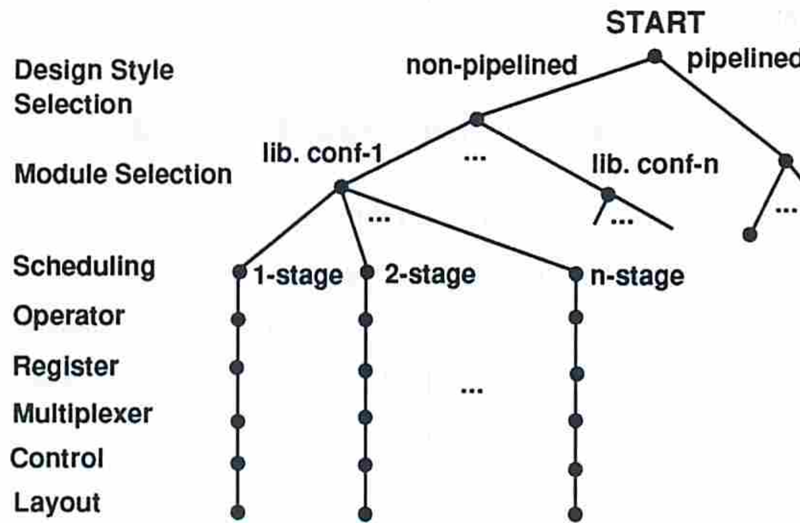


Figure 4.3: The prediction generation in BAD

Some of predictors which have been incorporated into BAD are based on design theory, some are heuristic, and some are empirical. These predictors will be discussed in the following sections.

4.5 Operator Estimation

There are two types of architecture models considered in our synthesis model:

Definition 4.5.5 Single-cycle Architecture Model *is a scheduling model in which all operations are assumed to be combinational and to take one time step*

to execute. The clock cycle time has to be longer than any operation delay to ensure the correct execution of the specification.

Definition 4.5.6 Multi-cycle Architecture Model *is a scheduling model in which all operations are assumed to be combinational and to take one or more time steps to execute.*

We assume some restrictions on the multi-cycle architecture model to keep the scheduling complexity at a moderate level. The execution of multi-cycle operations should either start at the beginning of a clock cycle or should end at the end of a clock cycle. Assuming otherwise would require the use multiple clocks to have well optimized designs.

The pipelining model assumed in this chapter (functional pipelining) is the same as the one devised by [PP88] and also used in [Jai89]. Resource allocation is strictly dependent on the initiation interval of the pipeline. The effect of scheduling infeasibilities (bottlenecks in scheduling) caused by certain graph topologies is reflected in the circuit delay through the pipeline, not in the resource allocation.

Definition 4.5.7 *Let S be a schedule and μ be a set of measures. S is an* **efficient schedule** *if and only if there does not exists any other schedule S' which is better than S for at least one measure in μ and is as good as S for the rest of the measures [Fre82].*

The bounds and estimations on operator allocation are based on utilization and the concept of operator optimality as previously defined in [Gir84], [PP88] and [Jai89].

The operator type utilization previously defined for the single-cycle architecture model by Jain [Jai89] has been extended to multi-cycle architecture model. There are separate definitions for pipelined and non-pipelined design styles.

Definition 4.5.8 *The utilization of operator type i in a pipelined design is defined as*

$$u_i = \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{o_i \times l} \quad (4.2)$$

where o_i is the number of operators of type i, d_i is the delay of operator type i, c is the clock cycle, l is the initiation interval, and n_i is the number of operations of type i. n_i^e is the maximum number of operations of type i which can be executed in parallel. n_i^e is equal to n_i for non-conditional graphs and it is less than or equal to n_i for graphs having conditional blocks [Par85].

Definition 4.5.9 *The utilization of operator type i in a non-pipelined design is defined as*

$$u_i = \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{o_i \times N} \quad (4.3)$$

where N is the number of time steps (stages).

4.5.1 Pipelined Design Style

The idea of performing predictions by varying the initiation interval and calculating the operator allocation for each initiation interval has been taken from Jain, et al. [JPP87].

4.5.1.1 Theoretical Bounds

In [JPP87], Jain predicts the lower bound on the operator allocation for the pipelined design style and single-cycle architecture model as shown in Equation 4.4:

$$o_i \geq \lceil \frac{n_i^e}{l} \rceil \quad (4.4)$$

A new relation capable of handling multi-cycle operations has been derived which handles Jain's original single-cycle architecture model as a special case [Küç90], [Jai90]. This new relation is given in Theorem 4.5.1.4.

Theorem 4.5.1.4 *Given n_i^e, d_i, c, l, and the pipelined design style, the sufficient number of operators of type i in the design is given as*

$$o_i \geq \lceil \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{l} \rceil \tag{4.5}$$

Proof: The proof is based on the utilization definition given in Equation 4.2. Given the fact that utilization of any operator type cannot be more 100%, Equation 4.2 can be re-written as

$$1 \geq \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{o_i \times l}$$

which can be re-organized to get Equation 4.5. □

The outer ceiling function in Equation 4.5 exists due to the fact that operators are discrete entities and any fractional requirements for operators have to be rounded to the smallest integer larger than the requested amount. The inner ceiling function in Equation 4.5 is due to the restricted multi-cycle architecture used, in which the operation delays are discretized into time steps using the clock cycle time.

4.5.1.2 Operator Allocation Estimation

In our pipelining model, it is always possible to achieve the lower bound operator allocation at the expense of longer pipelines; therefore, the lower bound, the most likely and the upper bound values for o_i are all the same.

Only operation delays, at this point, contribute to the whole clock cycle.

4.5.2 Non-Pipelined Design Style

Non-pipelined operator estimations are performed by varying the number of stages and calculating the lower-bound operator allocation for each case as proposed by Jain [JMP88]. The assumptions made by Jain on the clock cycle time

and architecture model are different from ours since we include multi-cycle operations.

4.5.2.1 Theoretical Bounds

In [JMP88], Jain predicts the lower bound on the operator allocation for the non-pipelined design style and single-cycle architecture model as shown in Equation 4.6:

$$o_i \geq \lceil \frac{n_i^e}{N} \rceil \tag{4.6}$$

A new relation capable of handling multi-cycle operations has been derived [Küç90] and given in Theorem 4.5.2.5.

Theorem 4.5.2.5 *Given n_i^e, d_i, c, N, and the non-pipelined design style, the lower bound on number of operators of type i in the design is given as*

$$o_i \geq \lceil \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{N} \rceil \tag{4.7}$$

Proof: The proof is based on the utilization definition given in Equation 4.3. Given the fact that utilization of any operator type cannot be more 100%, Equation 4.3 can be re-written as

$$1 \geq \frac{n_i^e \times \lceil \frac{d_i}{c} \rceil}{o_i \times N}$$

which can be re-organized to get Equation 4.7. □

4.5.2.2 Operator Allocation Estimation

In a non-pipelined design, it is not always possible to achieve a lower-bound operator allocation for any given number of stages, N. The operator types which are highly utilized are more likely to be bottlenecks in the schedule. A heuristic approach is used for such cases: When the utilization for a operator type i is

predicted to be above 80%, the upper bound for o_i is increased by one operator from the lower bound, while the most likely value stays at the lower-bound value.

Here, the whole clock cycle is again considered to be composed only of data path operation delays.

4.5.3 Operator Timing Characteristics

As stated earlier, the operator area predictions are performed by varying the initiation interval for pipelined designs and the number of stages for non-pipelined designs. In order to predict the entire area-delay tradeoff curve for pipelined designs it is necessary to know the realistic range for the number of stages and the circuit delay through the pipeline as well as the initiation interval. The methods presented in [JPP87] and [JMP88] did not include these predictions.

From this point in the chapter, we will assume that the architecture model used in always the multi-cycle model. Single-cycle model is a special case of multi-cycle model with $\lceil \frac{d_i}{c} \rceil = 1$.

Theorem 4.5.3.6 *Given an acyclic dataflow graph, $G(O,V)$, where O is a set of operations and V is a set of values, and a module library, L to implement operations of G, let C be the critical path delay (the longest path delay) through G (satisfying the data dependencies in G) for the operator delays of L, also let c be the clock cycle time used during scheduling. Then, the number of time steps, N in any schedule generated is lower bounded for both pipelined and non-pipelined design styles as:*

$$N \geq \lceil \frac{C}{c} \rceil \tag{4.8}$$

Proof: The scheduling task maps operations of G onto discrete time steps while satisfying data dependencies in G and externally imposed local timing constraints and discretizes the execution of operations by using time steps each of which is c long. C, by definition constitutes a lower bound on the execution time and

when expressed in terms of the clock cycle time, C is $\frac{C}{c}$ clock cycles. The ceiling function in Equation 4.8 is introduced since N is an integer and can not be less than $(\frac{C}{c})$ for functional correctness. Externally imposed timing constraints can only increase N and therefore not change the lower bound. □

4.5.3.1 Non-pipelined Designs

The number of stages for a non-pipelined design can be determined by the delays of individual operators and the total amount of processing required.

Theorem 4.5.3.7 *Given an acyclic dataflow graph, $G(O,V)$ and no externally imposed timing constraints. The number of time steps, N in any efficient non-pipelined schedule generated is upper bounded as*

$$N \leq \sum_i n_i \times \lceil \frac{d_i}{c} \rceil \tag{4.9}$$

where i varies over operation types.

Proof: The proof is based on the worst case scenario. Given the fact that there are no externally imposed local timing constraints, the longest schedule would have one operation executed at any time (no parallelism). Each operation type i takes $\lceil \frac{d_i}{c} \rceil$ time steps to execute. When this is summed over all operations in G, Equation 4.9 is obtained. The assumption that an efficient schedule is generated eliminates the possibility of idle time steps in the schedule. □

Estimations The upper bound given in Equation 4.9 corresponds to the most serial implementation and is a quite loose bound. In Equation 4.10 a more practical heuristic upper-bound estimate is given.

$$N \leq \max\left((\sum_i n_i \times \frac{d_i}{c}), \max_i(n_i \times \lceil \frac{d_i}{c} \rceil) \times \kappa \right) \tag{4.10}$$

where κ is a fudge factor used, and i varies over operation types.

The first term in Equation 4.10 models the computation requirement while the second term in Equation 4.10 crudely models the effect of the operation type with the highest probability of causing scheduling infeasibilities. For the non-pipelined design style, the number of time steps is varied between lower and upper bounds given in Equations 4.8 and 4.10 and the rest of the predictions are performed for each case.

4.5.3.2 Pipelined Designs

Estimation of the initiation interval range

The range for the initiation interval in the functional pipelining model is simply based on the computational requirements. First, we will derive an initiation interval upper bound for non-inferior designs and then explain how we use this bound in our estimations.

Theorem 4.5.3.8 *Given the pipelining model explained in Section 4.5 and if the lower bound operator allocation is used as explained in Section 4.5.1.2, any pipelined design with no conditionals and with initiation interval, l such that*

$$l \geq \max_i \left(n_i \times \lceil \frac{d_i}{c} \rceil \right) \tag{4.11}$$

will be an inferior design with respect to throughput (inverse of initiation interval) where i varies over operation types.

Proof: For each operation type i, the lower bound attainable operator allocation for initiation interval l is given by Equation 4.5. If there are no conditionals, this equation reduces to

$$o_i \geq \lceil \frac{n_i \times \lceil \frac{d_i}{c} \rceil}{l} \rceil$$

For $l \geq n_i \times \lceil \frac{d_i}{c} \rceil$, o_i becomes 1. When all operation types are simultaneously considered, the condition to have $\forall o_i$, $o_i = 1$ is simply found as

$$l \geq \max_i \left(n_i \times \lceil \frac{d_i}{c} \rceil \right)$$

Let l_{max} be $\max_i(n_i \times \lceil \frac{d_i}{c} \rceil)$. It is clear that l_{max} is the minimum initiation interval which satisfies the condition to have a implementation with $\forall o_i,\ o_i = 1$. Any other implementation with $l \geq l_{max}$ would have the same operator allocation and inferior throughput. □

Estimations As shown in Theorem 4.5.3.8, the initiation interval range for non-inferior pipelined implementations of unconditional graphs is bounded as follows:

$$1 \leq l \leq \max_i \left(n_i \times \lceil \frac{d_i}{c} \rceil \right) \tag{4.12}$$

In case of designs with conditionals, no formalization for conditional operator sharing yet exists. If this upper bound is used for designs with conditionals, the upper bound would be a looser bound compared to the tightness of the same upper bound for designs with no conditionals, but still would be valid.

For the pipelined design style, BAD currently varies the initiation interval from the lower bound to the upper bound given in Equation 4.12 and generates a set of predictions. The implementations with initiation interval higher than l_{max} are not considered in the predictions based on the assumption that those implementations will unnecessarily keep operators idle and are more likely to be inferior. But, of course, this does not guarantee that the final design would be inferior.

Estimation of the number of time steps The pipeline length is harder to predict because the effect of scheduling infeasibilities has to be taken into account. Currently, the pipeline length is temporarily predicted by a simple piecewise

linear approximation of empirical observations as described in Figure 4.4. This model is over-simplified to be able to accurately predict the number of time steps in a pipelined design. A possible way of improving this prediction will be discussed in Section 4.15.

Procedure Pipeline_Length (initiation interval: l)

Begin
if all o_i for interval l are same as o_i for $l-1$ **then**
  Use $Pipeline_Length(l-1)$
else
  $N.lb = \lceil \frac{C}{c} \rceil$
  if $l < \frac{l_{max}}{2}$ **then**
    $N.ub = \sum_i n_i \times \lceil \frac{d_i}{c} \rceil$ {Model 1}
  else
    $N.ub = \lceil \sum_i n_i \times \frac{d_i}{c} \rceil$ {Model 2}

  $N = \lceil N.lb + \frac{N.ub - N.lb}{l_{max} - 1} \times (l-1) \rceil$

  If switching between models **then**

    $N = (N +$ Pipeline_Length$(l-1))/2$

If $N < l+1$ **then**
  $N = l+1$ Pipelining criterion

End.

Figure 4.4: The heuristic for pipeline length estimation

4.6 Register Estimation

The estimation of register area is performed in 2 steps as proposed by Mlinar [Mli91]. The first step is to find the width of the dataflow graph which gives the maximum number of registers for a non-pipelined implementation. The second step is to use the graph width in predicting the actual number of registers in pipelined and non-pipelined design styles.

Kurdahi proposed a min-flow max-cut algorithm to find the width of the dataflow graph [Kur87]. This method is later improved and used by Mlinar [Mli91]. Although the min-flow algorithm is quite fast for human interaction, a heuristic explained in Section 4.6.2 is temporarily used to find the graph width because of two reasons. The first reason is that the run-time was a pressing factor in our applications. During approximately the same time spent to perform register estimations using Mlinar's method, we can generate complete estimations including operator, register, multiplexer, control, and wiring estimations. The second reason is that the algorithm developed by Mlinar needs modifications for practical use. His technique lacks the notion of multi-output operations and reduces all outputs from an operation to a single value since the technique does not have a value concept. Modification of his technique and inclusion of it in our prediction techniques is under consideration.

4.6.1 Notation

Some notation concerning graph characteristics which will be used in the remaining sections of the chapter is listed below:

ibw_i	The total input bitwidth for operation/operator type i
obw_i	The total output bitwidth for operation/operator type i
$icnt_i$	The number of inputs for operation/operator type i
$icnt_i$	The number of outputs for operation/operator type i
I	The total number of external input bits

I_c	The number of external input bits which are constant or don't need storage
O	The total number of external output bits
F	The total output bits of operations which only depend on the external inputs
A	The average bitwidth of operations in the dataflow graph
Y	The total output bits of all operations
X	The total input bits of all operations
R	The estimated number of bits of registers
M	The estimated number of bits of 2-to-1 multiplexers
T	The maximum number of stages considered for the design style
N_c	The number of operations on the critical path
$\lceil \frac{N}{I} \rceil$	The number of data sets simultaneously executed in the pipeline

4.6.2 Determination of graph width

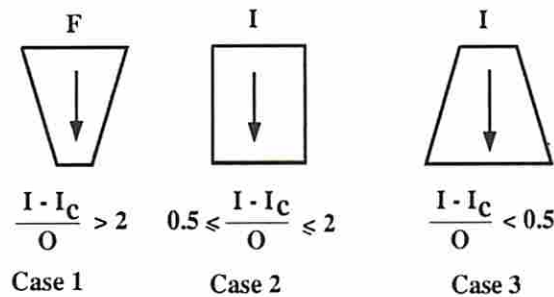


Figure 4.5: Possible graph shapes considered

R_{max}, the estimated width of the graph (in terms of bits) is estimated by characterizing the shape of the graph based on the global, static characteristics of the graph, without performing extensive graph processing. Figure 4.5 shows three graph shapes considered in the estimation of graph width. R_{max} is a function of

the number of inputs to the graph and *the average value fanout* in the dataflow graph, and is estimated by the technique shown in Figure 4.6.

Procedure Estimate Graph Width

Begin
if $(I - I_c) > 2 \times O$ **then** {The graph narrows at the output}

$$R_{max} = F \times \frac{X}{Y + F}$$

else if $O > 2 \times (I - I_c)$ **then** {The graph widens at the output}

$$R_{max} = \max\left(O, (I - I_c) \times \frac{X}{Y + I - I_c}\right)$$

else

$$R_{max} = (I - I_c) \times \frac{X}{Y + (I - I_c)}$$

End.

Figure 4.6: Graph width estimation technique

4.6.3 Register Area Estimation

After R_{max} is estimated, the heuristic shown in Figure 4.7 is used to estimate the number of registers. The heuristic is taken from [Mli91] with some modifications. The register area is estimated for each pair of initiation interval and the number of time steps supplied from earlier prediction methods. The estimated register area is mainly a function of estimated graph width, the total bitwidth of graph outputs, the total bitwidth of graph inputs, the total bitwidth of constant graph inputs, and functional parallelism.

The pipelined register estimate is based on the fact that several instances of the dataflow graph will be executed simultaneously requiring storage proportional to the parallelism and furthermore, the primary inputs also need to be stored. The register estimate for the non-pipelined case is based on a linear interpolation of register count between the lower bound and the width of the

graph (upper bound estimate). Although our experience show that the number of registers in non-pipelined designs do not directly correlate to the number of time steps [4.1], we biased our register estimations using an interpolation such that serial implementations have higher register cost than parallel implementations. The reasoning behind this is that the prediction tool automatically goes through several implementations to find the best implementation which satisfies the constraints. During this search process, we did not want to favor the selection of a more serial implementation just because it is predicted to have one or two less registers. The deviations of predicted number of registers from the actual number of registers are not very different for both our interpolation technique and Mlinar's technique.

4.6.4 Register Delay Estimation

The delay introduced into the clock cycle due to register propagation delays, rd, is 2 times the register delay propagation delay. Register propagation delays contribute to the clock cycle twice, first for storing data path values, second for the storage of controller outputs due to non-overlapped execution of data path and control path used in the ADAM Synthesis System.

4.7 Interconnect Estimation

Interconnect estimation is based on use of multiplexers only since it is quite hard to characterize and estimate global bus structures for our design architecture assumptions. Multiplexer estimates are themselves highly heuristic. The algorithm is shown in Figure 4.9. When functional operators and registers are shared, there generally exists some multiplexing logic to route the incoming data to the operators. The main idea behind multiplexer estimation is to characterize the

[4.1] Mlinar's estimation technique assumes a constant number of registers in non-pipelined designs.

Procedure Register Estimation

Begin
if $N > l$ **then** {Pipelined}
$R.lb = \min(I - I_c, O) + \lceil \frac{N}{l} - 1 \rceil \times R_{max}$

$R.ub = \max(I - I_c, R_{max}) \times \lceil \frac{N}{l} \rceil$
$R.ml = \frac{R.lb + R.ub}{2}$
else {Non-pipelined}
if $N = 1$ **then**
$R.ml = O$ {Only outputs are stored}
else
if $N > T$ **then** $N = T$ {Saturate N}
$R.ml = \lceil \frac{N-1}{T-1} \rceil \times \max(R_{max} - O, 0) + 1.5 \times O$ {Linear interpolation}

$R.ub = \max(O, R_{max})$
$R.lb = O$
If inputs need to be stored
$R.ml = \max(R.ml, I - I_c)$
$R.ub = \max(R.ub, I - I_c)$
$R.lb = \max(R.lb, I - I_c)$
End.

Figure 4.7: Register Estimation Algorithm

multiplexing requirements of each predicted design. This is done by estimating the total number of module inputs for each module type and the total number of sources incoming to these module inputs so that the number of multiplexers to route the data from these sources to module inputs can be estimated. The multiplexer delays introduced to the clock cycle are estimated by using the estimated size of multiplexer trees in front of operators (considering operator chaining) and registers.

The basic assumption behind multiplexer estimation is that each operation/value type will only be mapped to a single operator/register type. This

assumption allows us to estimate the multiplexing requirement for each operator/register type based on the resource sharing of the operator/register type.

4.7.1 Upper Bounds on the Multiplexer Area

The upper bound on the amount of multiplexing required to share hardware has been derived earlier by [Kur87] and also used in MABAL [KP89a] to show the amount of multiplexer optimization performed.

Theorem 4.7.1.9 (Taken from [Kur87]) *Given n_i operations (values) of type i to be implemented by o_i operators (registers) of the same type and the total input bitwidth of all operations, X, the amount of 1-bit 2 input multiplexers required to route the data from data sources to the operator (register) inputs in the design is upper bounded by*

$$X - \sum_i ibw_i \times o_i \quad (4.13)$$

where i is over the operation/operator (value/register) types.

Proof: The proof is based the worst case analysis in which all data to be multiplexed are unique. The proof is given in [Kur87]. □

4.7.2 Multiplexer Area Estimation

In performing multiplexing estimations from behavioral specifications, the actual number of register and operators are not known. Therefore, a heuristic approach is taken to estimate the upper bound for operator multiplexing. Two estimates for the upper bound are calculated using different methods and the average of the two is used as the estimated upper bound (see $muxcnt1$ and $muxcnt2$) in Figure 4.9.

The estimation of the upper bound multiplexing required for the registers is performed in the same fashion. The multiplexer requirements of operators

and registers are summed to get the total interconnect requirements. The estimations for lower bound and expected multiplexer allocation are based on the empirical data generated by MABAL. MABAL uses Theorem 4.7.1.9 to compare its optimized results against the upper bound it calculates. Based on the empirical data, factors relating the estimated lower bound and the expected multiplexer allocation to the theoretical upper bound have been derived to be used in the multiplexer estimation heuristic shown in Figure 4.9. These factors may require to be modified if a different data path synthesis program is to be used and the multiplexer optimization quality of that program is very different than MABAL's.

4.7.3 Multiplexer Delay Estimation

In a general design produced by synthesis tools, the critical paths determining the clock cycle go through multiplexers for operators and registers. Due to operator chaining in scheduling, it is possible that signals have to go through several operators (hence multiplexers) during one clock cycle. It is assumed that abstract multiplexers in the design are built from 1-bit 2-input multiplexers as multiplexer trees. Hence, the height of each multiplexer tree is proportional to the logarithm of the number of inputs to be multiplexed. An example of operator chaining is shown in Figure 4.8.

Definition 4.7.3.10 Average operator chaining *is the ratio of the number of operations on the critical path over the number of time steps,* $\frac{N_c}{N}$.

The lower bound on the multiplexer delays introduced into the clock cycle is zero because, in well optimized designs, it may be possible to share hardware and route data to operator/register inputs without using any multiplexers.

The most likely (expected) multiplexer delay introduced into the clock cycle, *md.ml* is calculated using the average operator chaining and average size multiplexer trees at the inputs of operators and registers as described in Figure 4.9.

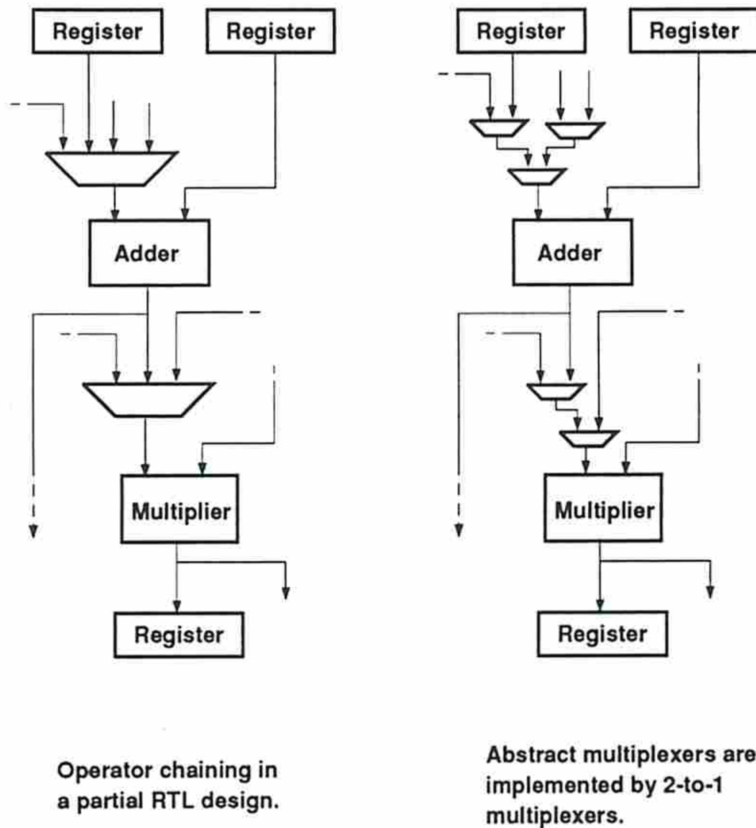


Figure 4.8: Operator chaining in a schedule

op_mux and *reg_mux* are the estimated upper bound multiplexer complexities [4.2] for operators and registers, respectively. It is assumed that due to the optimizations in the data path synthesis, the average multiplexer complexities for operators and registers are lower than these maximum values and will be half of corresponding upper bound multiplexer complexities. The rest of the multiplexer inputs are assumed to be eliminated in the optimization process. The $\log_2$ function is redefined to return zero for arguments which are less than one.

[4.2] Multiplexer complexity is the number of inputs for an abstract multiplexer.

The upper bound on the multiplexer delays introduced into the clock cycle $md.ub$ is estimated similarly using the average operator chaining and upper bound multiplexer complexities for operators and registers.

4.8 Control Estimation

The controller can be implemented by random logic or a programmable logic device. The controller implementation style is currently assumed to be a programmable logic array (PLA). The estimation of controller characteristics is performed in 2 phases. The first phase is to estimate abstract PLA characteristics which are namely the number of PLA inputs, the number of PLA outputs and the number of product terms. The second phase is to estimate the actual PLA area and delay characteristics from abstract PLA characteristics.

4.8.1 Estimating abstract PLA characteristics

Estimations for the abstract PLA characteristics are taken from Mlinar [Mli91]. Only a subset of Mlinar's PLA estimation techniques are currently used in BAD. For a behavioral description with no conditional branches and loops, the numbers of inputs, outputs and product terms of a PLA are currently estimated as follows:

The number of product terms is estimated as the number of states the PLA-based controller will go through. The number of states is estimated as the number of stages in the design plus one more state to store the primary outputs after the execution is complete as shown in Equation 4.14.

$$\text{Product Terms} = N + 1 \tag{4.14}$$

The inputs to the PLA consists of 3 fields: state bits, reset and start inputs. The number of bits to represent states of a finite state machine (FSM) with binary state coding is proportional to the logarithm of number of states. The total number of input bits of the PLA is estimated as shown in Equation 4.15.

Procedure Multiplexer Estimation

Begin

$muxcnt1 = \sum_i ibw_i \times (n_i - o_i)$ {Operator multiplexing}

$muxcnt2 = I + R.ml + \sum_i o_i \times (obw_i - ibw_i)$ {Operator multiplexing}

$muxcnt = (muxcnt1 + muxcnt2)/2$ {Average of two methods}
$op_mux = \max_i(n_i - o_i)$ {Largest operator mux tree}
$muxcnt = muxcnt + \max(Y + I - R, 0)$ {Register multiplexing}
$reg_mux = \max(Y + I - R, 0)/A$ {Largest register mux tree}

$avg_delay = (\lceil \log_2(\frac{reg_mux}{2}) \rceil + \lceil \frac{N_c}{N} \rceil \times \lceil \log_2(\frac{op_mux}{2}) \rceil) \times$ 2-to-1 mux delay

$max_delay = (\lceil \log_2 reg_mux \rceil + \lceil \frac{N_c}{N} \rceil \times \lceil \log_2 op_mux \rceil) \times$ 2-to-1 mux delay

if $l = 1$ **then**
  $muxcnt, muxdelay, op_mux, reg_mux = 0$ {No multiplexing}
  $avg_delay, max_delay = 0$ {No multiplexing}

$M.lb = 0.4 \times muxcnt \times$ 2-to-1 mux area {Empirical Observation}
$M.ml = 0.6 \times muxcnt \times$ 2-to-1 mux area {Empirical Observation}
$M.ub = muxcnt \times$ 2-to-1 mux area {Empirical Observation}

$md.lb = 0$ {Multiplexing delay}
$md.ml = avg_delay$ {Multiplexing delay}
$md.ub = max_delay$ {Multiplexing delay}

End.

Figure 4.9: Multiplexer estimation algorithm

$$\text{PLA Inputs} = \lceil \log_2(N+1) \rceil + 2 \tag{4.15}$$

The output of the PLA consists of 2 fields: state bits and control bits to control registers and multiplexers in the data path. It is assumed that one unique control line per stage in a well optimized design would be sufficient to control registers and multiplexers with the help of some additional logic gates as suggested by Mlinar. This assumption was verified to be very accurate and results are shown in Table 5.1 of Chapter 5. The total number of PLA output bits is estimated as shown in Equation 4.16. The area of the additional logic gates is generally a second order effect.

$$\text{PLA Outputs} = \lceil \log_2(N+1) \rceil + N + 1 \tag{4.16}$$

The state bit outputs of the PLA have to be stored. The registers to store $\log_2(N+1)$ state bits are also included in the predictions.

The PLA estimations for behavioral descriptions with conditionals and loops which are yet to be incorporated into BAD are quite complicated. The feasibility of integrating existing methods in [Mli91] into BAD is being investigated. More details about Mlinar's PLA estimations can be found in [Mli91].

4.8.2 PLA Area Estimation

PLA layout generators use predefined macro cells [Mli91]. Mlinar gives an expression to estimate the PLA area given the number of inputs, outputs and product terms of the PLA. This expression characterizes the use of predefined macro cells in the Berkeley PLA Synthesis Tools. More details can be found in [Mli91].

4.8.3 PLA Delay Estimation

PLA delays are transition dependent. Given a personality matrix to be implemented, a very tight upper bound for any possible transition can be estimated. Since the personality matrix is not available in the estimation phase, the estimations are based on the best, the average, and the worst cases.

The PLA delay estimation technique is taken from Gupta [Gup90]. This technique is based on characterizing the charging/discharging behavior of AND-OR planes (or NOR-NOR planes) of the PLA. An RC model is used to characterize the PLA delay by estimating the load capacitance and the drive capability corresponding to particular PLA transitions. Three types of PLA delay estimations were imported from the work in [Gup90], namely:

- **Best-Best** Case: Only one product term is connected to the output and all inputs are driving the product term (minimum load capacitance, maximum driving capability).
- **Average-Average** Case: Only half of the product terms are connected to the output and only half of the inputs are driving the product terms (average load capacitance, average driving capability).
- **Worst-Worst** Case: All product terms are connected to the output and only one input is driving the product term (maximum load capacitance, minimum driving capability).

More details can be found in [Gup90].

4.9 Wiring Estimation

4.9.1 Wiring Area Estimation

Consideration of only RTL characteristics of a design is generally not enough to make major design decisions. The prediction of layout characteristics is also

quite important. In our approach, it is assumed that a standard cell placement and routing methodology is used although another wiring area estimator could be substituted for whichever design method was being used. Among other placement and routing methodologies, we can easily count gate array implementations, custom placement and routing style and the programmable logic device (PLD) style. PLEST [KP86] is a program which predicts the wiring area of a design if a standard cell placement and routing approach is used. It has been experimentally shown that the wiring area estimates of PLEST are within 10% of the actual chip areas [KP89b].

The input to PLEST is the total RTL area of the design and the number of 2-point nets. PLEST currently uses 3 micron technology parameters. The estimated RTL area of the design, $Area_{RTL}$ is defined as a summation of estimated operator, register and multiplexer areas.

$$Area_{RTL} = R + M + \sum_i o_i \tag{4.17}$$

where i varies over operator types.

The number of 2-point nets is estimated as

$$nets = \left(\sum_i o_i \times ibw_i\right) + R + 2 \times M + O \tag{4.18}$$

where i varies over operator types.

PLEST could have been called repeatedly to estimate the wiring area. But, the run time for PLEST (in the order of tens of seconds on a Sun-3 workstation) was too long to meet our requirements. Therefore, interpolation of previous PLEST results is used. By varying the RTL area and the number of 2-point nets we ran PLEST several times and we obtained a two dimensional array of PLEST results ranging from the smallest chip size to the largest size possible and from very few nets to a large number of nets. This is required only once

for each technology. A binary search algorithm together with a two-dimensional interpolation method is used to approximate the expected wiring area, $W.ml$.

Since PLEST results are known to be within 10% of the actual area figures, the lower bound for wiring area, $W.lb$ is estimated as $0.9 \times W.ml$ and the upper bound for wiring area, $W.ub$ is estimated as $1.1 \times W.ml$. The wiring area approximation, $W.ml$ is usually within 1 to 3% of PLEST results. The accuracy could be further improved by increasing the number of sample points in the table at the expense of longer run times. The simple two dimensional interpolation algorithm used in the wiring area estimations is shown in Figure 4.10.

Our approach is not only limited to using PLEST. The look-up table can be constructed using any other estimator and/or actual layouts. But, a sufficient amount of data has to be tabularized so that the accuracy of approximations would be acceptable.

4.9.2 Wiring Delay Estimation

The estimation of wire delays is performed in two steps. The first step is to estimate the delay for individual wires. The second step is to estimate the wiring delays introduced into the clock cycle.

Definition 4.9.2.11 A uniform layout methodology *is a layout methodology for RTL designs consisting of connected operators, registers, and multiplexers such that the connections are designed in a uniform way. That is to say all outputs have the same driving capability, all inputs have the same load capacitance, all wires have the same conductance, etc. The only variable parameters in the layout are the wire length and fanout.*

A uniform layout methodology defined above is assumed for the wiring delay model, which will be explained in the following sections.

Procedure Estimate PLEST($Area_{RTL}$,$nets$)

{$aa[K]$ and $net[L]$ are the arrays of active area and number of 2-point nets supplied to PLEST, and $r[K,L]$ is wiring area predictions from PLEST}

Begin
Find the smallest row number p such that $Area_{RTL} > aa[p]$
Find the smallest column number q such that nets $> net[q]$

$$P = r[p,q] + \frac{Area_{RTL} - aa[p]}{aa[p+1] - aa[p]} \times (r[p+1,q] - r[p,q])$$

$$Q = r[p,q+1] + \frac{Area_{RTL} - aa[p]}{aa[p+1] - aa[p]} \times (r[p+1,q+1] - r[p,q+1])$$

$$W.ml = P + \frac{nets - net[q]}{net[q+1] - net[q]} \times (Q - P)$$

$W.lb = 0.9 \times W.ml$
$W.ub = 1.1 \times W.ml$
End.

Figure 4.10: The interpolation algorithm used to estimate PLEST results

4.9.2.1 Delay Estimations for a single wire

The delay estimation for a single wire is based on a RC model, taken from Gupta [Gup90] and was originally published by Sakurai [Sak83]. This model estimates the delay through a connection with a known length and known fanout given some technology parameters. The RC model for wire delays for a given technology is represented by the following function:

$$d(wire_length, fanout)$$

where $fanout$ is defined as the number of signal ports driven by the wire.

In the wiring delay model used, wire delays arise from signal propagation through the RC network and also from charging the load capacitance. The wire

length is used to estimate the signal propagation delays using the RC delay model and the fanout is used to estimate the load capacitance and charging time for that load capacitance.

A wire delay through a zero-length wire means that there will not be any wire delays due to signal propagation and only load capacitance charging time will contribute to the wire delay.

4.9.2.2 Estimations of delays introduced into the clock cycle

The wire delay introduced into the clock cycle is predicted by estimating the number of wires in series (due to operation chaining and multi-level multiplexing) and the fanout in the design. In the architecture we assume, a single clock will be used and the clock cycle time determined by the maximum register to register path delay in the design.

Theorem 4.9.2.10 *Let w.lb be the minimum wire delay introduced into the clock cycle. Then, in a synchronous design with a single phase clocking and a uniform layout methodology, wd.lb is 2 times the delay through a zero-length wire with no fanout.*

Proof: The proof is based on the assumed architecture style used by the synthesis tools. There has to be at least one operator in the design. The design style is synchronous and operators are combinational blocks. The shortest path possible through this operator in the design is composed of

- one path from registers to operator inputs,
- one path from operator inputs to operator outputs, and
- one path from the operator outputs to register inputs.

Only the first and third paths contribute to the *wiring delays* introduced into the clock cycle. The delay through the second path is taken into account in functional delays. The smallest wire delays in the same uniform layout methodology are obtained by the minimum wire length and the minimum fanout. □

Case 1: Minimum wire delay The estimation of the minimum wire delay introduced into the clock cycle, $wd.lb$ uses Theorem 4.9.2.10 and the delay through a minimum-length, minimum fanout wire has be to estimated by the RC model mentioned earlier as shown below:

$$wd.lb = d(0, 1.0) \times 2 \qquad (4.19)$$

Case 2: Average wire delay The most likely wire delay introduced into the clock cycle, $wd.ml$ is calculated using estimated values of average wire length (avg_wl), average fanout per wire ($avgf$), average operator chaining, and average size multiplexer trees at the inputs of operators and registers.

The average wire length is estimated within PLEST during layout area estimation and is also used in the wiring delay estimations. Given an estimated design o_i operators for each operation type i, R registers, M 2-to-1 multiplexers, I external inputs and O external outputs, the numerator of Equation 4.20 calculates the signal (wire) destination points in the design. Similarly, the denominator calculates the number of source points. The average fanout per wire is then estimated as the fraction of the total number of signal destination points to the total number of signal source points in the estimated RTL design as shown in Equation 4.20. The effects of fanout which is internal to modules are assumed to be already included in delay characteristics of the modules.

$$avgf = \frac{\left(\sum_i o_i \times ibw_i\right) + R + 2 \times M + O}{\left(\sum_i o_i \times obw_i\right) + R + M + I} \qquad (4.20)$$

Given the estimated delay characteristics of an average wire, $d(avg_wl, avgf)$, the next step is to estimate how many of such wires in series would be contributing to the clock cycle time.

The estimation of average wire delays introduced into the clock cycle is similar to the estimation of average multiplexer delays. The effects of multiplexing on the clock cycle are discussed in Section 4.7.3. When abstract multiplexers are constructed from 2-to-1 multiplexers, each multiplexer contributes a delay proportional to the logarithm of the multiplexer size to the clock cycle. Since, 2-to-1 multiplexers are basic modules, each 2-to-1 multiplexer has an associated wire delay for connections to other modules. Therefore, each abstract multiplexer also introduces wire delay proportional to the logarithm of the multiplexer size into the clock cycle time.

In general, due to the optimizations in the data path synthesis, the multiplexer complexities (number of inputs for the abstract multiplexer) for operators and registers are lower than the maximum values, op_mux and reg_mux (previously defined in Figure 4.9). The estimations for multiplexer complexities in Equation 4.21 assume that half of the multiplexer inputs would be eliminated in the optimization process in accordance with multiplexer delay estimations.

The most likely wire delay introduced into the clock cycle is estimated as

$$wd.ml = d(avg_wl, avgf) \times \left(1 + \lceil \log_2 \frac{reg_mux}{2} \rceil + \frac{N_c}{N} \times \left(1 + \lceil \log_2 \frac{op_mux}{2} \rceil\right)\right) \tag{4.21}$$

where reg_mux and op_mux are calculated as described in Figure 4.9.

Case 3: Maximum wire delay The maximum wire delay introduced into the clock cycle, $wd.ub$ is estimated in the same manner as the average wire delays. But, this time, estimations for maximum wire length, average fanout per wire, average operator chaining, and the maximum multiplexer complexity at the inputs of operators and registers are used. The length of the longest wire,

max_wl is roughly estimated as $\sqrt{Area_{RTL}}$. A good prediction method for the maximum fanout is not available at the time. The maximum fanout can be estimated assuming that a signal is connected to all inputs of all modules in the design. But, we believe that predicting the wiring delay upper bound using the maximum wire length and the maximum possible fanout would result in a very loose upper bound which is of little use.

$$wd.ub = d(max_wl, avgf) \times \left(1 + \lceil \log_2 reg_mux \rceil + \lceil \frac{N_c}{N} \rceil \times (1 + \lceil \log_2 op_mux \rceil)\right) \quad (4.22)$$

4.10 The effects of the clock cycle time on design characteristics

The determination of the clock cycle time is very important in order to be able to obtain highly efficient schedules. The clock cycle time used to introduce the concept of *time steps* during scheduling determines the amount of discretization caused. A very fast clock rate would minimize the discretization and produce highly efficient schedules but, on the other hand, it would result in a very large number of control states. Having a large number of control states results in very large and slow controllers which are not preferable at all. Having a large number of control states also may cause unacceptable performance degradation when exceptions occur. The determination of the clock cycle time, as it can be seen from above, is a tradeoff issue by itself. There has not been any tradeoff studies to our knowledge on the effects of varying clock cycle time on the design quality.

The clock cycle time is also constrained by other design considerations, examples of which include communication with peripheral devices, running synchronously with other parts of the design.

4.11 Run-time complexity of the predictions

The run-time complexity of the prediction tool introduced in this chapter depends on

- the amount of pre-processing required to extract the information requested by the prediction techniques,
- the amount of branching performed in the design decision tree shown in Figure 4.3, and
- the complexity of individual prediction techniques.

The run-time complexity of the pre-processing phase is $\mathcal{O}(n^2)$ where n is number of operations in the dataflow graph. The pre-processing run-time complexity is dominated by the complexity of levelizing the dataflow graph which is $\mathcal{O}(n^2)$ assuming that the number of edges in the dataflow graph is $\mathcal{O}(n^2)$. Determining other static graph characteristics (some are listed in Section 4.6.1) has an $\mathcal{O}(n)$ complexity.

One type of branching in the design decision tree is dependent on the number of possible library configurations. Let m_i be the number of operators which can implement operation type i, then, there are $\mathcal{O}(\Pi_i m_i)$ possible library configurations in our synthesis model. Both pipelined and non-pipelined operator prediction techniques produce $\mathcal{O}(n)$ prediction points for each library configuration in the single-cycle architecture style. In the following complexity analysis, only single-cycle architecture style will be considered.

For each library configuration considered, the critical path delay of the dataflow graph has to be calculated. This process takes $\mathcal{O}(n^2)$ for each library configuration. All prediction techniques (operator, register, multiplexer, control and wiring) take constant time per prediction point.

The overall run-time complexity of BAD is

$$\mathcal{O}(n^2 + n\Pi_i m_i + n^2\Pi_i m_i)$$

which can be simplified to

$$\mathcal{O}(n^2 m)$$

where m is the number of possible library configurations ($m = \Pi_i m_i$).

The run-times encountered in real life are also highly dependent on the constraints given. The complexity shown in this section is for the worst case. In reality, BAD prunes the search on the design decision tree as soon as the partial predictions violate the constraints. The prediction generation in BAD is also controlled by the constraints and BAD does not create predictions which would be too slow for the constraints given. When tighter constraints are given, less number of prediction points are created and prediction tool executes faster. Other factors affecting the run-time similarly are the clock cycle time and the architecture style.

In case multi-cycle architecture is used, the expected number of prediction points per library configuration is still $\mathcal{O}(n)$ for each design style if the given clock cycle time is close to operator delays. But, the clock cycle time can be made arbitrarily small resulting in finer grain discretization and a very large number of prediction points. However, the increase in the number of prediction points generated is linearly proportional to decrease in the clock cycle time while other things are equal. E.g. if the clock cycle time is reduced by half, then the number of prediction points will approximately double.

4.12 Feasibility Analysis

Feasibility analysis of prediction results can be requested to be performed automatically using the statistical model mentioned in Section 4.2. Feasibility analysis is performed for the chip area constraint, the performance (initiation interval) constraint and the overall system delay constraint. In analysis of timing-related constraints, first, an adjusted clock cycle time is calculated for each predicted design to accommodate the estimated register, multiplexer, control and wiring delays introduced into the clock cycle. Then, standard statistical methods [Rob85] are used during the feasibility analysis for each constraint.

4.13 Synthesizing the predicted designs

The results from BAD include not only the overall characteristics (e.g area, throughput and total circuit delay) of designs satisfying design constraints, but also guidance of how to reach those predicted design points. For each predicted design a complete trace of predictions are available. This trace includes the design decisions made to reach a particular prediction point (e.g. library configuration (add3, mul2), pipelined design style, initiation interval of 2 clock cycles) as well as the individual estimations (e.g. 2 adders, 3 multipliers, 10 registers, 20 multiplexers, wire and PLA area/delays). All of estimated entities (including lower bounds, most likely values, and upper bounds for each estimation) mentioned in this chapter are virtually available.

Therefore, it is quite easy to drive the behavioral synthesis tools to produce a design with similar characteristics to the predicted design. However, it can be possible to produce designs differently than what BAD suggests to obtain better design characteristics.

4.14 Advantages of Behavioral Area-Delay Predictions (BAD)

In the previous version of the ADAM Synthesis System, the design style selection was performed as the first step in the synthesis process. Module selection followed manual design style selection. After the design style and the module library configuration have been decided, conventional high-level synthesis tasks were performed. SLIMOS [JPP88] is a program which performs module selection for pipelined design style, but also performed acceptable for non-pipelined design style. In SLIMOS, single-cycle architecture style is assumed, which was later improved on the new version of the module selection program, MOSP [Jai90]. MOSP assumes multi-cycle operations and performs module selection for pipelined design style with a different algorithm than SLIMOS. There are no published results yet showing how good MOSP performs. Both SLIMOS and MOSP selects the module library for given constraints. Since they only consider functional area, the area constraint has to be in terms of the functional area which is not readily available.

SLIMOS and MOSP type of programs as well as manual design style selection performed in ADAM become obsolete with the introduction of BAD. Both SLIMOS and MOSP make the module selection decisions based on the lower bound functional resource allocation predictions, which is not an accurate model. On the other hand, BAD performs predictions from resource allocation to layout for all possible module library configurations SLIMOS and MOSP models consider. By estimating the clock cycle time, initiation interval, overall delay and layout area, and enumerating possible library configurations, BAD searches a much larger portion of the design space with more comprehensive predictions. Automatic feasibility check and elimination of inferior predictions by BAD correspond to simultaneously performing the design style and module selection.

4.15 Limitations of BAD

The prediction tool presented in this chapter is the first comprehensive predictor from design style selection phase to layout. Some of the prediction techniques used in this prototype program need improvement.

We can start with the scheduling model used in the predictions. In the predictions, scheduling infeasibilities arising from the data dependencies (dataflow graph topology) are currently not modeled. In highly parallel non-pipelined designs, scheduling infeasibilities considerably increase the functional resource requirement to produce a schedule. The prediction technique underestimates the functional resource allocation in such cases.

The major problem in predicting the resource allocation is that most scheduling approaches (an exception is [HCDd88]) treat operators equally during the resource allocation. According to our data path tradeoff studies [KP90d], the scheduling approaches should first try to allocate minimum number of operators whose area is very large and try to complete the rest of the resource allocation for smaller operators as required. The reasoning behind this is the fact that the area required to share a small operator and area to duplicate a small operator are often very alike, while on the other hand, this does not hold for large operators. If a scheduling approach as described above is used, our functional resource estimations would be very accurate for large operators and predictions would be underestimating resource allocation for small operators. But, this error would be partially compensated by the interconnect estimation. If the resource allocation for a particular type of operations is underestimated, then, the interconnect estimator which estimates the multiplexing requirements from the number of operations and the number of operators of each type will overestimate the multiplexing requirement for that particular operation type. As a result, the area overall estimations will still be very accurate mainly because of the existing functional area versus interconnect area tradeoff curve as shown in Chapter 3.

In our pipelining model, scheduling infeasibilities do not show themselves in the functional resource allocation. Therefore, functional resource allocation estimates are very accurate. However, scheduling infeasibilities in our pipelining model reflect themselves on the number of stages of the schedule produced. Since we are not currently modeling scheduling infeasibilities, these estimations are not as accurate as we would like to.

Scheduling infeasibilities can be taken into account by using distribution graphs for operation time frames in a similar way to ELF which constructs the initial resource allocation [Gir91] or as proposed by Hagerman [Hag91]. Distribution graphs are also employed in force-directed scheduling [PK89b, CT90].

One limitation of BAD, which is mostly due to assumptions of the ADAM Synthesis System is the fact that no hardware pipelining is considered. Most behavioral synthesis systems do not consider hardware pipelining although there are exceptions like [McF91]. An example of hardware pipelining can be given by the following case; two adders are chained (one directly feeding the other) in the same time step. In our scheduling model, we only consider the maximum path delays of individual operators. Given an adder delay of x, the delay through both adders will be assumed to be $2x$. In reality, for ripple carry adders, the second adder starts computation as soon as the lowest order output bit from the first adder is available resulting in less delays than assumed. If this information is used during scheduling, then more operations can be packed into each time step resulting in faster designs. The effect of hardware pipelining becomes more important when operator chaining is employed heavily, which occurs frequently in highly parallel designs generated by MAHA. But, the effect of hardware pipelining is reduced when multi-cycle operations are allowed in the schedules. Using multi-cycle operations allows using a faster clock, which in turn, reduces the possibility of operator chaining.

4.16 Summary

In this chapter, we have presented a comprehensive set of accurate prediction methods to be used in high-level and system-level synthesis. The RTL characteristics of designs can be predicted accurately by the methods presented here. The layout area predictions are currently performed after PLEST. The results of several experiments to validate prediction techniques can be found in Chapter 5.

This prediction tool and/or the techniques presented can be used effectively in a number of design tasks including but not limited to design space exploration prior to high-level synthesis, evaluation of behavioral transformations and guidance for system-level decisions. The prediction techniques are also suitable to be used within synthesis tools (e.g. scheduling) to guide synthesis decisions. At the behavioral synthesis level, design style selection and module selection problems are simultaneously solved by BAD. BAD is now being used in a behavioral partitioning package which uses prediction methods to guide its search. There is a strong need for both accurate and fast predictions during the synthesis process. It is important to be able to quickly predict the final result of any design decision and many of these design decisions must be tried in order to reach globally better designs. Therefore, the speed of the predictions become as important as the accuracy. BAD is coded in C and is approximately 9,000 lines of code. The run-time for BAD averages about 0.5 msec of CPU time per predicted design on a Sun Sparc 4/460 (180msec for 346 predicted designs), satisfying our run-time requirements for the predictions [KP90a].

Chapter 5

Validation of Behavioral Area-Delay Predictions

In this chapter, the results of a set of experiments which were performed to validate the prediction techniques will be given. The AR Filter, FIR Filter and Elliptic Wave Filter will be used as example designs in this chapter. There is also a demonstration of how feasibility analysis is performed on the predictions. Prediction and actual synthesis results for each design characteristic selected were plotted on the same graph for several examples, design styles, and implementations. Synthesis and prediction results are shown by points, but the points for prediction results are connected by a solid line to help easier visual comparison.

The synthesized designs were produced by the ADAM synthesis tools [JKMP89], namely

- MAHA [PPM86] (scheduling),
- Sehwa [PP88] (pipeline scheduling),
- MABAL [KP90e] (operator, register and multiplexer allocation, and module binding), and
- CSG [Wen89] (control specification generation).

Pipelined schedules were actually created using Sehwa's urgency scheduling algorithm which is an internal routine of Sehwa [PP88]. MABAL was used

to generate a single design for each schedule. MABAL was asked to generate designs with minimum operator allocation for the schedule, the minimum register allocation and no tri-state drivers. CSG is a program which takes MABAL output and generates a controller specification ready to be fed to ESPRESSO. But, a small utility program using a very simple greedy technique was written to minimize the number of PLA outputs since ESPRESSO only optimizes the number of product terms. The controller specification output from CSG was optimized via this utility, then results were fed to ESPRESSO and MKPLA for further optimization of product terms and PLA generation.

The design library shown in Table 1.1 was used in the experiments. This design library has the same 3 micron technology assumed by wiring and PLA delay estimations. The module library configuration (add3, mul2) was used to compare prediction and synthesis results in this section. This was done to ensure that no single library module characteristic could dominate the overall characteristics. For example, module library configuration (add3, mul1) would have made *the area consumed by adders* in the design to be insignificant with respect to the total area characteristics since the multiplier area would dominate the total RTL area. In such cases, deviations of prediction results from actual synthesis results are generally insignificant.

One problem encountered during the verification of prediction results was schedules generated by MAHA. Not being able to generate highly optimized schedules, MAHA occasionally generates schedules with inferior resource allocation. This case is more frequent in highly parallel implementations of designs with a lot of scheduling infeasibilities as can be clearly seen in Figure 5.16. When MAHA runs of out resources during the scheduling phase, it allocates more resources as needed without considering which operators are more expensive. It turns out that MAHA does not perform well in cases in which one may use more of the cheaper operators in order to use less of the expensive operators.

An example of inferior resource allocation from MAHA was with the 8-stage non-pipelined implementation of the AR Filter. MAHA used 2 adders and 3 multipliers to generate a schedule with 8 time steps. The final implementation of this design had a 30% larger area than predicted. It turned out that it is possible to get an 8 time-step schedule with 2 adders and 2 multipliers by manually scheduling AR Filter operations. In Figures 5.1 through 5.6 and Figure 5.14, results for the manual schedule are shown for the 8 time-step implementation of the AR Filter. Since the manual schedule made good use of the graph topology, the final implementation had less multiplexing than predicted and the final RTL area for the manual schedule was approximately 9% better than the predicted RTL area.

5.1 AR Lattice Filter

Figures 5.1 through 5.4 show area comparisons of prediction and synthesis results for the non-pipelined design style. The area is in mil^2. Comparisons of the estimated number of 2-point nets and the number of 2-point nets in synthesized RTL designs for the non-pipelined design style are shown in Figure 5.5.

Figure 5.6 shows the break-down of prediction and synthesis results for the design area characteristic by merging figures corresponding to individual components of total area. In this figure, the bottom curve is for the operator area. The second curve from the bottom is for (operator + register) and the third one is for the (RTL + control) area. The top curve is the expected layout area for the standard cell implementation. There are no synthesis results for the layout area characteristic since it was not possible to generate layouts for the 3 micron technology assumed by the wiring and PLA estimation techniques.

It is important to note the clock cycle times determined by MAHA for non-pipelined implementations are not exactly the same as given to the prediction program since MAHA determines the clock cycle time internally, and the clock

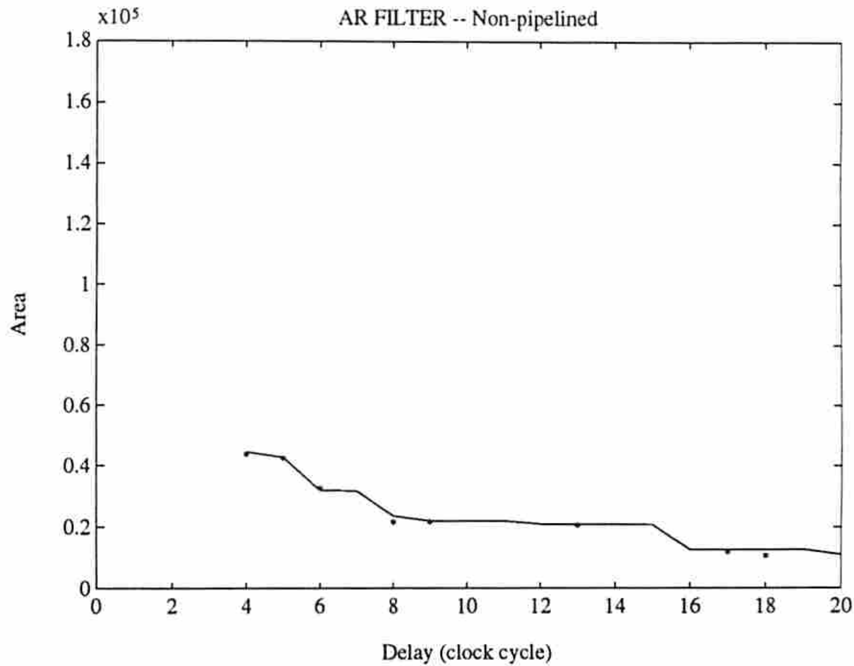


Figure 5.1: Operator area for the non-pipelined AR Filter

cycle time cannot be constrained exactly by the designer. But, cycle times generated by MAHA were with 10% of the clock cycle given to BAD. Therefore, we believe that results are still comparable.

The same type of comparisons for the pipelined design style are given in Figures 5.7 through 5.12. An additional design characteristic which is estimated for the pipelined design style is the number of stages in the pipeline. The comparison of the estimated number of stages and the actual number of stages is shown in Figure 5.13.

Assume that we have a near-perfect estimation technique to predict the layout area from an RTL description. If such a technique is used in BAD to estimate the total layout area, the direct layout area estimation in BAD would never be as accurate as estimating the layout area from an RTL description. One reason for this is that the estimated RTL characteristics from BAD would be different from the actual RTL characteristics. A second reason is that there would be more detailed information from an actual RTL design compared to the prediction

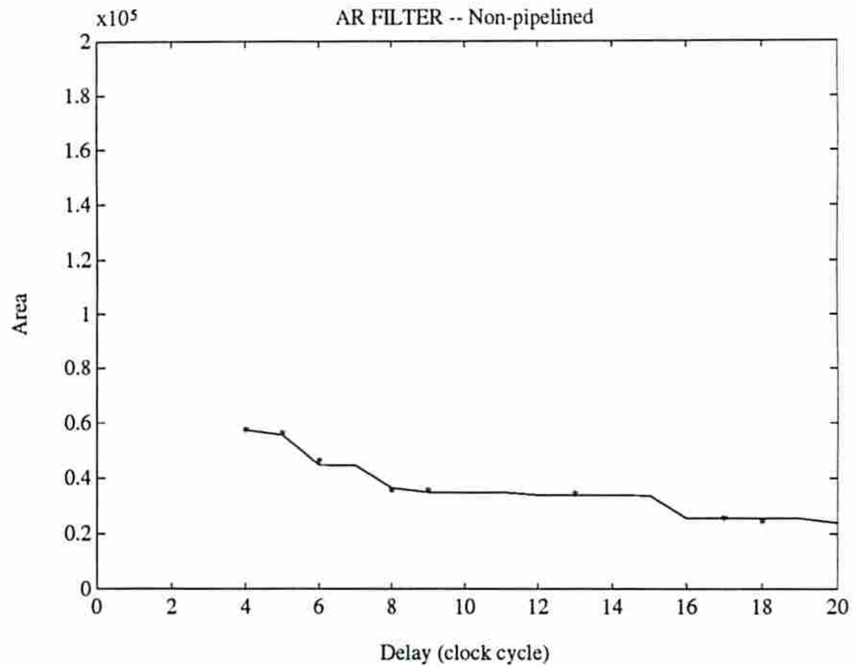


Figure 5.2: Operator + register area for the non-pipelined AR Filter

results from BAD, if the layout area estimator makes use of such information. Two comparisons of this type are shown in Figures 5.14 and 5.15 for the non-pipelined and pipelined design styles, respectively. Although it is not perfect, we use PLEST for this experiment. PLEST's estimated layout area for actual RTL designs synthesized are shown by asterisks and the points connected by lines show the estimated layout area produced by BAD. Although this comparison cannot show how accurate the layout prediction results are, it shows how sensitive the layout area is to the small variations in RTL characteristics, by using a previously verified layout area estimation package.

5.1.1 PLA Estimations

In this section, we will compare estimated and synthesized abstract PLA characteristics. Predicting the area of a PLA from its abstract characteristics is relatively easy since very regular structures are used in PLA synthesis [Mli91].

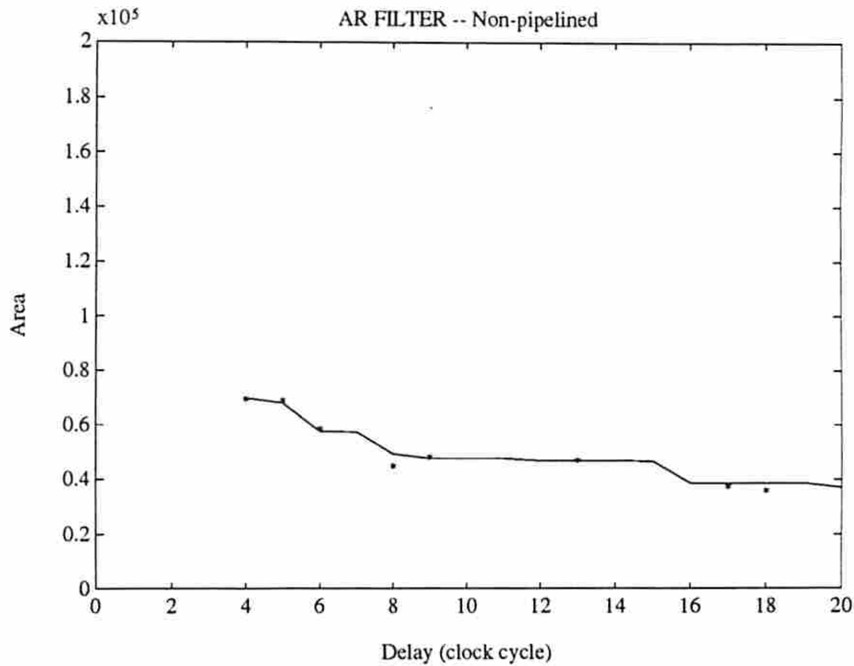


Figure 5.3: The RTL area for the non-pipelined AR Filter

It is important to note that PLA estimations assume that some small amount of external control logic will be used and predict the logic overhead. On the other hand, the control synthesis software we used creates all control signals within the PLA. Therefore, we expect that the number of PLA outputs for synthesized designs will be slightly higher than the predicted. But, results show that the multiplexer control signals are produced for free except in a few cases. Of course, there results cannot be generalized to designs with conditionals and loops.

Table 5.1 shows some comparisons of estimated and synthesized abstract PLA characteristics for the non-pipelined AR Filter.

5.1.2 Clock Cycle Time Estimation

As described earlier, we assume the clock cycle time given to BAD is spent only on the functional delays. In order to accommodate the delays through

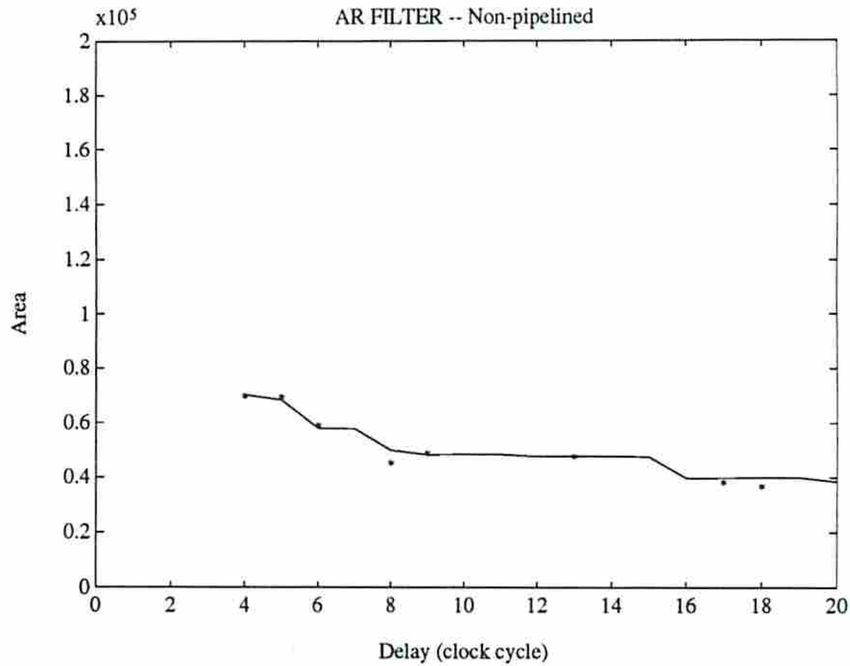


Figure 5.4: RTL + PLA area for the non-pipelined AR Filter

registers, multiplexers, controller and routing, an adjusted clock cycle time is estimated. Tables 5.2 and 5.3 lists the estimated clock cycle time for non-pipelined and pipelined implementations of the AR Filter. The clock cycle time given to BAD was 3000 ns. Table 5.2 starts with 4 time steps since the critical path of the design does not allow any schedules with 1, 2 or 3 time steps given that the clock cycle time is set to be 3000 ns. Unfortunately, the clock cycle times for the synthesized designs cannot be obtained accurately without producing layouts, which is in our immediate plans.

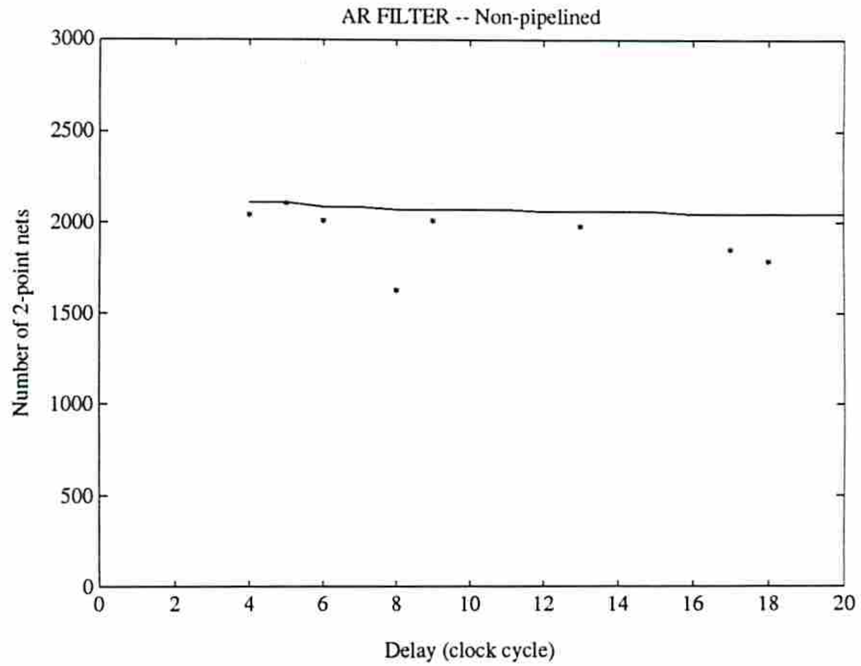


Figure 5.5: The number of 2-points nets for the non-pipelined AR Filter

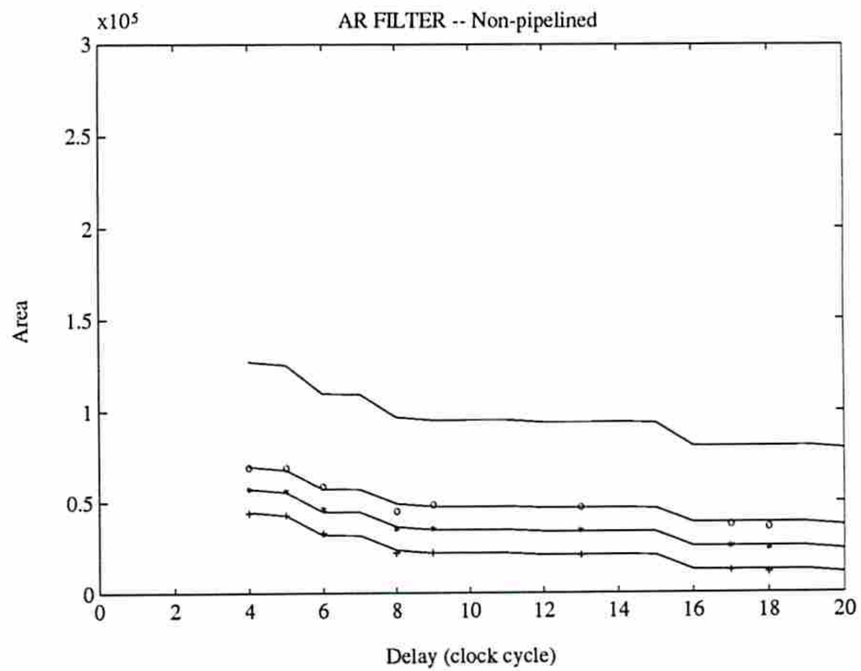


Figure 5.6: Area break-down for non-pipelined the AR Filter

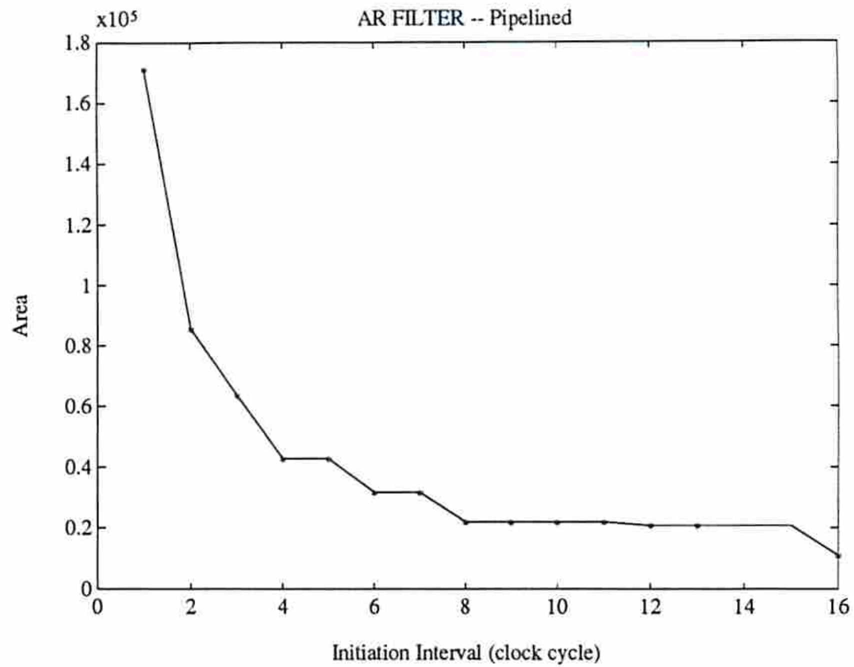


Figure 5.7: Operator area for the pipelined AR Filter

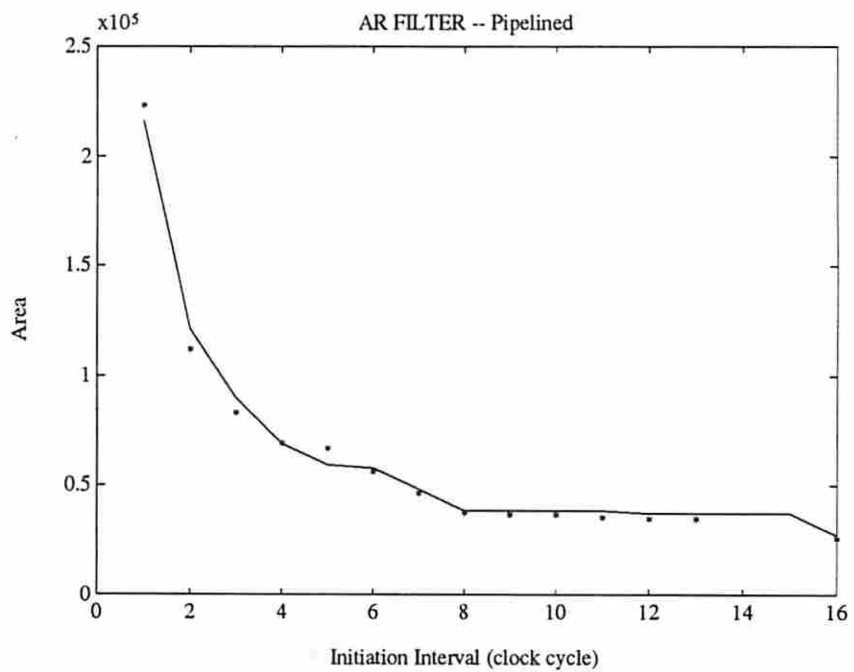


Figure 5.8: Operator + register area for the pipelined AR Filter

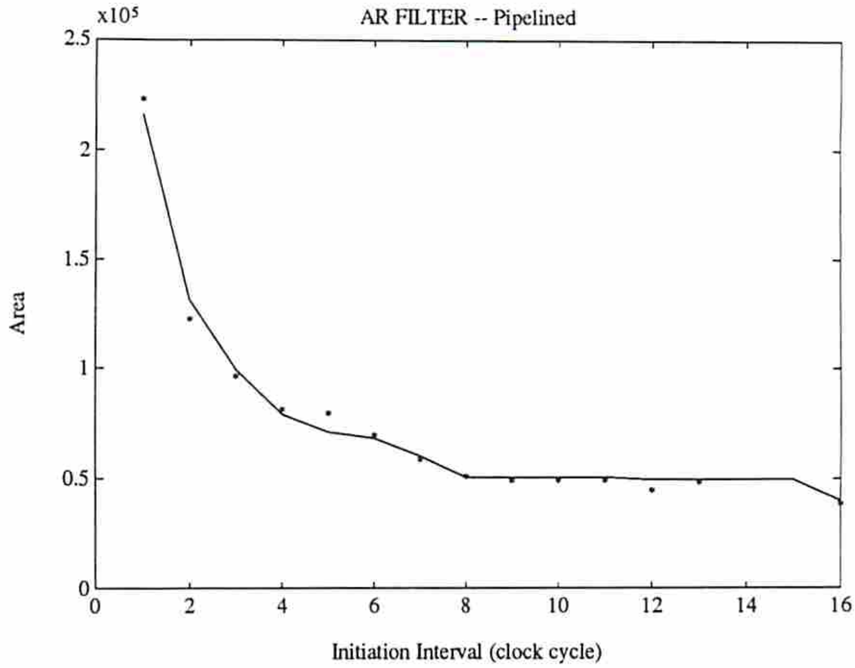


Figure 5.9: RTL area for the pipelined AR Filter

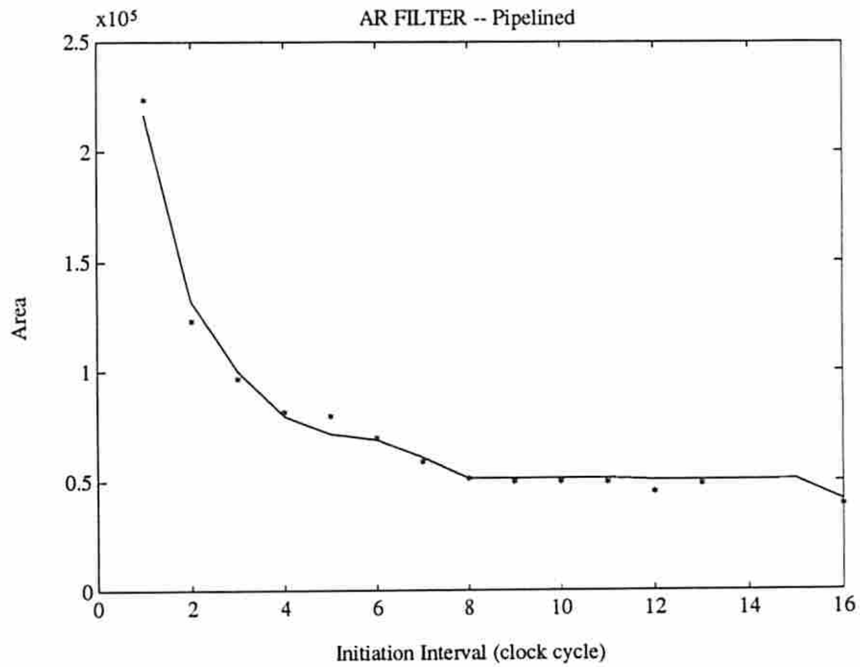


Figure 5.10: RTL + PLA area for the pipelined AR Filter

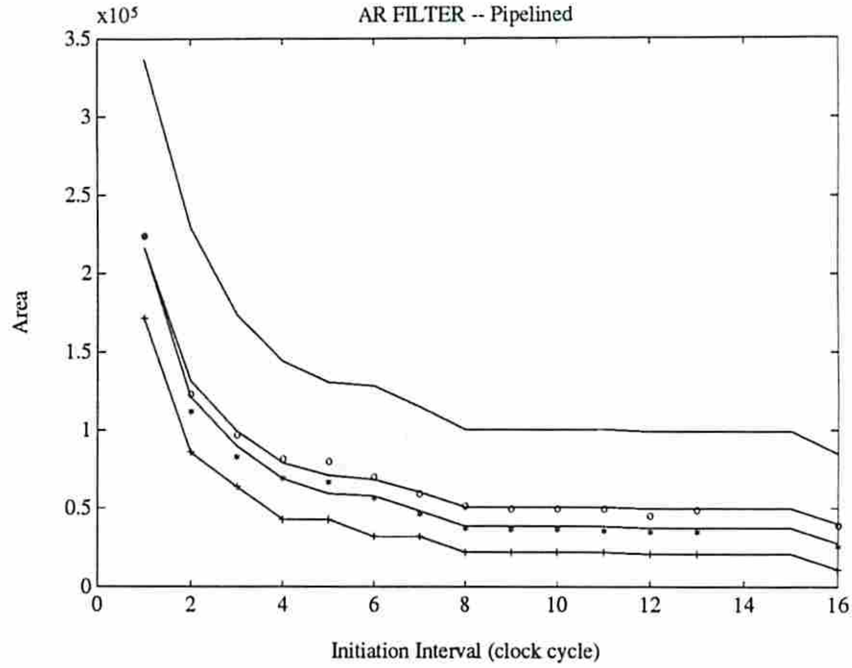


Figure 5.11: Area break-down for the pipelined AR Filter

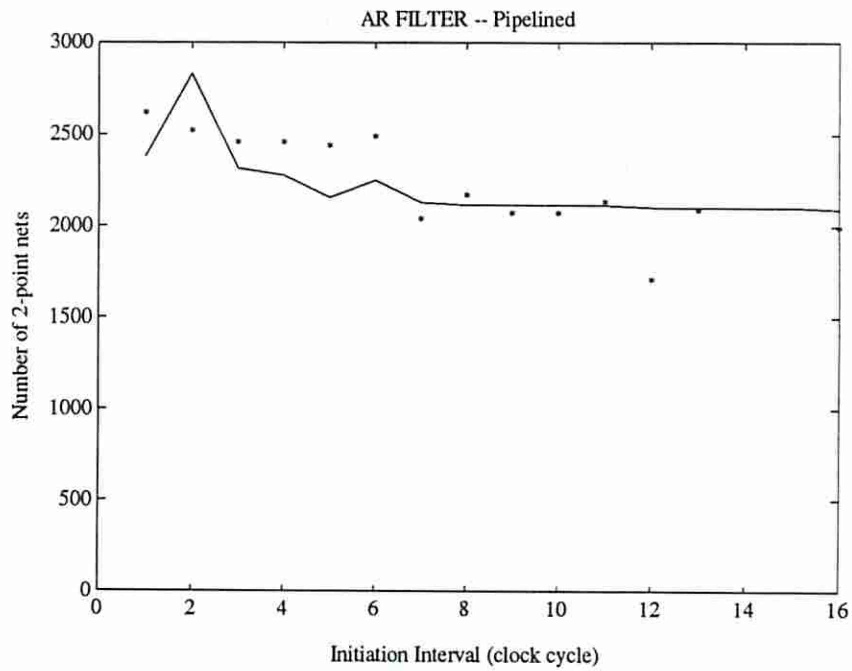


Figure 5.12: The number of 2-points nets for the pipelined AR Filter

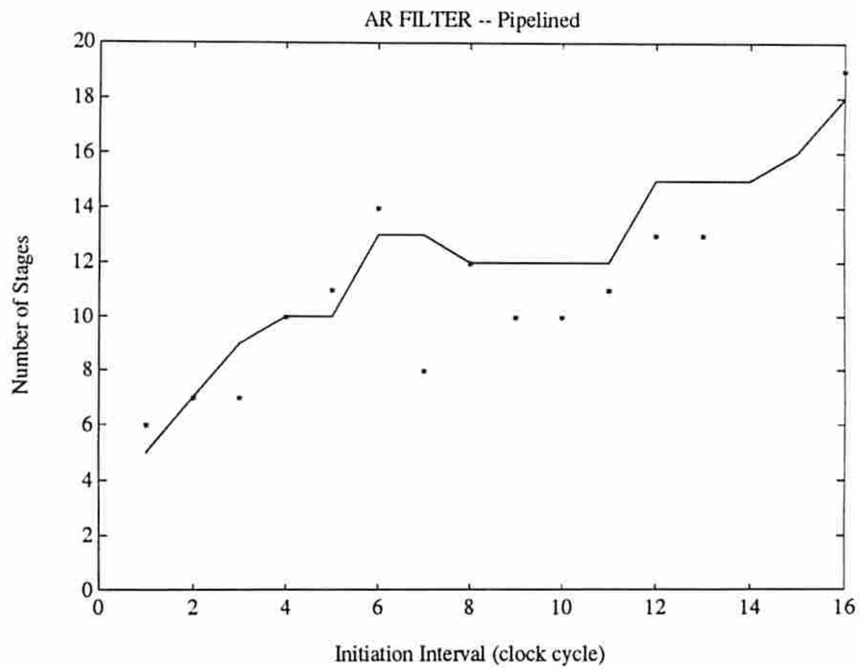


Figure 5.13: The number of stages in the pipeline for the AR Filter

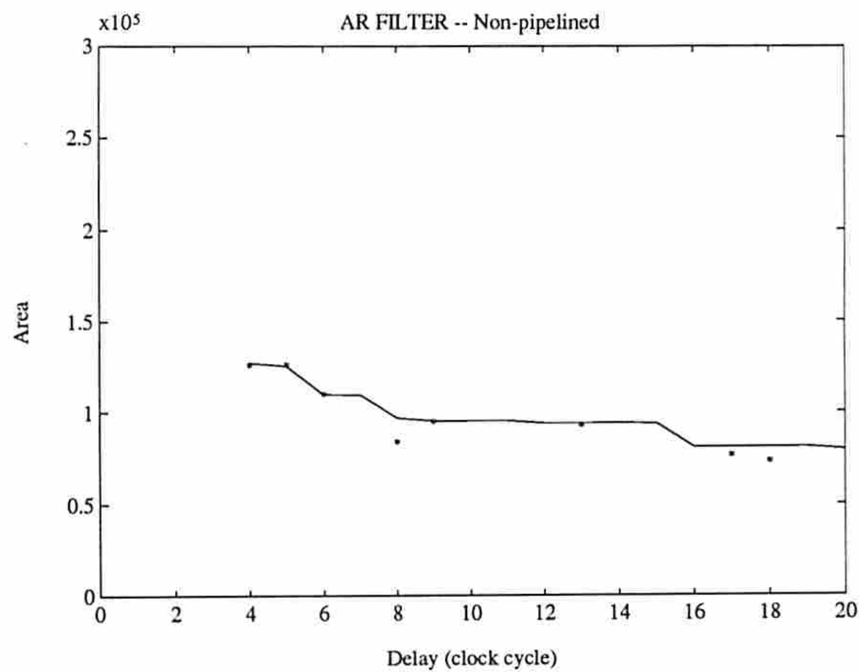


Figure 5.14: Comparisons of layout area for two methods (Non-pipelined)

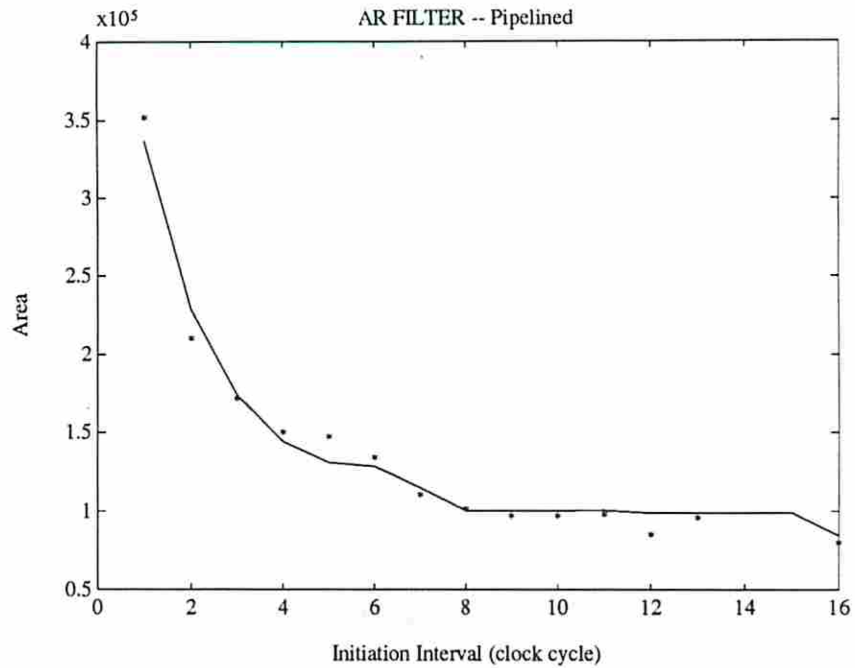


Figure 5.15: Comparisons of layout area for two methods (pipelined)

N	Estimated			Synthesized		
	Inputs	Outputs	Product Terms	Inputs	Outputs	Product Terms
1	3	3	2	4	2	2
3	4	6	4	5	7	4
4	5	8	5	5	11	5
5	5	9	6	5	10	6
6	5	10	7	5	15	7
8	6	13	9	6	13	9
9	6	14	10	6	24	10
13	6	18	14	6	19	14
17	7	23	18	7	21	18
18	7	24	19	7	22	19

Table 5.1: Estimated and Synthesized PLA Characteristics for the AR Filter

N	Estimated Clock Cycle Time (ns)
4	3114
5	3110
6	3105
7	3103
8	3086
9	3085
10	3085
11	3085
12	3084
13	3084
14	3084
15	3085
16	3083
17	3083
18	3083
19	3083
20	3082

Table 5.2: Estimated clock cycle times for non-pipelined implementations of the AR Filter

l	Estimated Clock Cycle Time (ns)
1	3046
2	3079
3	3076
4	3073
5	3090
6	3071
7	3088
8	3086
9	3086
10	3086
11	3086
12	3086
13	3086
14	3086
15	3087
16	3085

Table 5.3: Estimated clock cycle times for pipelined implementations of the AR Filter

5.2 Elliptic Wave Filter

The results for the Elliptic Wave Filter will be given in a compact form. The explanations on Figures 5.16 through 5.21 are the same as the ones for the AR Filter. The Elliptic Wave Filter has a feedback loop, but we ignored the feedback loop for the pipelined design style in order to present results for this well-known dataflow graph structure.

Figure 5.16 compares the prediction results to synthesis results using MAHA schedules. It can be seen from Figure 5.16 that the scheduling infeasibilities play an important role for highly parallel non-pipelined implementations of the Elliptic Filter and therefore, predictions show large deviations from the synthesis results for highly parallel implementations. These deviations are mainly due to MAHA's limited scheduling capabilities. Scheduling infeasibilities and how predictions can be improved were discussed in Section 4.15. Three of the implementations for which predictions showed large deviations from the synthesis results have been resynthesized using manual scheduling. Manual schedules utilized operator chaining more than automatic schedules. We also tried to use more adders than minimum needed to be able to use minimum number of multipliers possible. The clock cycle time is slightly stretched in order to be able to increase operator chaining. But, we believe the timing characteristics of these resynthesized implementations would be competitive with the implementations using MAHA schedules due to hardware chaining and reduced area. Figure 5.17 compares the prediction results to the results for resynthesized implementations. The prediction results follow the synthesis results closely.

5.3 FIR Filter

The results for the FIR Filter will also be given in a compact form. The explanations on Figures 5.22 and 5.23 are the same as the ones explained for the AR Filter.

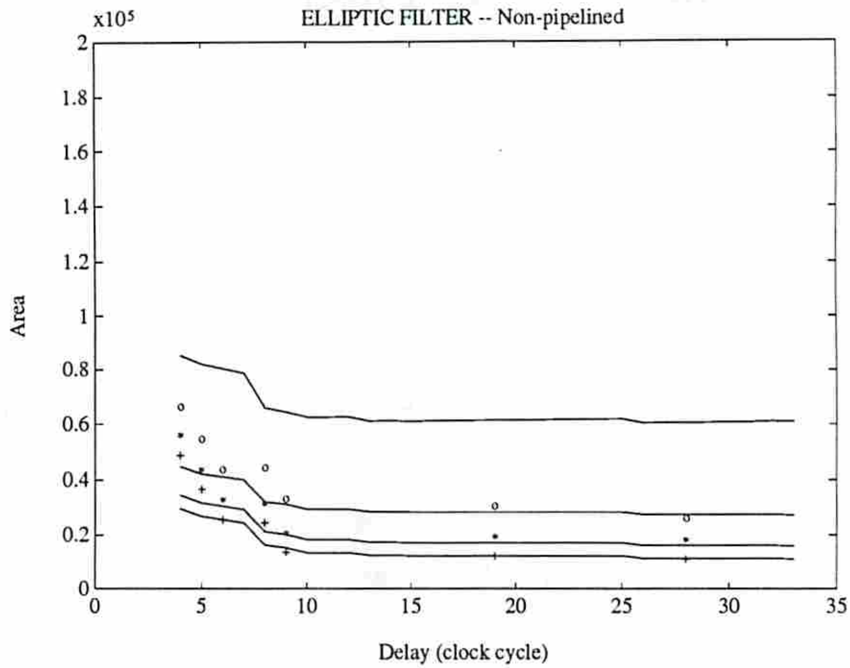


Figure 5.16: Area break-down for the non-pipelined Elliptic Filter

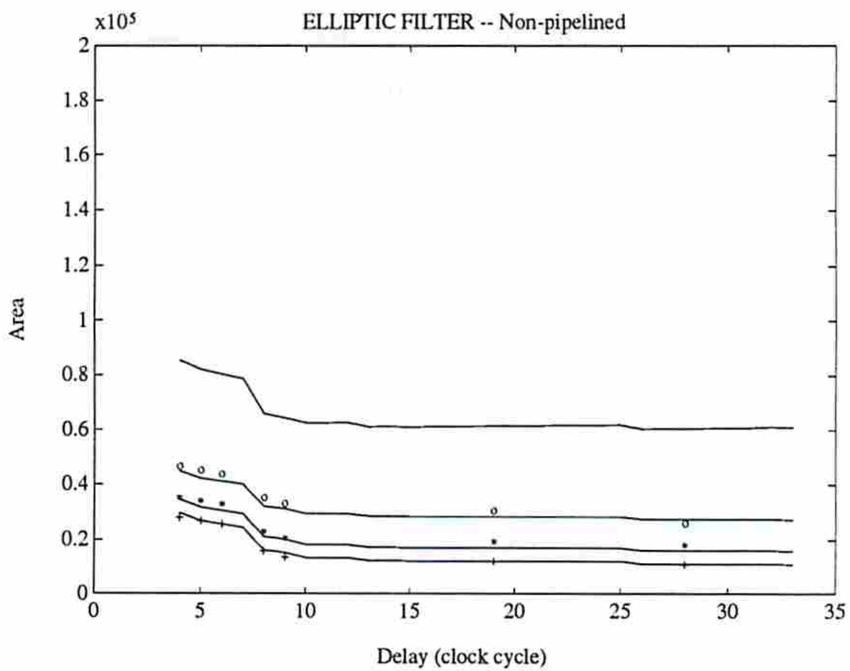


Figure 5.17: Area break-down for the non-pipelined Elliptic Filter

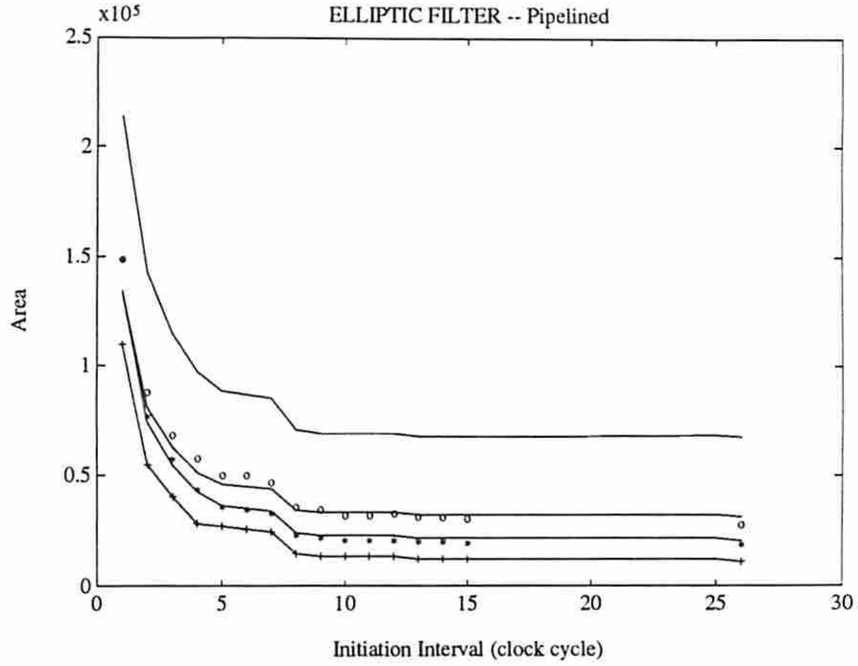


Figure 5.18: Area break-down for the pipelined Elliptic Filter

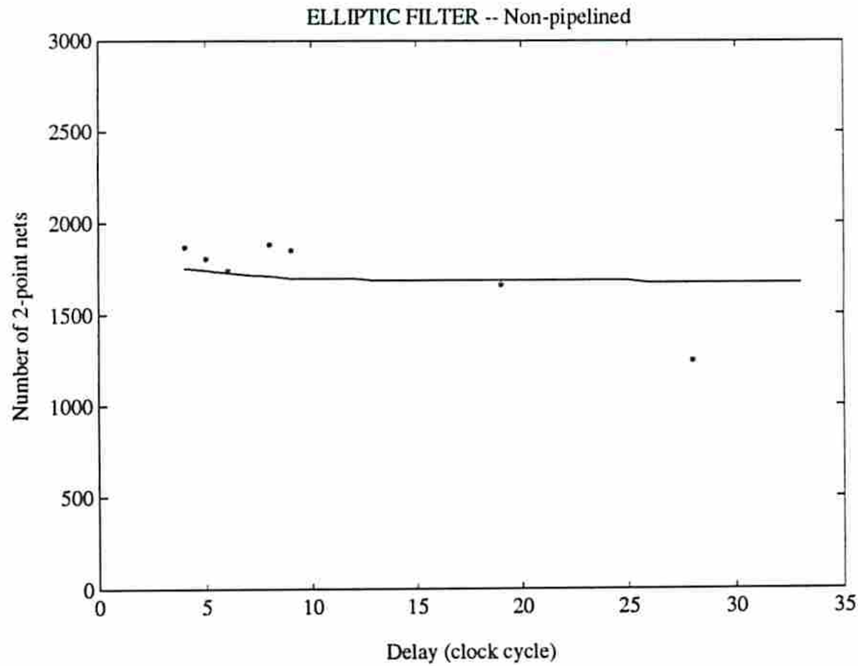


Figure 5.19: The number of 2-points nets for the non-pipelined Elliptic Filter

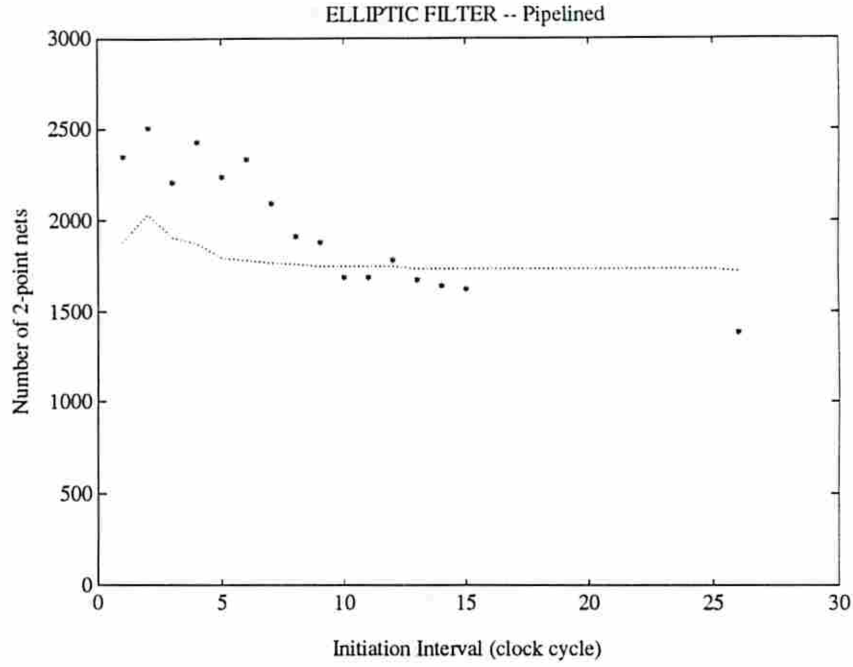


Figure 5.20: The number of 2-points nets for the pipelined Elliptic Filter

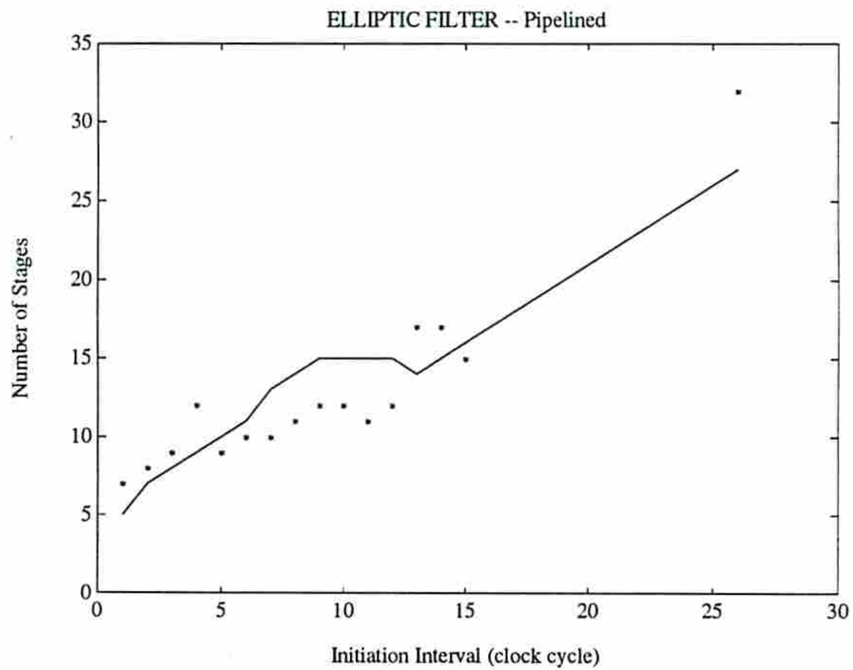


Figure 5.21: The number of stages in the pipeline for the Elliptic Filter

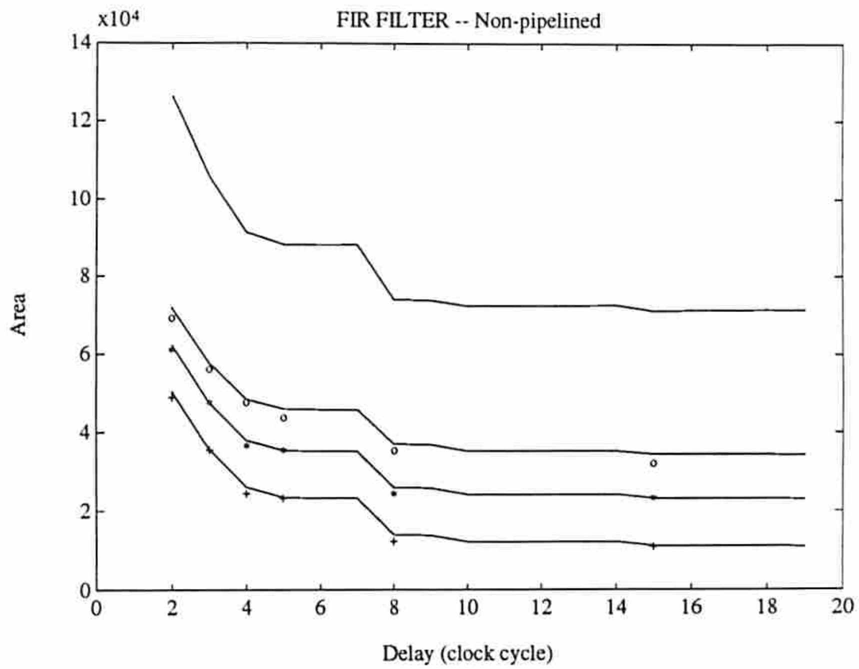


Figure 5.22: Area break-down for the non-pipelined FIR Filter

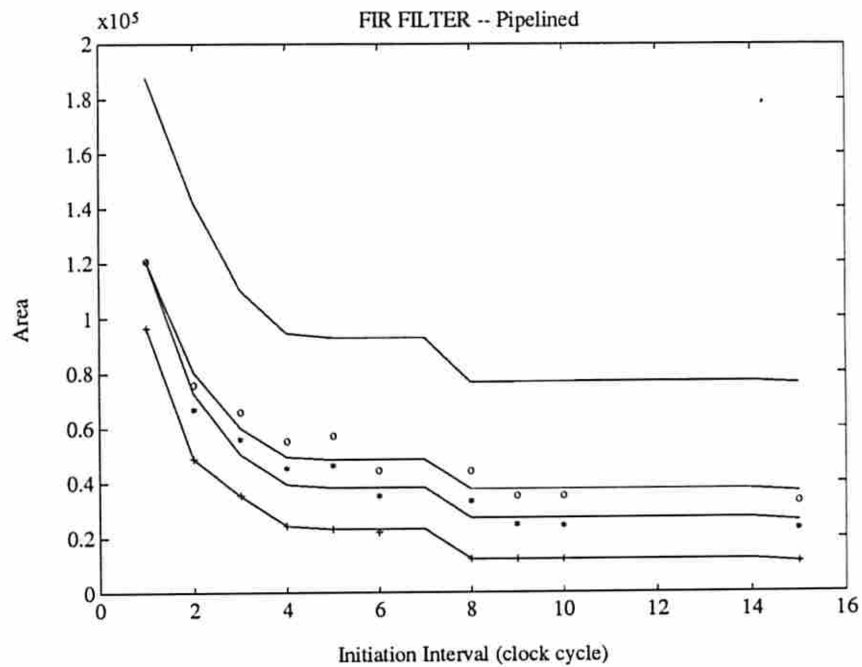


Figure 5.23: Area break-down for the pipelined FIR Filter

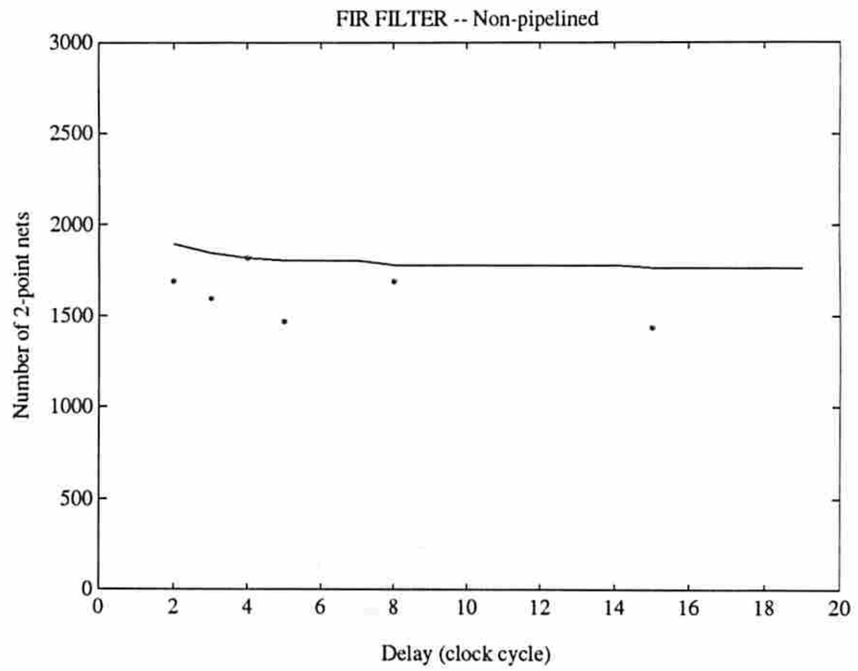


Figure 5.24: The number of 2-point nets for the pipelined FIR Filter

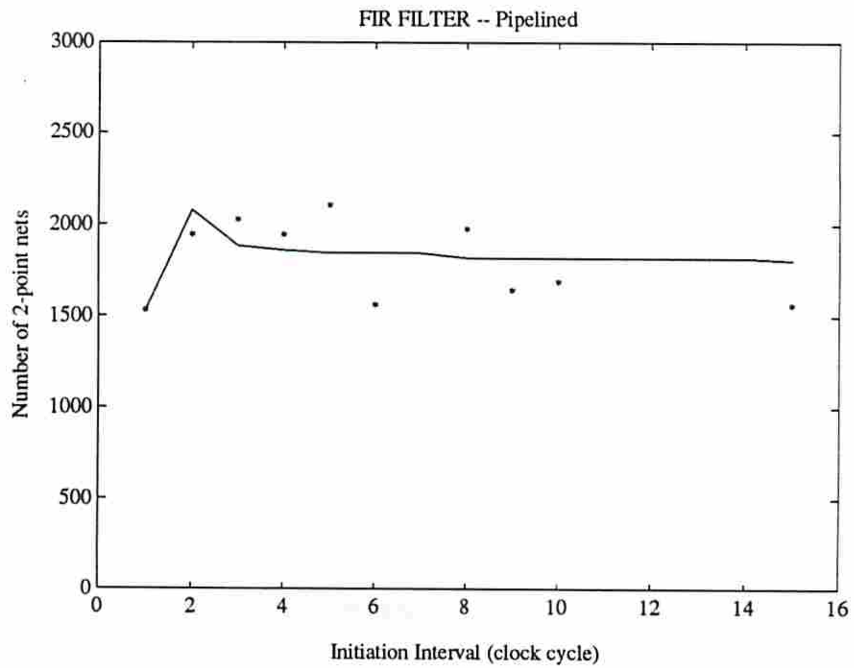


Figure 5.25: The number of 2-point nets for the non-pipelined FIR Filter

5.4 Feasibility Analysis Example

For the feasibility analysis experiment, two experiments were performed with the single-cycle architecture model. In the first experiment, BAD was given a clock cycle time of 3000 ns and no constraints and asked to generate all prediction points for all possible design styles, library configurations, and initiation intervals. The predicted designs are shown in Figure 5.26.

In the second experiment, a set of constraints were given to BAD and it was asked to generate feasible and non-inferior prediction points. The constraints given to BAD were as follows: chip area $\leq$ 137,641 mil^2, performance (initiation interval) $\leq$ 22,000 ns and circuit delay $\leq$ 60,000 ns. There were only two non-inferior prediction points which satisfy the given constraints, as shown in Figure 5.27.

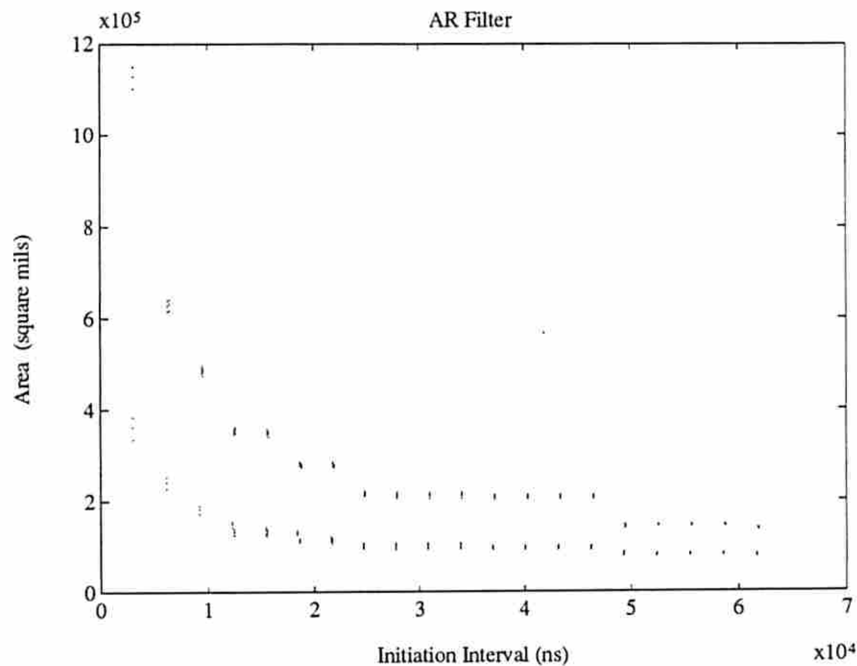


Figure 5.26: All designs predicted for the AR Filter

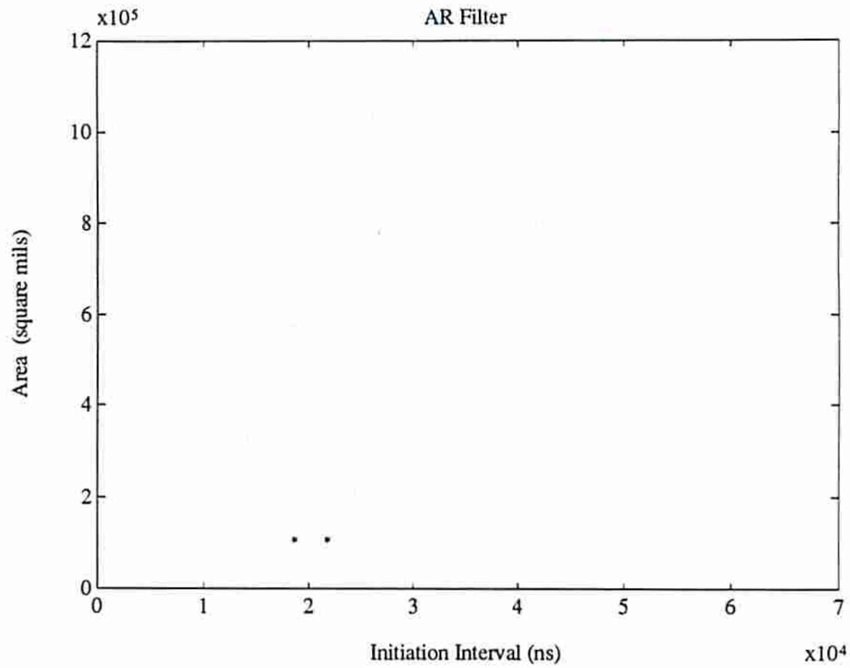


Figure 5.27: Feasible Predicted Designs for the AR Filter

5.5 Summary

In this chapter, we presented the results of experiments performed to validate the accuracy of the prediction techniques presented in Chapter 4. The validation of prediction techniques have been performed down to RT-Level and PLA layouts. Further validation of the full model (considering the layout) is in our immediate plans. The experiments showed that the prediction techniques are fairly accurate for both global characteristics (e.g RTL area) and specific characteristics (e.g. operator area, register area and multiplexer area).

Chapter 6

Behavioral Partitioning of Digital System Specifications

6.1 Introduction

Most digital designs are too large to fit on a single chip. At present, much chip partitioning is performed manually by designers. The time and the way partitioning is performed are quite important since the overall synthesis process consists of several interacting tradeoffs and design decisions. The quality of synthesis results is highly sensitive to these design decisions. Partitioning can be performed at different phases of the synthesis process; after logic synthesis, after behavioral synthesis but before logic synthesis (at RT-level), and finally before behavioral synthesis.

Since functional delays have become comparable to or less than off-chip communication delays as feature sizes decrease, the assumption that designs can be synthesized as single-chip designs and can then be partitioned onto multiple chips without too much penalty on timing rarely holds. After a design is synthesized as a single-chip design, it may not be possible to partition the design onto multiple chips while satisfying the constraints. In such a case, the synthesis process would somehow have to be repeated to synthesize another implementation whose subsequent partitioning would be feasible or some modifications in the synthesized

design would have to be made using iterative improvement, at the risk of producing an inferior design. This iterative improvement would undo some previously made synthesis decisions and would try to re-synthesize parts of the design to find a design whose partitioning would be feasible. This problem is very hard and therefore a trial-and-error method is generally used in such cases. Even if it may be possible to partition some designs after synthesis while satisfying the constraints, such multi-chip implementations would likely be inferior. This is because the synthesis tools perform optimizations while assuming a single-chip implementation as the target design, which translates into making many design decisions without the knowledge of the partition boundaries.

Alternatively, if the partitioning is performed before behavioral synthesis, the synthesis tools can perform optimizations which are suitable for the multi-chip target design. However, if the partitioning is performed at the behavioral level, the major problem is to be able to measure the *quality* of the partitioning and to find out if the partitioning will be feasible. Unfortunately, there is no known quality measuring technique except the final design characteristics, determining which would involve a complete synthesis process. As a result, partitioning before behavioral synthesis *without considering the implementation information* might severely suffer from long iteration times to reach feasible solutions. Therefore, we believe that implementation information has to be somehow incorporated into behavioral partitioning to provide means to measure the quality of a partitioning.

At the RT or logic level it is relatively easy to use accurate measures to evaluate the *quality* of the partitioning because the subsequent changes to the global *structure* after partitioning are minimal. On the other hand, at the behavioral level, global information about the final design characteristics useful to partitioning is non-existent since no structure exists. This further complicates the partitioning at the behavioral level due the fact that not only good and efficient partitioning techniques have to be found, but also the quality of the evaluation method has to be established.

In this chapter, a constraint-driven partitioning methodology will be presented. This methodology can be used to determine the feasibility of tentative partitions at the behavioral level by using accurate prediction methods. A prototype software package, CHOP which uses these ideas has been implemented.

6.2 Overview of CHOP

CHOP is part of the USC (Unified System Construction) Project [PKPW91]. The USC Project is being built on top of the former ADAM System [JKMP89]. A simplified block diagram of the USC Project is shown in Figure 6.1. CHOP has been modified since reported in [KP90c, KP90b, KP91].

CHOP allows the designer to interactively create and modify behavioral partitions and CHOP determines the feasibility of each tentative partitioning using fast prediction techniques. This technique, *evaluation by prediction*, does not suffer from long iteration times of the *evaluation by synthesis* technique. The partitioning is not evaluated in an abstract space; the approach uses actual physical design parameters, constraints and goals.

The goal of CHOP is to predict the best reachable global implementation(s) of a given behavioral partitioning onto a completely specified set of chips subject to performance and delay constraints. This global implementation prediction provides excellent feedback to the designer for interactive partitioning or provides a dependable partition evaluation mechanism to any higher-level automatic partitioning technique.

Predicting the final characteristics of a partitioned design at the behavioral level is very difficult. Given the fact that behavioral synthesis is highly flexible and can create several implementations for each individual partition with very different characteristics, a very large number of global designs can be constructed from these implementations. Even the process of constructing global designs from partition implementations is a constrained discrete optimization problem. In addition to chip area constraints, there are other constraints including chip

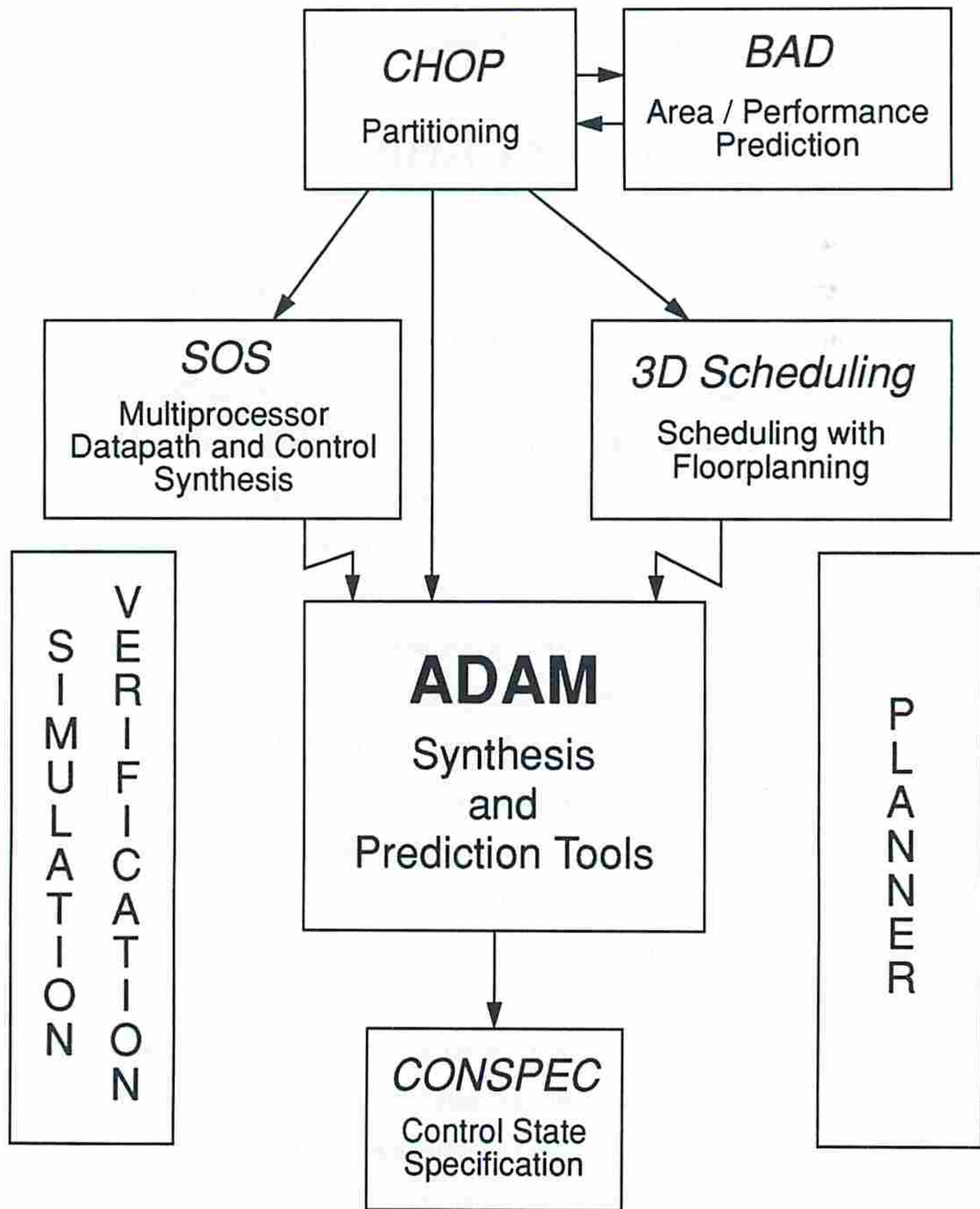


Figure 6.1: The block diagram of the USC (Unified System Construction) Project

pin limitations, memory bandwidths, and performance matching between the partitions. During the integration of partitions into multi-chip designs, there are several types of area and delay overhead (buffers, multiplexing, control) for the global design.

The prediction mechanism in the proposed partitioning evaluation method is comprehensive. Detailed prediction techniques addressing most digital design aspects are included. The prediction techniques include: design style selection (functional-pipelining/non-pipelined), module selection, operator, register, multiplexer, wiring, and controller areas/delays, memory bandwidths, off-chip communication bandwidths, performance matching between independent data paths, data buffering, and a distributed control mechanism.

6.3 Assumptions

CHOP's operation is highly dependent on the prediction techniques used. BAD is embedded in CHOP to provide the predicted implementations for the partitions. Therefore, CHOP inherits all of the assumptions of BAD which have been explained in Section 4.3. In addition to the clocking scheme assumed by BAD, CHOP assumes that there will be a second clock to be used in data transfers among the chips. The assumed clocking scheme will be explained in more detail in the next section.

6.4 Inputs

The input data required for CHOP can be summarized as follows: the behavioral specification in the form of a data flow graph, a library of components, the chip set onto which the design is to be partitioned, on and off chip memory modules to be used and assignments of memory modules to chips, partitions and assignments of partitions to chips, tentative data path and chip-to-chip data-transfer clock cycle times, the architecture style, the feasibility criteria, and the design parameters.

The library generally consists of more than one component which can implement each operation type. It is assumed that the memory hierarchy is designed prior to partitioning although, in practice, designers interleave iterations of memory and behavior partitioning, a step we intend to automate in the future. The chip-set information is in the form of actual chip packages to be used. The information about each chip includes the dimensions of the project area and the pin count of the chip, pad delays, and I/O pad area. The delays due to off-chip wiring are currently assumed to be lumped into pad delays. The architecture style, which can have single-cycle or multi-cycle operations, is to be compatible with the architecture style of the synthesis tools which will later be used to implement the hardware.

6.4.1 Clocking Scheme

Due to the fact that the data processing rate can be different from the chip-to-chip data transfer rate, we assume two separate clocks for data path and data transfer in our prediction mechanism. Using two clocks is more advantageous than using a single clock. If a single clock is used, and if the clock is too slow for either data processing or off-chip data transfer, then the inefficiency due to discretization can be very high. On the other hand, if the clock is too fast for either data processing or off-chip data transfer, then the number of states in the controllers becomes large, introducing long control delays into the clock cycle. Therefore, we believe that using two clocks, each of which is customized for the data processing or off-chip data transfer, is the best way. But, in order to keep design complexity at a moderate level, both clocks in our model are to be synchronized with cycle times which are integer multiples of a major clock cycle time.

Let c be the major clock cycle time. Then, the data path clock cycle time, c_{dp} is defined as

$$c_{dp} = c\,k_{dp} \tag{6.1}$$

where k_{dp} is a positive integer which represents the ratio of data path clock cycle time to major clock cycle time. Similarly, the data transfer clock cycle time, c_{xfer} is defined as

$$c_{xfer} = c\,k_{xfer} \tag{6.2}$$

where k_{xfer} is a positive integer which represents the ratio of data-transfer clock cycle time to major clock cycle time. The ratios of data path and data-transfer clock cycle times to the major clock cycle time are always preserved. Given c, k_{dp} and k_{xfer}, a more accurate clock cycle is estimated after the predictions are performed by taking into account miscellaneous predicted delays introduced into the clock cycle.

6.5 Partitioning Approach

The *evaluation by prediction* method used in CHOP is based on determining a number of possible implementations for each partition, followed by the construction of the best global design from these partition implementations. During this process, several system-level issues including feasibility checks and overhead to integrate partitions as a system are taken into account.

The predictions are performed in two phases. The first phase of predictions is used to determine several possible implementations of the partitions, and the second phase of predictions is performed during the construction of the global design to predict the overheads in integrating the partition implementations. In this way, the complexity of directly predicting the global design is reduced. The overall operation of CHOP is shown in Figure 6.2.

The partitioning model allows multi-chip implementations with multiple partitions per chip. It is sometimes advantageous to have multiple partitions on a

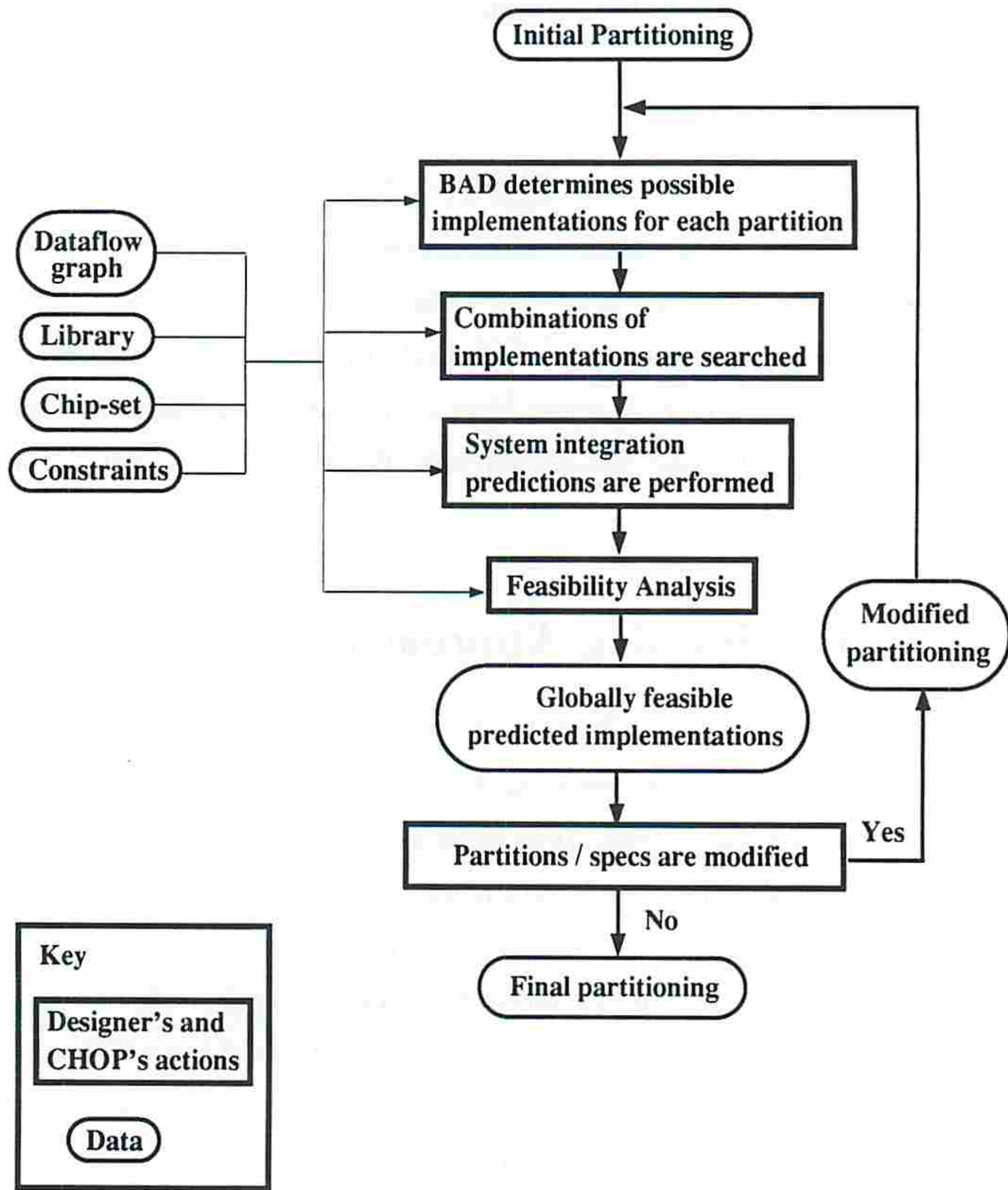


Figure 6.2: The overall operation of CHOP

single chip. Lagnese and Thomas showed that it is likely to reduce the routing area overhead by dividing a design into partitions without sharing any hardware between the partitions [LT91].

The partitioning model also allows non-uniform partitioning of the behavior onto chips, by allowing for variations in chip sizes and packaging. The main reason for non-uniform partitioning is to provide more flexibility for reaching a low cost multi-chip implementation. This cost is not directly a function of the number of chips or the total area, but rather is a function of costs of actual chips used.

A distributed control is assumed for the partitioned design in which finite state machines communicate with each other. This type of control is advantageous over a centralized controller for several reasons:

1. Each controller is relatively small, therefore, the controller delays are smaller.

2. The execution on the data paths are not directly affected by long off-chip control delays.

3. There is more flexibility to use different implementations for each controller (PLA or random logic) to perform case-specific optimizations.

It is assumed in our model that the chip pins are hard constraints which can not be changed. If the the total bitwidth of the values to be transferred to and/or from a chip is larger than the number of pins, this does not necessarily lead to infeasibility. The flexibility of behavioral synthesis may be used to transfer this data sequentially while executing other parts of the behavior concurrently. Therefore, chip pins are treated as resources.

6.5.1 An example for the partitioning model

The partition model used in CHOP can be best explained by an example. An example tentative partitioning consisting of 5 partitions ($P1-P5$) and 2 memory

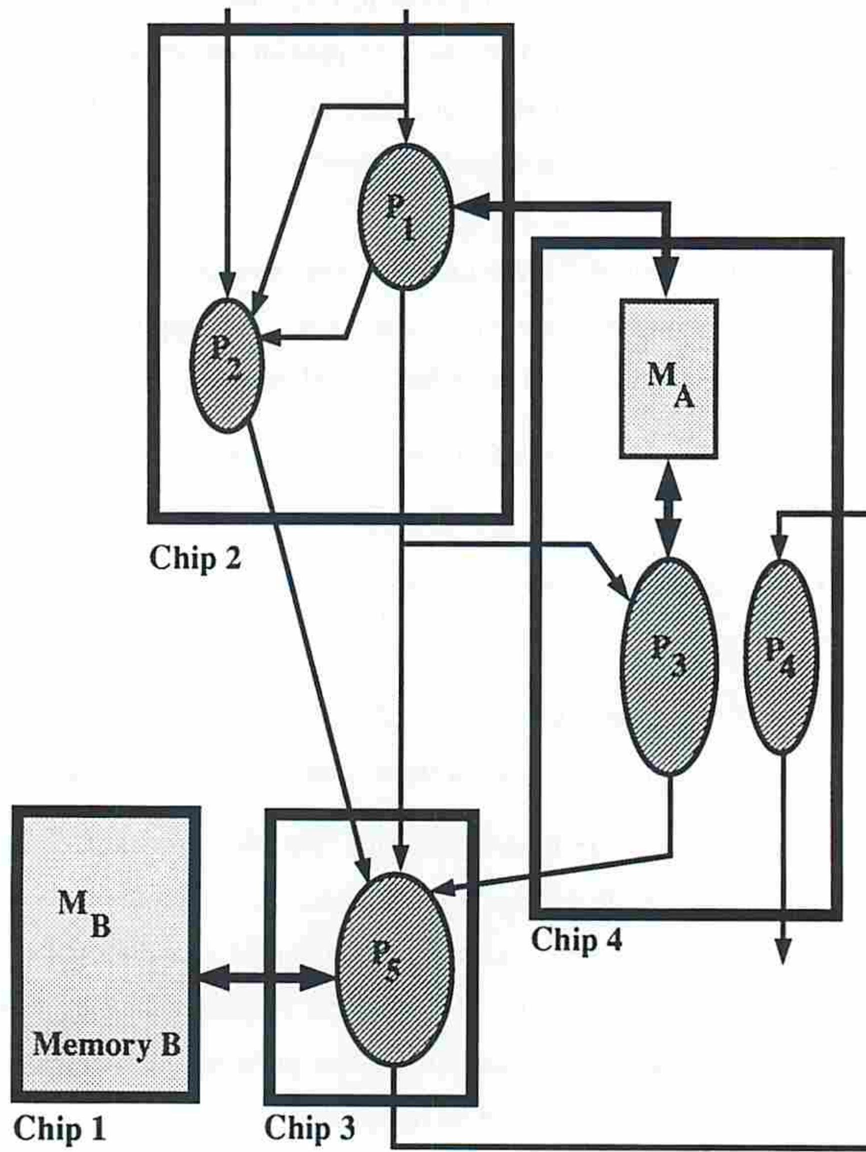


Figure 6.3: An example partitioning

units (M_A and M_B) as a four-chip design is shown in Figure 6.3. It is important to note the following: there can be multiple partitions assigned to a single chip, which may or may not have dependencies on each other. Our predictions assume that these partitions will be synthesized separately and independently. But, it may be possible to synthesize the partitioned design as a single piece when such tools become reality. Memory blocks can be assigned to the same chips as partitions and the use of off-the-shelf memory chips is also allowed.

6.5.1.1 Data Transfer Tasks

The first step in the partitioning process implemented by CHOP is the creation of a task graph. In this task graph, there are two types of tasks; data path partitions, and data transfer tasks. Data transfer tasks are created in order to represent off-chip data transfers as shown in Figure 6.4. This involves determining the manner and the amount of data to be transferred, reserving enough pins for control signals to assure proper communication between distributed controllers and also for other necessary signal pins which are not shared (e.g. select, R/W lines for memory blocks).

To handle the complexity of off-chip value transfers, data transfer tasks are created in such a way that each data transfer task represents several individual value transfers. But, while merging value transfers to form data transfer tasks, special care is taken not to over-constrain the data transfer process by mapping too many value transfers onto a few data transfer tasks. The basic idea is to organize the data transfers in a way that the transfer of values which are not urgently needed by other partitions can be delayed in order to produce efficient task schedules which represent reality as closely as possible.

The creation of data transfer tasks is performed differently for two cases as explained below: the transfer of external inputs into the chips and the transfer of data between the chips.

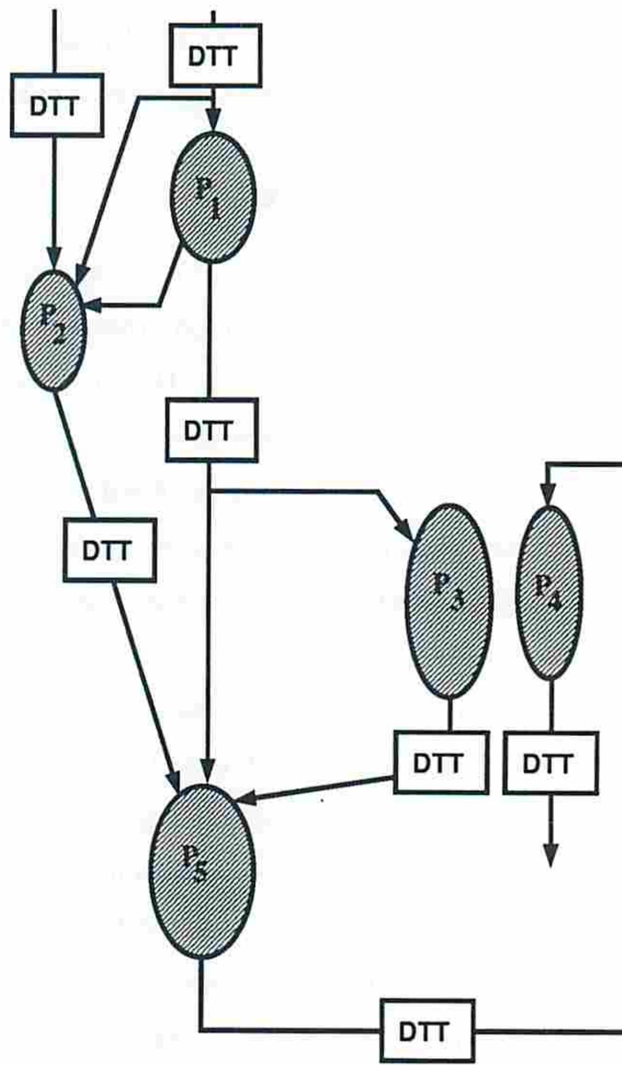


Figure 6.4: The task graph for the example partitioning

Transfer of external inputs into chips

For each chip, the external inputs which are used by the partitions on the chip are grouped into data transfer tasks in such a way that each data transfer task has a *unique set of destination partition(s)*. An example is shown in Figures 6.3 and 6.4. A set of external inputs is only used by partition P_2 and another set of primary inputs is used by both partition P_1 and partition P_2. Then, two data transfer tasks are created to transfer the external inputs into the chip as shown in Figure 6.4.

Transfer of values between chips

For each partition, the values which are transferred off-chip are similarly grouped into data transfer tasks in a way that each such data transfer task would have a *unique set of destination chip(s)*. Each set of values which is transferred to a different set of destination chips is assigned a different data transfer task.

6.5.1.2 The implementation

CHOP assumes that a special purpose hardware unit will be used to implement each task in the task graph. The architectural building blocks used to implement the tasks are shown in Figure 6.5. Data transfer tasks are implemented by data transfer modules, and partition tasks are implemented by processing units. Figure 6.4 shows the task graph created from the example partitioning shown in Figure 6.3. Implementing each task in Figure 6.4 with the corresponding architectural building block from Figure 6.5 will result in a design as shown in Figure 6.6.

In reality, many of these data transfer modules can be merged during an actual implementation. Extensive coarse grain pipelining among the architectural building blocks is assumed to exist in the final implementation of the design to obtain good performance/delay characteristics.

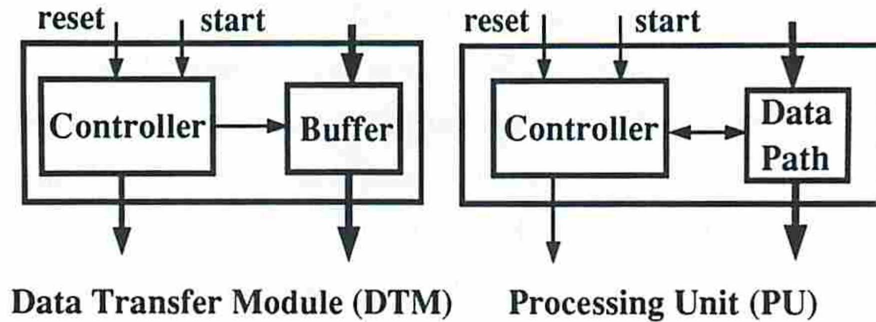


Figure 6.5: The architectural building blocks

6.5.2 Predicting partition implementations

After the creation of a task graph, CHOP first predicts possible individual implementations for each partition by calling BAD to generate predictions for several implementations using detailed prediction techniques. The results from BAD include completely specified characteristics (area, performance, delay, memory and pin bandwidth requirements), operator, register and multiplexer prediction, PLA-based controller area, and standard-cell routing area, as well as the delays introduced to the clock cycle (register, multiplexer, wiring and PLA delays). The details of how these predictions are performed have been explained in Chapter 4.

BAD is instructed to eliminate any infeasible predicted partition implementations before passing them to CHOP. The feasibility of performance, delay and area characteristics of the predicted implementations are checked. This feasibility check only eliminates predicted partition implementations which would not be feasible even as stand alone chips, thereby reducing the number of prediction points to be processed by CHOP. This feasibility check does not consider that several partitions can be on the same chip and/or that partitions would be executed in sequence resulting in longer overall delays. As a result, no good designs are eliminated but some partition implementations which would lead to globally infeasible designs are still retained at this point.

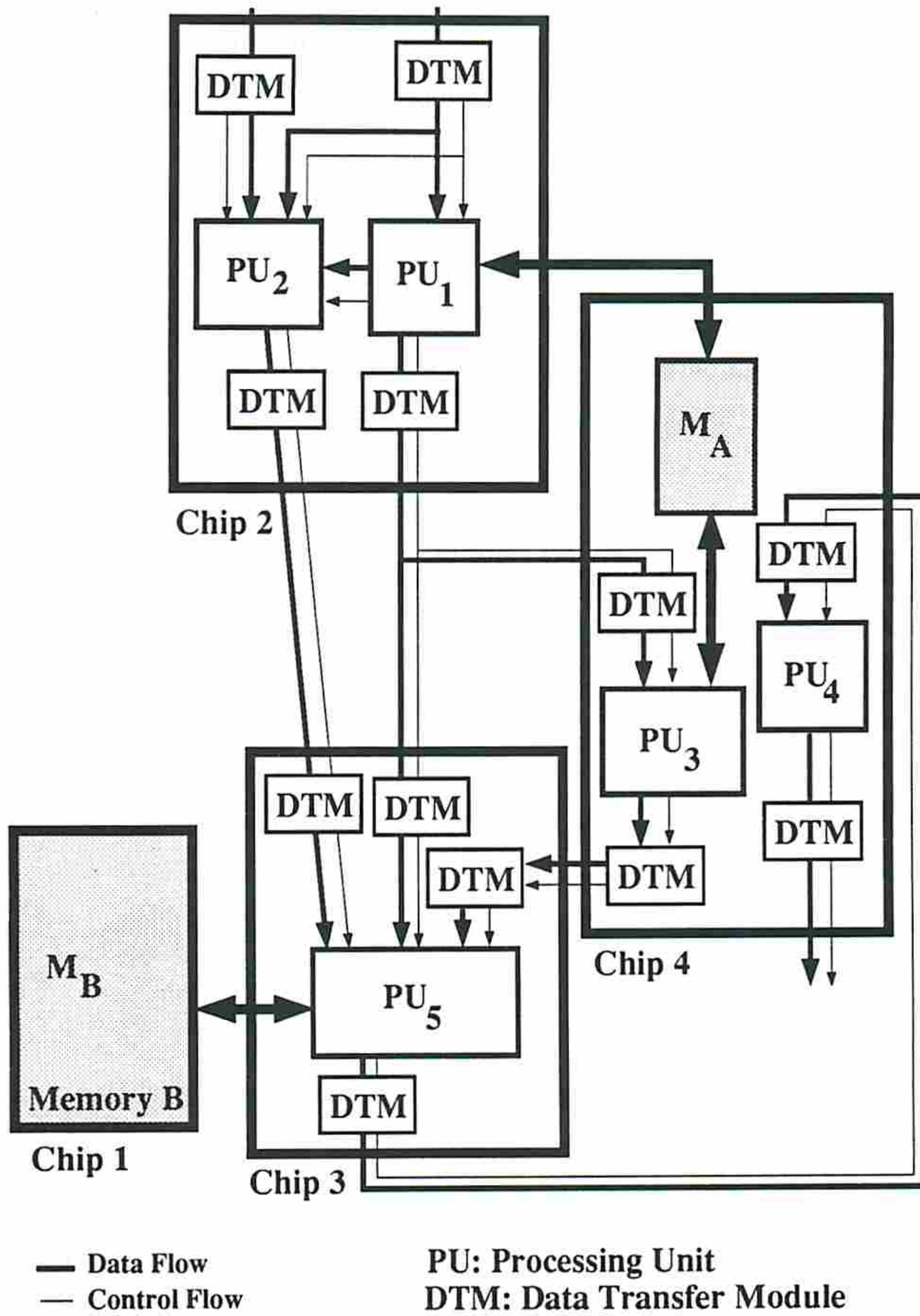


Figure 6.6: The final implementation of the example partitioning

6.5.2.1 Further elimination of predicted designs

After the simple feasibility check is performed by BAD, a more elaborate analysis can be performed to eliminate more predicted designs which cannot lead to a feasible global design by taking into account the combined area and delay characteristics of partitions. Further elimination of such predicted designs do not change the results of the partitioning but contribute to the reduction of the run-time. The analysis which is currently implemented in CHOP only considers the area characteristics. The global delay analysis which will be given by Theorem 6.5.2.15 has not been implemented yet.

Area Analysis

A separate analysis of area is performed for each chip. Let $A_{i,min}$ be the area of the minimum area implementation of partition i. Then, the following theorem holds.

Theorem 6.5.2.11 *Let i vary over the partitions on a chip. The implementation area A_k of partition k to be used in a feasible global design is upper bounded as follows:*

$$A_k < \text{Chip Area Constraint} - \sum_{i \neq k} A_{i,min}$$

Proof: The proof is trivial. The upper bound area of partition k corresponds to the lower bound on the total area of other partitions on the same chip since the area constraint is fixed. The reason for having $<$ in the relation instead of $\leq$ is that the area of the chip will also be consumed by data transfer modules and chip-pin multiplexing. Therefore, a non-zero chip area will always be consumed by logic other than the partitions themselves. □

Global Delay Analysis

The delay analysis is performed globally by taking into account the combined delay characteristics of the partition implementations. First, a set of definitions and theorems will be given which will be used in Theorem 6.5.2.15. This theorem gives the conditions for a partition implementation to be too slow to be included in any global design satisfying the design constraints.

Definition 6.5.2.12 *A measure, R, is a regular (non-decreasing) measure with respect to $C_1, C_2, \ldots, C_n$ such that if*
$C_1 \leq C_1', C_2 \leq C_2', \ldots, C_n \leq C_n'$ *then* $R(C_1, C_2, \ldots, C_n) \leq R(C_1', C_2', \ldots, C_n')$.

Theorem 6.5.2.12 *A regular measure is transitive. If C is a regular measure with respect to $C_1, C_2, \ldots, C_n$, and if $C_1, C_2, \ldots, C_n$ are regular measures with respect to $D_1, D_2, \ldots, D_m$, then C is a regular measure with respect to $D_1, D_2, \ldots, D_m$.*

Proof: The proof follows from Definition 6.5.2.12 and is by substitution. □

Lemma 6.5.2.13 *Given a directed acyclic graph $G(P, E)$ where P is a set of partitions and E is a set of edges representing data dependencies between the partitions. Let C_i be the minimum completion time for partition i. C_i is regular measure with respect to the partition delays it is a function of.*

Proof: For each edge in E, originating from partition j and terminating at partition i, the completion time of partition i has to obey the following inequality in order to satisfy the data dependencies:

$$C_i \geq C_j + D_i$$

where D_i is the delay of partition i. If there are two edges terminating at partition i and originating from, say, partitions j and k, then the completion time of

partition i has to obey two such inequalities. The *minimum* completion time of partition i is then apparently given as

$$C_i = \max(C_j + D_i, C_k + D_i) \tag{6.3}$$

This can be generalized when greater than two edges terminate at a partition. Now, we will show that C_i as given above is a non-decreasing function of partition delays. C_i is expressed by nested max and + functions of positive partition delays, in a similar way to the one shown in Equation 6.3. Therefore, C_i is a non-decreasing function of delays of partitions from which a path exists to partition i. Assume that an arbitrary partition delay, D_z has been changed to D'_z such that $D'_z \geq D_z$. If there are no paths from partition z to partition i, then C_i will remain unchanged (not decreased). If there is a path from partition z to partition i, the completion time for partition i may remain unchanged or may change in a non-decreasing fashion as explained above. Therefore, the minimum completion time for partition i is a regular measure with respect to the partition delays. □

Definition 6.5.2.13 *The critical path delay is defined as*

$$C(G) = \max_i(C_i) \tag{6.4}$$

where i varies over partitions.

Theorem 6.5.2.14 *The critical path delay is a regular measure with respect to the partition delays.*

Proof: The critical path delay is regular measure with respect to minimum completion times of partitions as it can be seen from Equation 6.4. Each C_i is a regular measure with respect to the partition delays as shown by Lemma 6.5.2.13. Therefore, using Theorem 6.5.2.12, the critical path delay is regular measure with respect to the partition delays. □

Theorem 6.5.2.15 *Let $D_{i,k}$ be the delay of the kth implementation of partition i. Let $D_{i,min}$ be the delay of the minimum delay implementation of partition i, which is given as*

$$D_{i,min} = \min_k(D_{i,k})$$

Also let $CP_{i,k}$ be the critical path delay for the global design using $D_{i,min}$ for all partitions except partition i, and $D_{i,k}$ for partition i. If

$$CP_{i,i_k} \geq \text{Global Delay Constraint} \tag{6.5}$$

then, the kth implementation of partition i can not be used in any feasible global design.

Proof: Since the critical path delay is a *regular* measure with respect to the partition delays, the minimum critical path delay with the delay of partition i being $D_{i,k}$ occurs when all other partition delays are the minimum possible ($D_{i,min}$). If this minimum critical path delay does not satisfy the global delay constraint, then constructing any other global design using different implementations for partitions other than partition i would even have a longer critical path delay, therefore, would not be feasible either. □

Theorems 6.5.2.11 and 6.5.2.15 state that certain predicted implementations of partitions cannot be used to produce a feasible global design. But, even partition implementations which satisfy the condition stated in Theorems 6.5.2.11 and 6.5.2.15 might not be used in any feasible global design since partition areas and delays only constitute a lower bound on the used chip area and the system delay, respectively. When the areas and delays of data transfer modules are included, these partition implementations may lead to infeasible global designs. Therefore, Theorems 6.5.2.11 and 6.5.2.15 can only be used to set an upper bound on each partition's implementation area and delay to eliminate some partition implementations and to reduce the number of partition implementations to be processed later.

6.5.3 Constructive predictions for the global design

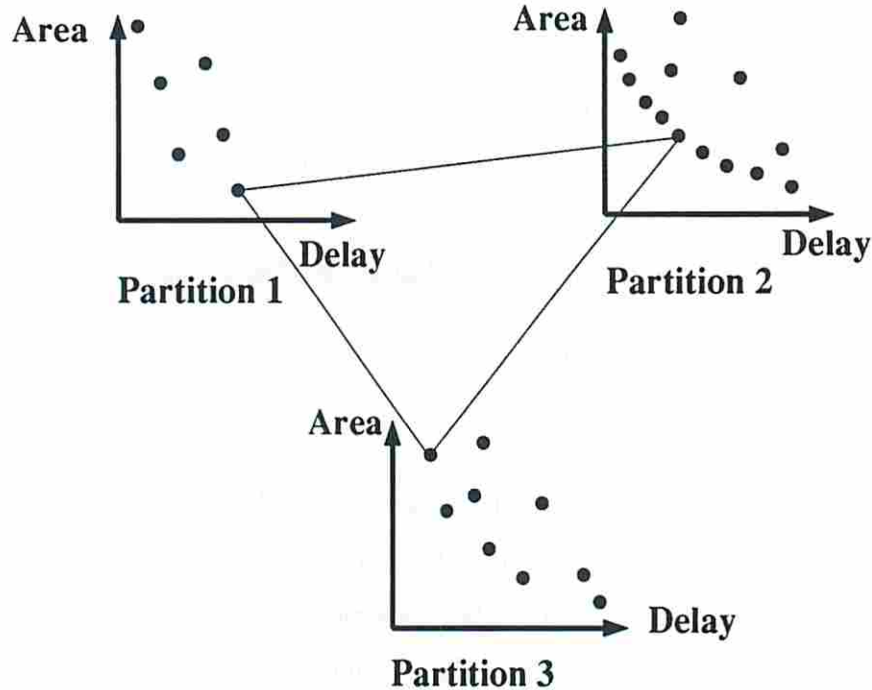


Figure 6.7: Selection of partition implementations

In order to predict the best global design, a constructive approach is used. Given the fact that there exist several implementations for each partition (with different characteristics) and global designs are constructed from these partition implementations, we must select an implementation for each partition (as shown for a 3-partition design in Figure 6.7) in order to satisfy the global design constraints and to optimize a design goal. The problem is similar to the module selection problem in behavioral synthesis. But, it involves several hard constraints (e.g. chip area, performance/delay constraints, pin counts and memory bandwidths) and we feel that it has to be addressed with a more detailed approach than the module selection problem has been addressed. First, the timing of each implemented partition must be compatible with other partitions so that the flow of data between partitions is also feasible. Second, if there are multiple partitions on a single-chip, the combined area characteristics of these partitions should

still satisfy the area constraint of the chip. This invalidates some combinations of partition implementations. Third, in a globally feasible design, faster non-pipelined implementations of partitions can be used along with slower pipelined implementations of others. Finally, pipelining between non-pipelined partitions also needs to be considered.

Prediction for the global design is performed as follows: First, a mechanism for constructing possible global designs from the partition implementations is invoked. This mechanism is a search process which constructs several global designs by selecting one partition implementation for each partition. For each constructed global design, the global design characteristics are predicted and a feasibility analysis is performed on the prediction results to find out if the constraints are satisfied.

Since the characteristics of partition implementations and data transfer modules vary considerably, it is hard to predict the global design characteristics accurately without tying down the characteristics of each partition which make up the global design. Furthermore, even after an implementation for each partition is selected for construction of the global design, one problem remaining is the fact that there could be several good ways of implementing the global design using the selected implementations for the partitions. Due to flexibility in this type of design process, we restrict ourselves as follows: For each global design under consideration, we assume a single way of constructing the global design, given the selected partition implementations and a global initiation interval as explained in Section 6.5.4. Since we consider coarse-grain pipelining at the task level, in this way, we can explore global designs which pipeline the tasks differently for varying initiation intervals. For example, pipelining a set of tasks with a global initiation interval may not be feasible due to area constraints, but pipelining the same tasks with a slower initiation interval may lead to a feasible global design by decreasing the area taken by the data transfer modules. As it will be explained in more detail in Section 6.5.4, the characteristics and the

feasibility of global design are highly dependent not only on the characteristics of partitions used but also the global initiation interval chosen. The initiation interval affects the area/delay characteristics of data transfer modules as well as the feasibility of sharing chip pins and memory blocks.

6.5.3.1 Selecting an Initiation Interval

Given a set of selected implementations for each partition, an arbitrary initiation rate may not be used for the global design. If the global initiation rate is faster than the initiation rate of any partition implementation used in the selection, then that partition implementation cannot process the data arriving at a faster initiation rate than it is designed for. Similarly, if there is any pipelined partition implementation in the selected set, the global initiation interval has to match the initiation interval of that partition implementation since the pipelined data path predictions assume a static schedule with a fixed initiation interval.

The problem of selecting implementations for partitions in order to reach a feasible global partitioning can be solved by exhaustive search. Two techniques using exhaustive search techniques along with a heuristic have been incorporated into CHOP. Even though effective pruning techniques can be used in these exhaustive search techniques, the exhaustive search is still computationally intensive for any design of a realistic size. The enumeration techniques have been incorporated into CHOP as an option to check how well the heuristic performs.

Both exhaustive search techniques enumerate all possible ways of selecting partition implementations. The difference is in deciding which global initiation interval(s) to use for the evaluation. Figure 6.8 shows how the global initiation interval(s) to be used during the evaluation are decided. The first technique predicts the global design for each enumerated case using a single initiation interval. This initiation interval is determined by the slowest partition implementation in the enumerated set. On the other hand, the second technique also tries several slower ways to pipeline the partition implementations in the enumerated set.

Exhaustive Search Technique 1
let l be the global initiation interval.
Begin
Enumerate all possible ways of selecting partition implementations.
for each enumerated case
    let L_i be the initiation interval of the implementation for partition i.
    for each partition implementation in the enumerated set
        $l = \max(l, L_i)$
    if the partitions can be pipelined by l
        Estimate the global design using l.
Report the best global design(s).
End.

Exhaustive Search Technique 2
let l be the global initiation interval.
Begin
Enumerate all possible ways of selecting partition implementations.
for each enumerated case
    let L_i be the initiation interval of the implementation for partition i.
    for each partition implementation in the enumerated set
        $l_{min} = \max(l_{min}, L_i)$
    $l_{max} = \max(l_{min}, \text{performance constraint})$
    for $l = l_{min}$ to l_{max}
        if the partitions can be pipelined by l
            Estimate the global design using l.
Report the best global design(s).
End.

Figure 6.8: The exhaustive search techniques for finding the global design

Obviously, the global designs searched by the second technique supersede the ones searched by the first technique. The first technique, although it performs an enumeration, may not be able to find the best global design the partitioning model allows, therefore must be treated as a heuristic search. Both search techniques have run-time problems.

A faster iterative heuristic has been developed to find a global design with the fastest initiation rate and the shortest system delay while satisfying the constraints. The heuristic starts with the fastest (initiation rate) predicted implementation for each partition and iteratively considers more slower implementations of partitions residing on chips whose area constraints are violated. Selection of slower implementations is done in such a way that the incremental delay in the overall system delay (critical path delay) caused by this selection is the least. This selection method is intended to favor the serialization of off-critical-path partitions (off-critical with respect to the current global design considered). The outline of the heuristic is given in Figure 6.9.

Given a selection of predicted designs (one selected implementation prediction per partition) and a global initiation interval from either selection method, the system integration predictions (predictions of overall design characteristics including data transfer modules, the overhead for the clock cycle, overall system performance and delay) are performed, as described in the next section. System integration overhead is highly dependent on the characteristics of the selected implementations for partitions and the chip-level constraints.

6.5.4 System Integration Overhead

Even though the partition implementations are considered to be complete, there are still tradeoffs and optimizations to be performed while integrating these partitions. Many of these independent partitions do require control over the memory blocks and chip pins. During system integration, the memory blocks and chip pins must be allocated properly to partitions and data transfer tasks in such a

```
Procedure find_feasible_partitioning_implementations
let W_i be a predicted implementation of partition i.
let L_i be the initiation interval of W_i.
let Q be the list of candidate partitions for serialization.
Begin
Sort all W_i for each partition i in increasing order, first for
    the initiation interval, then for the delay, and finally for the area.
for each feasible initiation interval l
    for each partition i
        Initialize W_i to the first (fastest) predicted implementation
            in the sorted prediction list of partition i.
        Advance each W_i until L_i ≥ l or
            W_i is a non-pipelined implementation with L_i ≤ l.
    while TRUE
        if there is not a W_i for any partition or the fastest W_i is too slow for l
            break
        Estimate system integrations using l and W_i for each partition i.
        if the estimation result is feasible
            Record the overall prediction results.
            break
        else
            Set Q to partitions on chips whose area constraint is violated by
                the last system integration prediction.
        Move W_i of each partition forward to skip the prediction points which
            have inferior characteristics and/or are not implementable with l.
        for each partition i in Q
            Move W_i forward (serialize) in partitions list.
            Find the expected system delay using the urgency scheduling
                for chip pins and memory blocks.
            Restore the old value of W_i.
            If the urgency schedule returns a feasible schedule
                Record the partition whose serialization results in minimum
                    system delay.
        if a partition has been recorded above
            Serialize the partition (Move W_i one element forward in the list).
        else
            Serialize the partition with the smallest L_i.
            {This move tries to serialize the partitions in a balanced way if
                serialization of any partition can not be preferred over others.}
End.
```

Figure 6.9: The simplified outline for the iterative heuristic

way that the global design would be feasible. Due to this optimization/design process, a global design with very different characteristics may be obtained depending on the characteristics of the partition implementations selected. Since it is not known yet how to predict the global design characteristics directly, the predictions are performed by trying to imitate what a simple synthesis tool would do to integrate the partition implementations.

The global design predictions are performed by first scheduling partition and data transfer tasks, considering the fact that chip pins and memory bandwidth are precious resources (hard constraints), followed by the prediction of data transfer module characteristics.

6.5.4.1 Preprocessing for Task Scheduling

There are two groups of information which need to be extracted in order to perform scheduling:

1. actual hard constraints for chip pins (effective pin counts for data transfer), and

2. delays and pin requirements of data transfer tasks.

When data transfer tasks are created, the first step is to perform some implementation-independent pin allocation for control signals between the distributed controllers and for signals which will not be transmitted through shared pins (e.g. select, R/W lines for memory blocks) as explained in Section 6.5.1.1. This allocation is static and only depends on the partitions, data transfer tasks, memory blocks, and assignments of each. The next step is the implementation-dependent allocation which is performed to ensure that memory blocks and chip pins are properly allocated for the selected partition implementations.

Then the delay (duration) of each data transfer task is predicted. Each such delay is predicted as a function of the amount of data to be transferred and the pin bandwidth available for the data transfer. It is assumed that the

maximum possible bandwidth is used for each data transfer. During bandwidth calculations, the effects of simultaneous off-chip memory access on the pin usage are also taken into account.

The duration of data transfer task i (in terms of the major clock cycle), D_i is calculated as

$$D_i = \lceil \frac{DS_i}{\min\limits_j(BW_j)} \rceil \, k_{xfer} \tag{6.6}$$

where DS_i is the size of data to be transferred by the data transfer task i, BW_j is the effective data transfer bandwidth of chip j and j varies over all chips involved in the data transfer (source and destination chips). $\min\limits_j(BW_j)$ is defined as the required pin bandwidth of data transfer task i. The term within the ceiling function gives the data transfer duration in terms of the data transfer clock cycle. This term is then multiplied by the ratio of the data transfer clock cycle time to the major clock cycle time to obtain the data transfer duration in terms of the major clock cycle time.

6.5.4.2 Task Scheduling

Definition 6.5.4.14 A non-preemptive schedule *is a schedule in which each task, once started, must be completed without any interruption and other tasks may not start to use any of resources allocated to the task before the completion of the task [Fre82].*

Having delays of all tasks (data transfer tasks and partitions), a non-preemptive pipelined urgency scheduling is performed to confirm the sufficient conditions for the feasibility of sharing the data pins of chips and keeping memory accesses to each memory block feasible. The urgency measure is based on the actual critical path delays of tasks and is similar to urgency measures used in [PP88]. The initiation interval given by any of the search techniques mentioned in the previous sections is used during the scheduling process. In this schedule,

each data transfer task or partition waits until its required pin and memory bandwidths are available, and once started, completes without any interruption. Of course, this is a simple assumption. An actual implementation might produce a more complicated pin allocation for the data transfers and hence better performance than CHOP predicts. But, this simple schedule is performed only to predict the overall system delay and characteristics of data transfer modules.

The results of task scheduling are used to predict most system integration overhead. The delay characteristics of the specific global design, the buffers and controllers needed to integrate the partitions, and some other design characteristics are predicted using the resulting task scheduling as explained in the following sections.

6.5.4.3 System Delay Predictions

The lower-bound system delay, SD_{lb} is estimated as the critical path delay through the tasks. The most likely value of the system delay, SD_{ml} is predicted as the completion time of the tentative task schedule generated. For maximum system delay predictions, the effects of memory block assignments are not considered yet. It is assumed that only data transfers will cause additional delays for the completion time of tasks. Let D_i be the delay of task i in the task graph, for both partitions and data transfer tasks. Then, the maximum system delay is estimated as the minimum of two estimates

$$SD_{ub} = \min(CP_p + l \times \nu, \sum_i D_i) \tag{6.7}$$

where CP_p is the critical path delay through partitions without considering data transfer tasks, l is the initiation interval considered, and ν is the maximum number of connected chips in the design. The first estimate models the delays caused by sharing chip pins. Assuming that the maximum possible pin bandwidth is used for each data transfer, the maximum delay caused by a single chip is estimated as the initiation interval considered for the implementation since longer

delays would cause data clashes on chip pins. The second upper bound estimate models the serial execution of all tasks. Both these estimated upper bounds can be quite loose; therefore the minimum of these is used.

6.5.4.4 Predictions for Data Transfer Modules

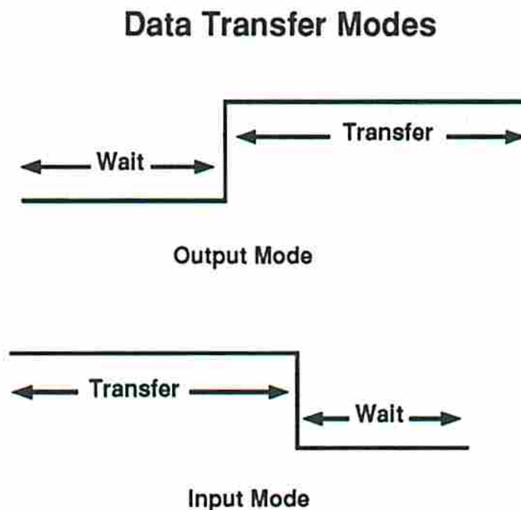


Figure 6.10: The Modes of Operation for Data Transfer Tasks

The resulting task schedule is used in determining the characteristics of data transfer modules. For each data transfer task, one data transfer module has to be placed on each chip involved in the data transfer (one in the output mode, the rest in the input mode). The operation modes of data transfer modules are shown in Figure 6.10. In the output mode, there may be a wait time until enough pins are available for the data transfer, followed by the data transfer. In the input mode, there is a data transfer which may be followed by a wait time to hold the data until the destination partition can accept the data. Therefore, each data transfer module will have its own wait time, W, a data transfer time, D, and will handle the transfer of data with data size DS. For each pair of data transfer modules (one in the input, the other in the input mode) involved in data

transfer, the data transfer time is the same but the wait times may differ. The wait time of each data transfer module is extracted from the task schedule.

Although individual pipelined and non-pipelined partition implementations may co-exist in the final design, the overall design is always assumed to be pipelined. Therefore, the wait time for a data transfer task could be longer than the initiation interval of the design resulting in excessive data buffering. On the other hand, the data transfer duration for each data transfer task cannot be longer than the initiation interval, l since it is assumed that the maximum bandwidth of the chips is used for data transfers and longer data transfer delays would cause data clashes on the pins (Pin counts are hard constraints which cannot be changed by CHOP).

Register Area Prediction:

The sufficient buffer size, B needed for each data transfer module is

$$B = DS \times \lceil \frac{W + D}{l} \rceil \tag{6.8}$$

where l is the initiation interval considered. Buffers between partitions which are on the same chip are considered similarly, except that there is no off-chip data transfer involved.

Control Prediction

The data transfer module controller should be active from the time there is a start signal for new data transfer until the time the data transfer module relinquishes control of the data to the destination partition(s). The wait and data transfer times for each data transfer module are used to predict the number of inputs, outputs and product terms of an *abstract* PLA to control the data transfer, from which PLA size and delay are predicted by the same methods used in BAD.

The number of product terms is estimated as the number of states the PLA-based controller will go through as shown in Equation 6.9.

$$\text{Product Terms} = W + D \qquad (6.9)$$

The inputs to the PLA consists of 3 fields: state bits, reset and start inputs. The number of bits required to represent states of a finite state machine with binary state coding is proportional to the logarithm of number of states. The total number of input bits of the PLA is estimated as shown in Equation 6.10.

$$\text{PLA Inputs} = \lceil \log_2(W + D) \rceil + 2 \qquad (6.10)$$

The output of the PLA consists of 2 fields: state bits and control bits to control registers and multiplexers in the data path. One output is reserved for starting other data transfer modules or partition implementations. It is assumed that the state bits would be sufficient to control the registers and multiplexers, at most with the help of some additional logic. The area of the additional logic gates is generally a second order effect, and is omitted here. The total number of PLA output bits is estimated as shown in Equation 6.11.

$$\text{PLA Outputs} = \lceil \log_2(W + D) \rceil + 1 \qquad (6.11)$$

The state bit outputs of the PLA have to be stored. The registers to store $\log_2(W + X)$ state bits are also included in the predictions.

Once the abstract PLA characteristics are predicted, the techniques discussed in Chapter 4 are used to predict the actual area and the delay of each PLA.

6.5.4.5 Pin Multiplexing Prediction

It is assumed in predictions that all data pins are I/O pins. Off-chip accesses are required due to

1. off-chip memory accesses, and
2. off-chip data transfers.

Given a selected implementation for each partition, the number of off-chip memory ports (and therefore address and data lines) used by each partition implementation is known. The total bitwidth of values being transferred off-chip is also available from the partitioning information.

Let O_i be the total number of bits to be transferred off chip i. Then, the amount of 2-to-1 multiplexers required to share γ_i unreserved data pins of chip i is estimated as follows:

$$muxcnt_{pin} = \max(O_i - \gamma_i, 0)$$

If there are off-chip memory accesses, then the multiplexing introduces delays into both data path and data transfer clocks. Therefore, an estimated delay is directly added to the clock cycle time as follows:

$$muxdelay_{pin} = \frac{\lceil log_2 \frac{O_i}{\gamma_i} \rceil \times \text{2-to-1 mux delay}}{\max(k_{xfer}, k_{dp})}$$

If pins are used only for data transfers, then an estimated delay is directly added to the data transfer clock as follows;

$$muxdelay_{pin} = \frac{\lceil log_2 \frac{O_i}{\gamma_i} \rceil \times \text{2-to-1 mux delay}}{k_{xfer}}$$

6.5.4.6 Wiring Estimation

The additional wiring area needed for each data transfer module is not currently modeled. But, on the other hand, wire delays introduced into the data transfer clock due to data transfers are predicted in a similar way to wire delay predictions for individual partition implementations. The minimum wire delay is given by the delay through a zero-length wire with no fanout as $d(0,1)$. Let the maximum estimated wire length used for system integration, max_wl be $\sqrt{\text{chip area}}$. Then, the maximum and average wire delays introduced into data transfer clock cycle are estimated as $d(max_wl, 2)$ and $d(max_wl/2, 1.5)$, respectively.

6.5.4.7 Clock Cycle Time Estimation

The predicted delay characteristics of processing units and data transfer modules are used to estimate the clock cycle time. The tentative clock cycle time is entirely reserved for functional delays. The major clock cycle time is adjusted so that data path and data transfer clock cycle times are sufficiently long to accommodate additional delays introduced into the data path and data transfer clock cycles.

Let d_{dp} be the sum of delays introduced into the data path clock due to register propagation, multiplexer, PLA and wiring delays. Also let d_{xfer} be the sum of delays introduced into the data transfer clock due to register propagation, pin-multiplexing, control PLA, wiring, packaging, pad, and chip-to-chip delays.

Given k_{dp}, k_{xfer} d_{dp} and d_{xfer}, the adjusted clock cycle, c' is estimated as follows:

$$c' = \max\left(c + \frac{d_{dp}}{k_{dp}}, \frac{d_{xfer}}{k_{xfer}}\right) \tag{6.12}$$

6.6 Partitioning Modification

After the feasibility of a partitioning is checked, the partitioning process may be continued as the partitioning and constraints are modified based on the feedback from the CHOP partitioner. Modifications can be grouped into four major types. Modifications in behavioral partitions and the memory hierarchy (memory partitioning) are types of modifications which change the partitioning directly. Modifications in the target chip set may be performed in 2 ways; Modifications in the characteristics of individual chips (changing the pin count, or the project area) affect the area and pin count constraints on the global design. Partitioning the design onto a different number of chips affects the partitioning as well the area and pin count constraints. Finally, timing constraints can be modified directly.

Behavioral Partitions: The changes can be as simple as operation migrations from partition to partition, or migration of partitions from chip to chip. It is also possible to decrease or increase the size of partitions and/or change the partitioning completely. The granularity of partitions is a tradeoff issue by itself because too fine granularity as well as too coarse granularity can reduce the design quality.

Memory blocks: The hierarchy of the memory blocks as well as the assignments of the memory blocks can be modified to possibly decrease the number of off-chip memory accesses. The memory blocks are also resources which may constrain the maximum parallelism allowed in the design. These may affect the final design characteristics considerably. The design of memory partitioning and behavioral partitioning are highly interrelated and unless both can be performed simultaneously, interleaving memory and behavioral partitioning may be required.

Past work coupling the memory synthesis with behavioral synthesis focuses on the synthesis of the memory architecture by merging memory modules after some of the design process is performed. In [CS90], Chen and Sobelman propose a method for merging isolated registers into single-port and/or multi-port memory modules. In addition to merging registers into multi-port memory modules, Balakrishnan et al. also consider the effect of memory synthesis on the interconnect of the design [BMB+88]. Both of the methods mentioned above correspond to the storage of values in a design. A more recent approach for memory architecture synthesis for high speed digital signal processing (DSP) applications taking into account more tradeoffs is proposed by Lippens et al. [LvMvdW+91]. They address memory merging versus interconnect cost tradeoffs and tradeoffs in address allocation and generation (relative, counter, absolute addressing).

Unfortunately, no published work to our knowledge determines the effects of higher-level design decisions like memory hierarchy design for architectural

memory modules which are in the behavioral specification. Therefore, the memory and the behavioral partitioning are currently performed in an interleaved manner, CHOP supplying the feedback information.

Target chip set: The feasibility of behavioral partitioning is also highly dependent on the target chip set. Less area available for the design generally translates into slower, more serial designs. By the same token, the fewer the number of pins available for data transfers, the longer the data transfers take. Modifications in target chip-set characteristics affect the feasibility of the behavioral partitioning. Target chip characteristics generally dictate the overall manufacturing cost of the design.

Timing constraints: High performance and tight delay constraints translate into more parallelism in the feasible designs which are larger in size. High performance constraints also cause the I/O pin usage to increase. Relaxing these constraints may increase the number of feasible designs since the feasibility criteria change.

6.7 Feasibility Analysis

The feasibility of the partitioning is checked when the individual characteristics of PUs and data transfer modules are available. The feasibility criteria are same as the criteria used in BAD as explained in Section 4.2. The PERT modeling technique [LK66] is used to store the prediction results. In the PERT model, each prediction result has 3 values: a lower bound, a most likely and an upper bound value. These values are used to compute an average and a standard deviation for each prediction result. Then, standard statistical methods [Rob85] are used for the feasibility analysis.

Feasibility analysis is performed for each chip area constraint by considering the area taken by PUs, data transfer modules residing on each chip, and

multiplexing to share the data pins of each chip. The performance and delay characteristics of PUs, chip packaging delays, chip-to-chip delays, and controller delays as well as delays of pin multiplexing logic are included in the feasibility analysis of the performance and the system delay.

6.8 Synthesizing the partitioned designs

Virtually all estimated characteristics in CHOP are available to the user or to a higher-level program. This includes BAD's predictions on the partition implementation characteristics (e.g area, throughput and total circuit delay) of the selected partition implementations as well as how to reach those predicted designs. More details on the data available from BAD can be found in Section 4.13. In addition to what is available from BAD, the prediction results for the global design are also available from CHOP in the same format as BAD results. The results include the area/delay characteristics of the data transfer modules (buffers and PLAs), any additional hardware needed (e.g. pin multiplexing), the estimated clock cycle time by taking into account both data path and data transfer delays, and the tentative global schedule of tasks (when each partition/data transfer module starts/ends execution).

Therefore, it is quite easy to drive the behavioral synthesis tools to produce a design with similar characteristics to the predicted design. However, it may be possible to produce designs differently than what CHOP suggests in order to obtain better designs than CHOP predicts.

6.9 Complexity Analysis

The run-time complexity of CHOP depends on

- the complexity of BAD to predict several implementations for partitions,
- some pre-processing,

- the number of predictions generated by BAD to be used by CHOP,
- the complexity of system integration predictions, and
- the amount search performed.

The complexity of BAD on ρ partitions is $\mathcal{O}(\rho m n^2)$ where m is the number of library configurations as explained in Section 4.11 and n is the total number of operations. BAD produces $\mathcal{O}(mn)$ prediction points to be used by CHOP. The complexity of pre-processing is dominated by the creation of the task graph and is $\mathcal{O}(nt + t^2)$ where t is the number of tasks in the task graph.

The complexities of both system integration predictions and task scheduling are $\mathcal{O}(t^2 + tz^2)$ where z is number of possible initiation intervals. Because the complexity of system integration predictions is dominated by the complexity of the task scheduling.

Depending on which search technique is used, the run-time complexity of CHOP varies. The overall run-time complexity of CHOP for each search technique is separately given below.

6.9.1 Exhaustive Search Technique 1

This technique enumerates all possible configurations of partitions, therefore evaluates $\mathcal{O}((mn)^\rho)$ configurations. The overall complexity of CHOP for this search technique is given as

$$\mathcal{O}(\rho m n^2 + nt + t^2 + (mn)^\rho (t^2 + tz^2))$$

which reduces to

$$\mathcal{O}((mn)^\rho (t^2 + tz^2)) \qquad \text{if } \rho > 1$$

6.9.2 Exhaustive Search Technique 2

The difference with this technique and the first exhaustive search technique is that this technique searches several ways to implement the global design for each enumerated partition configuration. The amount of search performed by this technique is characterized as $\mathcal{O}(z(mn)^{\rho})$. The variable is dependent on the performance constraints and the clock cycle time. Then, the overall complexity of CHOP for this search technique is given as

$$\mathcal{O}(\rho mn^2 + nt + t^2 + z(mn)^{\rho}(t^2 + tz^2))$$

which reduces to

$$\mathcal{O}((mn)^{\rho}(t^2 z + tz^3)) \qquad \text{if } \rho > 1$$

6.9.3 Heuristic Search Technique

The heuristic first sorts all prediction points, and this has a complexity of $\mathcal{O}(mn \log mn)$. For each feasible initiation interval, the heuristic can at most evaluate mn partition configurations and can at most create ρmn schedules to drive the search process. As a result, the overall complexity of CHOP for the heuristic search technique becomes

$$\mathcal{O}(\rho mn^2 + nt + t^2 + mn \log mn + z\rho mn(t^2 + tz^2))$$

which reduces to

$$\mathcal{O}(\rho mn^2 + mn \log m + \rho mn(t^2 zt + z^3))$$

6.10 Considering design costs

6.10.1 Background

The cost of a design depend on too many factors to be considered in this thesis. Such a cost analysis involves several disciplines [TR89]. Consideration of only a subset of costs, namely the manufacturing costs, will be discussed here. In a very simplified cost model, the manufacturing cost of a multi-chip ASIC design can be approximated by the costs of chips, costs of terminals and connections, costs of higher-level packaging (e.g. printed circuit boards, thin and thick film substrate-level packaging, and multi-chip modules), and the cost of the assembly [TR89, Cos89], excluding the costs related to non-synthesized parts of the design. The costs of higher-level packaging and connections may not be available since the whole design is generally not designed by the synthesis tools. However, the manufacturing costs only for the chips being designed by the synthesis tools can be approximated with a greater accuracy. Costs of individual chips depend on the die size, the packaging type (fast, medium, slow), and the pin count. Since chip packages are mostly standardized, discrete costs of chips for varying die sizes and pin counts are readily available.

6.10.2 How to use costs in synthesis

Incorporating costs fully into the synthesis process makes the hard synthesis problem even harder by introducing a new dimension into the problem. In its simplest form, design costs can be introduced into synthesis as a way to evaluate possible design alternatives. We believe that system-level partitioning is the best candidate among synthesis tasks for using the manufacturing costs.

The simple cost model can be used to calculate the cost of a multi-chip implementation whose performance and delay characteristics are predicted by CHOP. The following iterative technique can be used to reach low-cost multi-chip implementations which satisfy the performance and the delay constraints after

the single-chip implementation is evaluated to roughly determine the design size. First, the tentative target chip characteristics are decided. Then the behavior is partitioned onto the target chip set using a partitioning program like CHOP. Based on the feedback (the predicted performance and delay characteristics of the partitioning, and the predicted chip area utilizations), the target chip set is modified and the partitioning is repeated.

Another approach to incorporate costs into the system-level partitioning in a similar way would be to determine the chip package requirements for partitions such that the performance and delay constraints are satisfied, rather than to constrain the evaluation mechanism by the already-selected chip package characteristics.

6.11 Extensions to use off-the-shelf data path parts

Off-the-shelf data path parts can also be used in the partitioning with some limitations. By an *off-the-shelf data path part*, we mean a *pre-designed macro cell.* The current implementation of CHOP does not yet allow the use of off-the-shelf or pre-designed *chips* except memory chips. We differentiate between two ways of supporting off-the-shelf data path parts.

In the first case, only a group of operations are to be implemented by an off-the-shelf data path part. This case is handled by moving the abstraction level higher in the operation/operator specifications. These operations are replaced by a composite operation and the composite operation is given a new operation type. The off-the-shelf data path part is then entered as a regular library component (operator) of the same type. Currently, CHOP's structure allows this process to be performed manually on the specifications. Minor changes would be required in CHOP to allow this type of off-the-shelf data path parts with minimal user effort.

If a whole partition is to be implemented by an off-the-shelf data path, then rather than calling BAD to generate the predicted implementations of the partition, the characteristics of the *off-the-shelf data path part* can simply be used. CHOP can easily be modified for this purpose.

6.12 Limitations of CHOP

CHOP is the first behavioral partitioning tool which can take physical design characteristics into account for partitioning of the behavioral specifications. In this section, the limitations of CHOP's approach will be discussed.

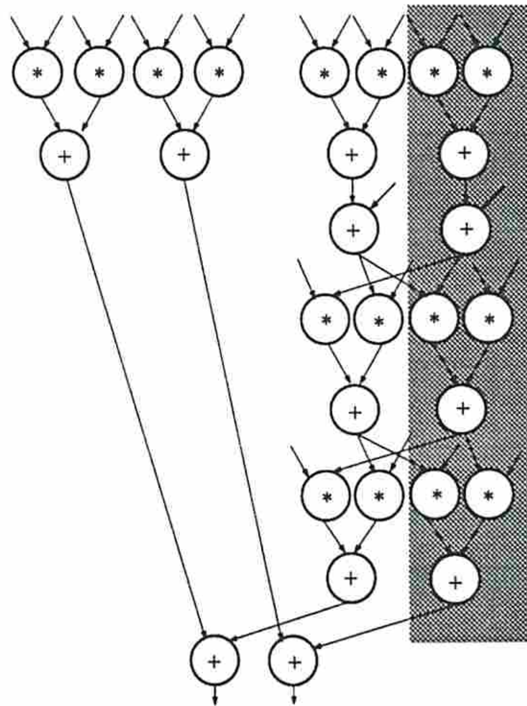


Figure 6.11: The partitioning with dependency loop

There are currently some restrictions on the topology of the partitioning to ensure the accuracy of the predictions. This is due to the fact that most scheduling approaches assume that all of the inputs are simultaneously available for

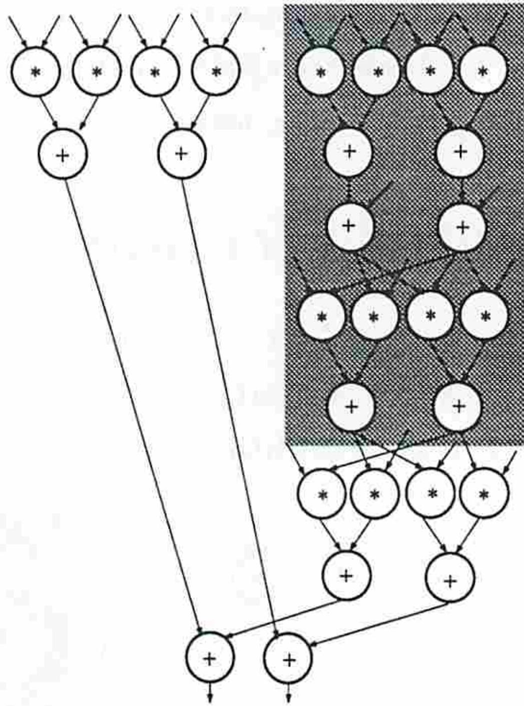


Figure 6.12: The partitioning without dependency loop

scheduling. Our data path predictions are also based on the same assumption. It is also assumed that the partitions are to be synthesized separately. The reasoning behind this is that today's behavioral synthesis packages already deal with several hard problems, and it becomes extremely difficult to synthesize multi-chip designs while considering system level issues. Therefore, some form of simplification in the process is needed. We preferred an approach in which the partitions are separately synthesized to reduce the complexity, but, the interaction between the partitions is still considered to produce well optimized designs.

When the partitions are to be synthesized separately, it becomes extremely hard to predict characteristics of any partitioning accurately in which there exists mutual data dependency (dependency loop) between any two partitions as shown in Figure 6.11. For the partitioning shown in Figure 6.11, it is harder to

accurately predict characteristics of each partition than to predict the characteristics of the unpartitioned behavior. In our scheduling model, the execution times of partitions are dependent on the the amount of parallelism available (resource allocation) and the internal data dependencies. For the partitioning shown in Figure 6.11, the area/delay characteristics of each partition are dependent on the area/delay characteristics of the other partition. In order to have accurate predictions, both partitions have to be processed simultaneously which is contrary to the partitioning assumptions made. On the other hand, the partitioning shown in Figure 6.12 is much more well behaved. Once the characteristics of the top partition are predicted/synthesized, that information can be fed to predict/synthesize the second partition. Therefore, we currently make an assumption that only partitions which do not have mutual data dependency on each other will be considered. This does not pose a serious problem for partitionability of the specifications since we allow multiple partitions on a single chip, heterogeneous partition sizes and cyclic dataflow among the chips (see Figure 6.3 for an example).

The scheduling model as well as pin sharing assumptions used in CHOP are quite simplistic. The data can be transferred in many pieces at different times (when the pins are available) instead of transferring them in a single uninterrupted transfer, resulting in more efficient and faster designs. When the behavioral specification is partitioned onto multiple chips, it is not acceptable to have off-chip steering logic to prevent possible off-chip bus clashes. In order to prevent this, more emphasis is need on how to allocate chip pins to data transfers. The CHOP model currently overestimates both the scheduling difficulty of pins and the bandwidth to be used in the data transfer. Therefore, CHOP's results may be acceptable for average designs although a more accurate pin allocation scheme combined with an improved scheduling model which considers unique input ready and output available times are needed.

Other types of analysis which are very useful during the partition evaluation step include the power consumption prediction for the implementation and a testability overhead prediction. In order to be able to check the feasibility of a real implementation accurately, the power rating of the design and as well as the testability features in the design need to be explicitly considered. There has not been any work yet to our knowledge to predict the power consumption of a design or the testability overhead for the design from behavioral-level specifications.

6.13 Summary

In this chapter, a constraint-driven behavioral partitioning methodology has been presented. The interactive partitioner, CHOP is coded in C and contains more than 12,000 lines of code (including BAD). CHOP provides fast feedback and allows a designer to search for several alternative implementation schemes. The feasibility of tentative multi-chip implementations is evaluated at the behavioral level using accurate predictions, thereby accurately addressing physical design constraints (area, throughput, delay and pin counts) at an early stage of the design process. A tool like CHOP is extremely beneficial in speeding up the system-level design process which also helps to increase the quality of designs.

The methodology presented in this paper is also suitable as a system-level advisor. The designer can easily check the effects of system-level decisions in real-time. The tool will also be extremely helpful in exploring partitioning methodologies, effects of high-level transformations on a multi-chip design, and effects of memory hierarchy on the performance, delay and area of a design.

Chapter 7

Partitioning Experiments and Results

In this chapter, several experiments performed with the partitioner will be presented. The first set of experiments was performed to validate the prediction techniques used in CHOP, and the results are given in Section 7.1. The next set of experiments includes several partitioning evaluations and explorations using the interactive partitioner, CHOP and the results are presented in Section 7.2. Based on the results of these evaluation experiments, the actual run-time complexity of CHOP and the variation of prediction results depending on the search method used in CHOP were also investigated in Section 7.3.

A simple automatic 2-way partitioning technique and results on the example designs are presented in Section 7.5 to show how the partitioning evaluation by CHOP can be used within a higher-level technique. Finally, several issues related to behavioral partitioning (interactive or automatic) are discussed in Section 7.6.

7.1 Validation of partitioning predictions

The validation of the partitioning predictions was performed in part by implementing the partitions and by comparing the prediction results to actual synthesis results.

One partitioning for each of the AR and FIR Filters was synthesized by the ADAM synthesis tools to validate the predictions used by CHOP. During the

synthesis of partitions CHOP's synthesis directions were used directly. We did not try to make major changes in the implementation of the partitions from the way that CHOP suggested. The processing units were synthesized using

- MAHA (scheduling),
- MABAL (operator, register and multiplexer allocation, and module binding),
- CSG (control specification generation),
- ESPRESSO and MKPLA (control specification optimization and PLA generation)

A small utility program was used to minimize the PLA outputs before each personality matrix from CSG was passed to ESPRESSO. The interface hardware (buffers, data transfer PLAs, and additional control logic) was manually designed. PEG, EQNTOTT, ESPRESSO, and MKPLA were used to generate the data transfer PLAs from manually-written finite state machine specifications for the interface hardware.

Partitioning predictions were performed as follows; The constraints for the initiation interval and the overall system delay were set to 25,000 and 60,000 ns respectively. The tentative data path clock cycle and data transfer clock cycle (to be later adjusted by CHOP) were set to 3,000 and 300 ns, respectively. The single cycle architecture style was chosen to be compatible with ADAM Synthesis tools. An 84-pin chip package was used for all chips in this experiment. The project area of the chip package used was 137,641 $mils^2$ ($371\,mils \times 371\,mils$). The pad area of the chips occupied 21,638 $mils^2$ resulting in actual usable chip area of 116,003 $mils^2$. The library used in the experiments is shown in Table 1.1. The library has a three micron technology which is consistent with technology assumed by some of the prediction techniques. The library allowed 9 library configurations (one operator type per operation type). CHOP's area estimates

were biased to be conservative. This was done so that the synthesized design would have a greater chance of satisfying the chip area constraints.

7.1.1 AR Filter

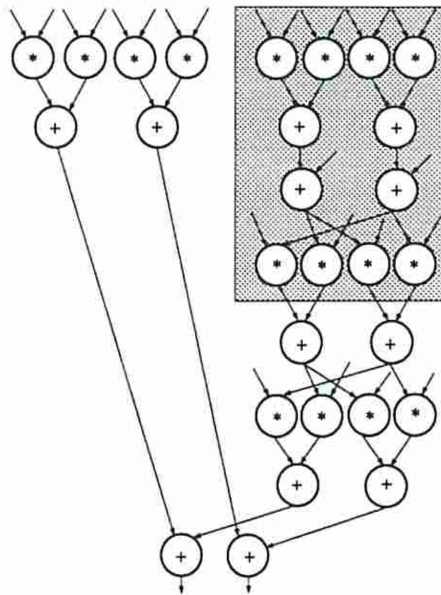


Figure 7.1: A 2-way partitioning of the AR Filter

The 2-way partitioning shown in Figure 7.1 has been selected as an example to validate the accuracy of the partitioning predictions. Each partition shown in Figure 7.1 was assigned to a separate chip. Although CHOP suggested to use *add3* as the adder type in partition 2, the larger and slower adder type *add2* was used in order to keep the clock cycle time of the schedule generated by MAHA close to the intended clock cycle time.

The final design characteristics followed the predictions closely as shown in Table 7.1. The total chip area (layout area) for the synthesized design is not available since it was not possible to generate layouts for the 3 micron technology assumed by some of the prediction techniques (PLA and wiring).

	Estimated	Synthesized
Initiation Interval (clock cycles)	20	20
System Delay (clock cycles)	45	45

Area† in mil^2	Chip No	Estimated	Synthesized
Processing Unit Active Area	1	58940	53770
Interface Active Area	1	14936	12344
Total Active Area	1	73876	66114
Total Area	1	130279	N/A
Processing Unit Active Area	2	55811	56314
Interface Active Area	2	23030	22897
Total Active Area	2	78841	79211
Total Area	2	137031	N/A

Table 7.1: Estimated and Synthesized Design Characteristics of the 2-way partitioning of the AR Filter shown in Figure 7.1

7.1.2 FIR Filter

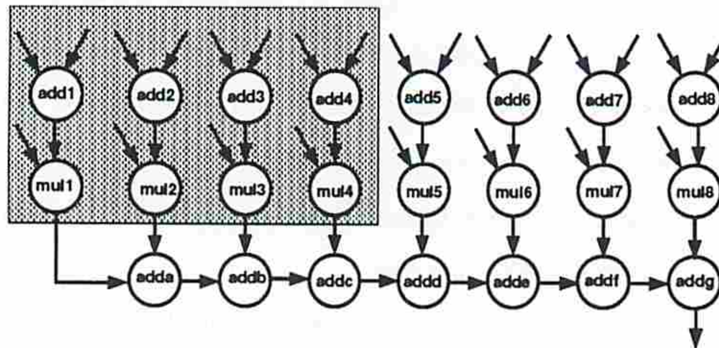


Figure 7.2: A 2-way partitioning of the FIR Filter

†Active area includes RTL and PLA area. The total area includes standard-cell routing area for processing units and the pad area for chips.

The 2-way partitioning of the FIR Filter shown in Figure 7.2 has also been selected as an example to validate the accuracy of the partitioning predictions. Each partition shown in Figure 7.2 was assigned to a separate chip. The MAHA schedule had to be modified manually since it was using one additional multiplier than was necessary. The prediction and synthesis results for this 2-way partitioning are shown in Figure 7.2.

	Estimated	Synthesized
Initiation Interval (clock cycles)	20	20
System Delay (clock cycles)	36	35

Area† in mil^2	Chip No	Estimated	Synthesized
Processing Unit Active Area	1	60604	52839
Interface Active Area	1	7000	6484
Total Active Area	1	67604	59323
Total Area	1	114202	N/A
Processing Unit Active Area	2	30279	33178
Interface Active Area	2	17079	14518
Total Active Area	2	47358	47696
Total Area	2	96964	N/A

Table 7.2: Estimated and Synthesized Design Characteristics of the 2-way partitioning of the FIR Filter shown in Figure 7.2

7.2 Partitioning evaluations

A set of experiments has been performed using CHOP interactively to investigate how much speed up can be obtained by partitioning behavioral specifications. The AR and FIR Filters were used as examples. In the experiments, both single-partition and multi-partition implementations were considered. CHOP's area estimates were biased to be conservative.

The constraints for the initiation interval and the overall system delay were both set to 60,000 *ns*. The constraints were intentionally set very loose, making

	Project Area (mil)				Single Pad Area
Package	Width	Length	Pad Delay	Pin Count	(mil^2)
A	90.55	133.86	25.0	28	297.60
B	181.10	267.72	25.0	40	297.60
C	271.65	267.72	25.0	40	297.60
D	271.65	267.72	25.0	64	297.60
E	271.65	267.72	25.0	84	297.60
F	311.02	362.20	25.0	64	297.60
G	311.02	362.20	25.0	84	297.60

Table 7.3: MOSIS standard chip packages

most implementations feasible. This was done to obtain predictions for the best possible implementation given the chip package constraints. The tentative data path clock cycle and data transfer clock cycle (to be later adjusted by CHOP) were set to 3,000 and 300 ns. The architecture style was selected to only allow single-cycle operations. The library used in the experiments is shown in Table 1.1. The library has a 3 micron technology which is consistent with technology assumed by the predictions. Unless otherwise noted, all experiments used the full library which allowed 9 library configurations [7.1].

Unless otherwise noted, in all 1, 2 or 3 chip implementations, the F type chip package is used from Table 7.3. The chip packaging information in Table 7.3 is taken from the MOSIS User Manual [MOS90]. Except for the partitions shown in Figures 7.5 and 7.8, every partition was assigned to a separate chip. The decision of how to partition the example specifications initially was performed mostly by intuition, followed by interactive search to reach the best possible partitioning for each case (2-chip or 3-chip partitioning).

The predicted design characteristics reported in Tables 7.4 and 7.5 are the best implementation predictions reported by CHOP for each partitioning [7.2].

[7.1] Each library configuration has one operator type per operation type.

[7.2] CHOP automatically eliminates results which are predicted to be inferior.

The iterative heuristic and the second exhaustive search technique always found the same results. The results of the first exhaustive search technique were occasionally inferior to what was found by the other techniques. More discussion on this can be found in Section 7.3.

As it can be seen from Tables 7.4 and 7.5 the effects of delays introduced into the clock cycle (roughly ranging from 40 to 120 ns) were estimated to be minimal due to the fact that the delays introduced into one data path clock cycle were actually distributed over 10 major clock periods. In designs with fast data path clock rates, impacts of such delays would be considerably higher. More discussion on this can be found in Section 7.6.6.

7.2.1 The AR filter

Four evaluations (including the unpartitioned design) are reported for the AR Filter. The partitions for these evaluations are shown in Figures 1.9, 7.3, 7.4, and 7.5. The statistical data on the evaluation results presented in Table 7.4 are reported in Table 7.6.

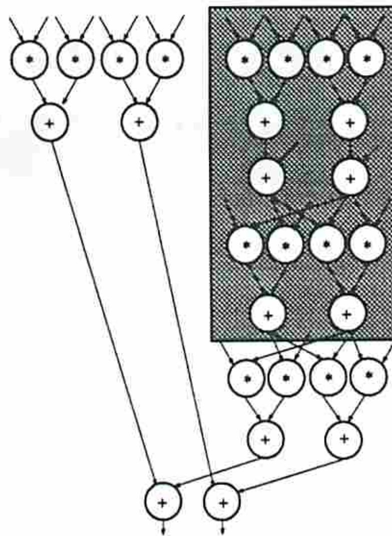


Figure 7.3: A 2-way partitioning of the AR Filter

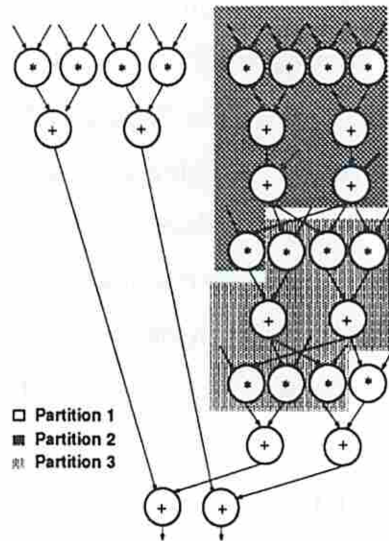


Figure 7.4: A 3-way partitioning of the AR Filter

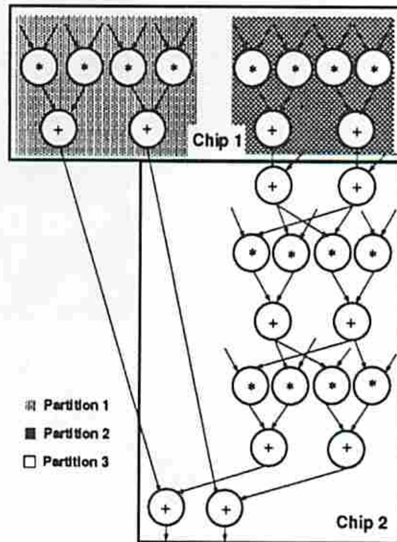


Figure 7.5: A 3-way, 2-chip partitioning of the AR Filter

P_{ID}	P	C	Initiation Interval (clock cycles)	System Delay (clock cycles)	Clock Cycle Time (ns)
1.9	1	1	168	168	309
7.3	2	2	40	67	309
7.4	3	3	20	67	308
7.5	3	2	30	58	309

P_{ID} : Partitioning identification (Figure) number
P : Number of partitions
C : Number of chips

Table 7.4: AR Filter Partitioning Evaluations

As it can be seen from the data reported in Table 7.4, the single partition implementation of the AR Filter had to be highly serial due to chip area constraints. By introducing another chip, the throughput can be increased 4 to 5 times with marked improvement for the system delay. Such a big difference in the throughput between the 1-chip and the 2-chip implementations is mainly due to 2 factors; the 2-chip implementation simply allowed more fine-grain parallelism within the partitions, but also allowed coarse-grain pipelining among the chips. Adding the third chip still improves the throughput with some small penalty on the system delay over the 2-chip implementation. Almost in all cases, the predicted chip area utilizations were above 85%.

7.2.2 The FIR filter

Four evaluations (including the unpartitioned design) are reported for the FIR Filter. The partitions for these evaluations are shown in Figures 1.8, 7.6, 7.7, and 7.8. The statistical data on the evaluation results presented in Table 7.5 are reported in Table 7.7.

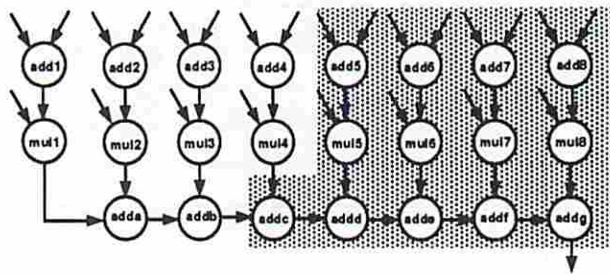


Figure 7.6: A 2-way partitioning of the FIR Filter

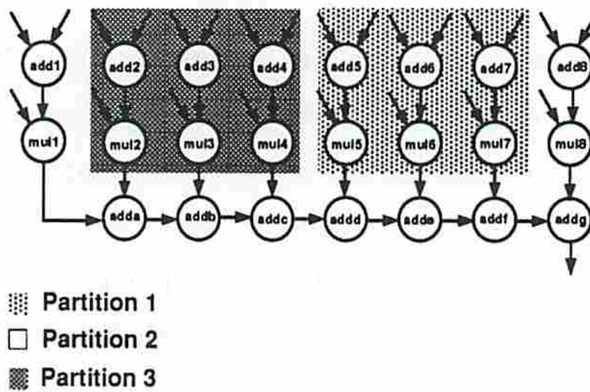


Figure 7.7: A 3-way partitioning of the FIR Filter

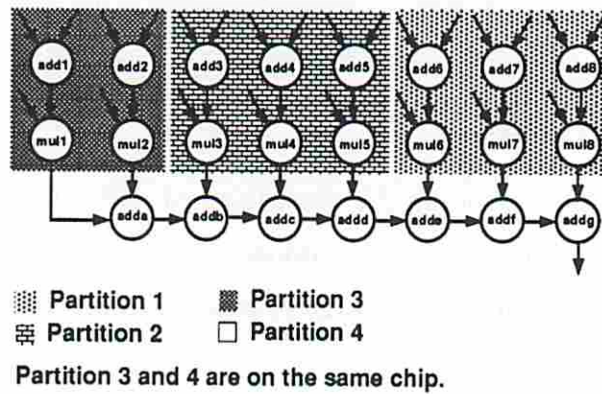


Figure 7.8: A 4-way, 3-chip partitioning of the FIR Filter

P_{ID}	P	C	Initiation Interval (clock cycles)	Delay (clock cycles)	Clock Cycle Time (ns)
1.8	1	1	48	48	313
7.6	2	2	20	46	309
7.7	3	3	20	39	311
7.7	3	3	10	46	305
7.8	4	3	10	27	305

Table 7.5: FIR Filter Partitioning Evaluations

As it can be seen from the data reported in Table 7.5, the throughput of the FIR Filter can be increased more than twice by partitioning the design onto two chips without any penalty on the system delay. Adding the third chip still doubles the throughput with respect to the 2-chip implementation again with no penalty on the system delay. The 4-partition, 3-chip implementation results in minimum initiation interval and delay since it allows maximum parallelism between the partitions. With the exception of the low performance 3-chip implementation (P_{ID} : 7.7), all predicted chip area utilizations were above 80%.

7.2.3 Multiple partitions per chip

It has been discussed in Chapter 6 that placing multiple carefully decided partitions on a single chip (without hardware sharing) is sometimes advantageous to improve the characteristics of designs. This happens mainly due to the fact that synthesizing everything on a chip as a single entity may cause some inefficiencies in the design. Smaller independent partitions may use their resources more effectively than large partitions. Another fact is that the wiring complexity increases as the design size increases. Figure 7.5 shows such a partitioning for the AR Filter. The evaluation results for this partitioning are shown in Table 7.4. This partitioning is expected to achieve an initiation interval and system delay of 30 and 58 clock cycles, respectively. But, if the two partitions which are on

the same chip are merged into a single partition, then the new estimation results for the best achievable global design have the initiation interval of 40 and the system delay of 69 clock cycles. Similarly for the FIR Filter, if the two partitions in Figure 7.8 which are on the same chip are merged into a single partition, then the new estimation results for the best achievable global design have the same results as reported for the partitioning shown in Figure 7.7.

7.3 Comparison of search methods

As it was discussed in Chapter 6, three techniques are currently implemented in CHOP for the global implementation search. Since these techniques perform the global implementation search very differently, the results as well as the corresponding run-times can be quite different. The statistical information and run times on the experimental results presented in Tables 7.4 and 7.5 are given in Tables 7.6 and 7.7. All CPU times reported are on a Sun Sparc 4/460 and in seconds. CPU times quote the actual computation time and do not include I/O and pre-processing times.

In our experience so far, the iterative heuristic found the same results as the second exhaustive search technique with orders of magnitude speed advantage. The first exhaustive search technique failed to find as good estimates as the other techniques for some of the evaluations reported. Three such cases are shown in Figures 1.9, 7.3, 1.8. As an example, for the partitioning shown in Figure 7.3, both the iterative heuristic and the second exhaustive search techniques selected the same global implementation with a three stage implementation for each partition, using the global initiation interval of four data path clock cycles (40 cycles in terms of the major clock cycle). On the other hand, the first exhaustive search technique had to select the global initiation interval of three data path clock cycles for the same partition implementations selected in the search process, making that implementation infeasible due to additional data transfer module area. The first exhaustive search technique then reported a

global implementation using different partition implementations (a three stage implementation for one partition and a four stage implementation for the other partition) and the global initiation interval of four data path clock cycles as the best global implementation, resulting in a longer system delay.

		# of global implementation trials		
P_{ID}	N_p	ES 1	ES 2	Heuristic
1.9	207	39	378	83
7.3	207	1681	22965	38
7.4	228	42315	622153	19
7.5	206	18522	273321	18

N_p : Total number of predictions produced by BAD
ES 1 : Exhaustive Search 1
ES 2 : Exhaustive Search 2

	CPU Time		
P_{ID}	ES 1	ES 1	Heuristic
1.9	0.32	1.06	0.44
7.3	4.13	34.02	0.41
7.4	68.19	789.85	0.36
7.5	25.84	236.44	0.34

Table 7.6: Statistical data on AR filter evaluation results in Table 7.4

7.3.1 Constraining the search performed by CHOP

CHOP performs a brute-force search in the global design space. The number of global implementations searched is highly dependent on the number of individual partition implementations as well as the performance constraint. Even though the heuristic presented in Chapter 6 performs favorably, special care may be taken to constrain the design search process, thereby decreasing the run-time.

		# of global implementation trials		
P_{ID}	N_p	ES 1	ES 2	Heuristic
1.8	201	75	820	35
7.6	198	2016	28980	19
7.7	203	49348	700408	20
7.8	156	454272	7239576	20

	CPU Time		
P_{ID}	ES 1	ES 2	Heuristic
1.8	0.37	1.58	0.35
7.6	3.96	35.64	0.27
7.7	71.77	722.68	0.36
7.8	361.99	3644.47	0.26

Table 7.7: Statistical data on FIR filter evaluation results in Table 7.5

The number of partition implementations is proportional to the number of possible library configurations. The predictions in BAD are also controlled by the performance constraint in order not to generate predicted designs implementations which would be infeasible. The looser the performance constraint is, the higher number of predicted designs. Finally, given selected implementations of partitions, a looser performance constraint translates into a higher number of global implementations which can be constructed from the set of already selected partition implementations.

If the information given to CHOP allows a very high number of global implementations, CHOP may have to cover an enormous search space due to the brute-force nature of its search methods. If special care is taken to make sure that the performance constraint is as tight as possible, and the library does not contain operators which would be too slow or to large for designs being partitioned, dramatic decreases in run-times can be obtained.

The partitioning shown in Figure 7.4 was used as an example. Rather than using the loose performance constraint given in Section 7.2, the performance constraint was tightened to $7,000\ ns$. The multipliers, mul1 and mul3 were removed from the library by intuition as being too large or too slow. We also asked BAD not to predict any pipelined data path implementations since we believed that the pipelined implementations of the AR Filter would be too large (the AR Filter has a large number of primary inputs to be stored).

After making the changes in the specifications, we ran CHOP again and CHOP's predictions were unchanged. But, there were dramatic decreases in the amount of search performed (and of course, the run-time) for each search method used in CHOP as it can be seen by comparing the statistical data presented in Tables 7.6 and 7.8.

		# of global implementation trials		
P_{ID}	N_p	ES 1	ES 2	Heuristic
7.4	6	6	12	2

	CPU Time		
P_{ID}	ES 1	ES 1	Heuristic
7.4	0.05	0.07	0.03

Table 7.8: Statistical data on a constrained run using the AR Filter

7.4 Non-uniform partitioning in CHOP

Since the target chip characteristics are related to the manufacturing cost of the design, if the performance and delay constraints allow, it is preferable to search for lower-cost implementations which may happen to be achievable only by a non-uniform partitioning. An example is given in Table 7.9. According to 1990 MOSIS price list, the prices of chips for packages F and C are \$467

and \$305, respectively. An extensive search was performed for 2-partition 2-chip implementations of the AR Filter onto different target configurations of the chip sets (composed of packages, type F and/or type C) as shown in Table 7.9. If the second design in Table 7.9 is still acceptable for its performance and delay characteristics, then it is definitely cheaper to use one large and one small chip rather than 2 large chips for the 2-chip implementation.

Design	Package Type		Price	Initiation Interval (clock cycle)	Delay (clock cycle)
	Chip 1	Chip 2			
1	F	F	\$934	40	67
2	F	C	\$774	50	99
3	C	C	\$610	90	172

Table 7.9: Different cost 2-chip implementations of the AR Filter using MOSIS chip packages

7.5 A simple automatic 2-way partitioning

The interactive partitioner provides excellent means of evaluation for automatic partitioning methods. A simple automatic 2-way partitioning technique for horizontal cuts have been implemented on top of the interactive partitioner to show how easily and effectively higher-level automatic partitioning techniques can be implemented using CHOP. The actual additional coding took only one day. The outline for the technique is shown in Figure 7.9. The technique automatically partitions a behavioral specification onto two completely specified chips. The technique first orders the operations. It starts the search with all operations assigned to one of the chips. Then, it moves the operations one by one to the other chip according to the ordering and evaluates each partitioning using CHOP. The technique tries two different orders (ASAP and ALAP) for moving operations and also tries different orders for the chip assignments, if the chip characteristics are different.

Procedure 2-way partitioning
Read in the chip characteristics.
Assign all operations to chip 1.
Move operations in ASAP order to chip 2 and evaluate each partitioning.
Assign all operations to chip 1.
Move operations in ALAP order to chip 2 and evaluate each partitioning.
If characteristics of chip 1 and 2 are different
  Assign all operations to chip 2.
  Move operations in ASAP order to chip 1 and evaluate each partitioning.
  Assign all operations to chip 2.
  Move operations in ALAP order to chip 1 and evaluate each partitioning.
Report the best partitioning.
End.

Figure 7.9: A simple automatic 2-way partitioning technique.

This simple automatic 2-way partitioning technique was used on the AR Filter and the FIR Filter. The experiment set up was the same as the one assumed in Section 7.2. For each design, it found the same partitioning which was previously found by interactive partitioning to have the best predicted global characteristics. These are shown in Figures 7.3 and 7.6 and their evaluation results are reported in Tables 7.4 and 7.5.

The automatic 2-way partitioning of the AR Filter and FIR Filter took 54 and 44 partition evaluations, respectively. The total run times of each case (using the heuristic) were and 31.43 and 16.33 seconds, respectively. An experiment like this (using the partitioning evaluation several times) shows the importance how useful it is to tightly constrain the search in CHOP. When the performance constraint was tightened to 15,000 ns, and BAD was asked not to generate any predictions for pipelined data paths (the library was kept unchanged), the total run time for each case dropped to 3.93 and 3.5 seconds, respectively.

7.6 Discussion of Behavioral Partitioning Issues

In order to produce *good quality* behavioral partitions, there are several aspects of behavioral partitioning which need to be analyzed during the partitioning process. Some of these design issues, if considered separately, have conflicting goals. The experiments showed that an automatic behavioral partitioning technique should be able to address these issues as much as possible. Each of these topics mentioned in the following sections are difficult research problems.

Behavioral partitioning onto multiple chips can be viewed as the combination of a task creation and a task allocation in a multi-processor synthesis environment. Partitioning the behavior into groups of operations corresponds to the task creation. The assignment of these partitions onto chips, ordering them in time, and allocating global resources (chip area, pins, memory) to the partitions corresponds to the task allocation. Finally, synthesizing a data path for each partition corresponds to processor synthesis.

7.6.1 Parallelism

Although it is possible to achieve higher performance by increasing the fine-level parallelism between the operations in partitions (this can be achieved up to a limit by having more available chip area), it is essential to have enough parallelism between the partitions (tasks) to increase the performance and to reduce the system delay. A partitioning with as much parallelism between the partitions as possible is more likely to have good delay characteristics.

7.6.2 Partition Granularity

When multiple partitions per chip are allowed in a multi-chip implementation, the partition size becomes a tradeoff issue by itself. At one extreme, having

very few and very large partitions may over-constrain the synthesis process by not allowing some tradeoffs otherwise possible. At the other extreme, having many small partitions allows a great flexibility in synthesis. But, in this case, predicting the final design characteristics becomes a problem. If the prediction techniques are simplistic, then the prediction results tend to be inaccurate. If the prediction techniques are highly detailed, then the prediction process almost becomes equivalent to the actual synthesis process which suffers from run-time complexity. Therefore, finding the right partition sizes is a difficult problem.

7.6.3 Non-linearity of moves in the design space

Since behavioral synthesis is highly flexible and since there are many interacting and counteracting discrete optimizations and constraints, predicting the effects of a design decision on the implementation is very difficult. A minor perturbation to the design specification (moving an operation to another partition, slightly modifying the clock cycle time, slightly modifying a constraint) can cause significant changes in the synthesized design. The overall design characteristics after the perturbation may still be very similar to the old design characteristics, but the structure of the implementation may change completely, as we have occasional observed. For example, having a different length schedule with a different clock cycle time, less number of operators and more number of registers may still result in similar design characteristics. This non-linearity prevents the prediction of how a small perturbation would affect the design characteristics, making it very difficult to design automatic behavioral partitioning techniques which are robust.

7.6.4 Utilization

Since the chip areas are the major resources in a design, a good partitioning should make a good use of the area of each chip in the design. Tradeoffs for fine and coarse grain parallelism can be used to achieve this objective.

Another type of utilization is the utilization of operators used in each partition. Even though the synthesis techniques can perform optimizations later, the way the partitions are specified also has an effect on the operator utilization in the implementation. Although, it is theoretically possible that a small perturbation to a partitioning may cause a significant change in the synthesized design, in several cases, some major design decisions virtually remain unchanged for small perturbations to partitions. For example, if a particular 2-stage implementation for a partition is predicted to be far superior to other ways of implementing the partition, and if there are 3 multiplication operations in the partition, it is quite likely that one of the multipliers will not be utilized well in the synthesized design. In such a case, it is possible to move a multiplication operation from another partition to this partition without making any significant changes (e.g some additional multiplexing) in the partition's characteristics, but possibly improving the characteristics of the partition from which the multiplication operation is moved. Therefore, among other issues, trying to reach high utilizations of the chip area and operators reduces unused resources, thereby increasing the chances of synthesizing a better design.

7.6.5 Off-chip Communication

In a multi-chip design, the values being transferred off chip can be viewed as communication requests, and chip pins as the communication channel. Of course, the minimization of off-chip value transfers should be among the goals of the partitioning. But, at the same time, indirect consequences of over-minimizing off-chip communication have to be taken into account since communication channels are sharable resources. It is possible to have a non-minimal number of off-chip value transfers, but, it may still be possible to serialize these value transfers in time and overlap them with the execution of partitions, resulting in highly optimized designs. The right parallelism between the partitions and the right partition granularity play an important role in allowing several tradeoffs in sharing the

communication channels and minimizing the buffer and the control overheads, resulting in globally better implementations.

7.6.6 Clock Cycle Times

Another tradeoff to produce different implementations is to vary the clock cycle times and the architecture type (single or multi cycle operations). Using a faster clock by allowing multi-cycle operations definitely decreases the discretization caused by the scheduling compared to using a slower clock and single-cycle operations. But, the gains are not necessarily automatic since there are limits on increasing the fine-grain parallelism. For example, using the tentative data path clock cycle time of $3,000\,ns$ and only allowing single-cycle operations, the partitioning shown in Figure 7.3 has a predicted best implementation with initiation interval and system delay of $12,360$ and $20,703$ ns, respectively. When the tentative data path clock cycle time is reduced and multi-cycle operations are allowed, the initiation interval and system delay in terms of the clock cycle are reduced. But, the actual initiation interval and system delay did not change considerably since the major clock cycle time became longer, as shown in Table 7.10. The delays introduced into the data path clock (delays due to registers, multiplexers, control, wiring, off-chip communications) mostly depend on the parallelism which does not change drastically by changing the clocking scheme. As a result, these roughly same delays cause a higher overhead per clock cycle for faster data path clocks.

k_{dp}	II (clock cycle)	System Delay (clock cycle)	c' (ns)	II (ns)	System Delay (ns)
10	40	67	309	12360	20703
3	33	64	325	10725	20800
1	28	56	374	10472	20944

k_{dp} : The data path clock in terms of the major clock cycle time
c' : Adjusted major clock cycle time
II : Initiation Interval

Table 7.10: Evaluations for varying clock cycle times for partitioning shown in 7.3

Chapter 8

Conclusion and Future Research

8.1 Contributions

As it was mentioned in Chapter 1, most design automation systems consist of a collection of tools which are focused on specific optimizations and do not have a global understanding of the problem, due to the high complexity of synthesis. As a result, excessive design iterations may be required to produce good quality designs.

In this thesis a system-level synthesis methodology using behavioral predictions and behavioral partitioning to reduce the design time was proposed. In the proposed system-level synthesis methodology a very large discrete design space is quickly searched using predictions techniques and the large discrete design space is reduced to a relatively small subspace for synthesis. Then, the necessary synthesis directions are given to the synthesis tools or designer to perform extensive optimizations within the subspace to find the best possible design.

Chapter 3 describes a data path synthesis tool, MABAL which was written to create a pathway to produce RTL designs from behavioral specifications. The fact that MABAL is comprehensive, very flexible, and fast allowed us to perform various tradeoff studies, to reach a better understanding of the synthesis process, which were extremely helpful in developing the prediction techniques.

Chapter 4 describes the first comprehensive and integrated behavioral predictor for high-level synthesis. BAD uses some previously proposed prediction techniques (some with minor modifications) and many new prediction techniques to predict the area and the delay characteristics of designs. BAD's prediction techniques span from the design style selection subtask to the layout.

Chapter 6 describes the first system-level behavioral partitioning approach which takes into account actual physical design characteristics and system-level design issues. Using the prototype software, CHOP, several important issues for behavioral partitioning were identified.

8.2 Future Research

The research performed in each major topic in this thesis (data path synthesis, behavioral predictions, and behavioral partitioning) uncovered several future directions.

The current behavioral synthesis process is quite complicated, as described in the thesis. But, in order to able to produce good quality designs with constantly changing technology, the synthesis systems need to use techniques which are not technology-dependent. But, the techniques should be technology sensitive so that the optimizations can still produce good quality results for different applications. Synthesis techniques should be flexible. Having a rigid design style may cause the software to miss some good designs. The lower-level synthesis process (logic synthesis, placement, and routing) needs to be parameterized into higher levels since the lower-level synthesis tasks may significantly affect the results of optimizations performed during high-level synthesis. The synthesis systems should somehow take layout features and wire delays into account since the wire delays are becoming comparable to functional delays. Since designs are becoming larger and larger in size, partitioning needs to be taken into account during synthesis. New synthesis techniques which take off-chip delays into account, and which can treat chip pins as resources are needed.

Newly proposed synthesis techniques try to produce globally better designs by simultaneously considering several synthesis subtasks together and utilizing more tradeoffs. With this increasing complexity of synthesis, it is essential to be able to reduce the design search space without missing good designs. Therefore, we believe that the prediction techniques proposed in this thesis should be improved to be more robust and accurate. The prediction techniques should also keep up with the new improvements in the synthesis technology. To be specific, the prediction techniques should be upgraded so that unique input ready and output available times can be taken into account. A better handling of dataflow dependencies for operator allocation and for the estimation of number of stages of a pipelined design is definitely needed. An interesting task is to include predictions for software (e.g a microprocessor) implementations in the predictions. Then, it would be possible to perform hardware/software tradeoffs as a part of the system-level synthesis process. Of course, including prediction techniques for power characteristics and testability overhead is also essential to be able to address realistic problems.

Upgrading the prediction techniques to consider individual input arrival and output available times would allow a more accurate evaluation mechanism for arbitrary partitions, in contrary to the current topological restrictions on the partitioning. Modification of partitioning predictions to model and to consider the structure of off-chip interconnect/buses is also needed to make CHOP a more practical tool. Although considerable work has been done for CHOP to measure the quality of a given partitioning, fully-automatic behavioral partitioning is still an open problem. More detailed analysis of what constitutes a first order effect in behavioral partitioning is definitely needed to move further in automatic behavioral partitioning.

Finally, in the proposed system-level synthesis process, the integration of high-level transformations, predictions, hardware/software tradeoffs, memory hierarchy design, and synthesis together with the inclusion of design costs into synthesis are among the future research topics which would enable the system-level synthesis process to address realistic design issues.

Reference List

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

[BG90] F. Brewer and D. D. Gajski. A System for Constraint Driven Behavioral Synthesis. *IEEE Trans. on Computer-Aided Design*, CAD-9(7):681–695, July 1990.

[BK87] G. Borriello and R. H. Katz. Synthesis and Optimization of Interface Transducer Logic. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 274–277. IEEE, November 1987.

[BKM+89] M. Beardslee, C. Kring, R. Murgai, H. Savoj, R. K. Brayton, and A. R. Newton. SLIP: A Software Environment for System Level Interactive Partitioning. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 280–283. IEEE, November 1989.

[BMB+88] M. Balakrishnan, A. K. Majumdar, D. K. Banerji, J. G. Linders, and J. C. Majithia. Allocation of Multiport Memories in Data Path Synthesis. *IEEE Trans. on Computers*, CAD-7(4):536–540, April 1988.

[CB87] R. Camposano and R. K. Brayton. Partitioning before Logic Synthesis. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 324–326. IEEE, November 1987.

[CB88] X. Chen and M. L. Bushnell. A Module Area Estimator for VLSI Layout. In *Proc. 25th Design Automation Conf.*, pages 54–59. ACM/IEEE, June 1988.

[Che90] C. Chen. VHDL2DDS: A VHDL Language to DDS Data Structure Translator. Department of Electrical Engineering, University of Southern California, 1990. Internal Report.

[Cos89] J. La Coss. USC-Information Sciences Institute, September 1989. Personal Communication.

[CS90] C. H. Chen and G. E. Sobelman. Singleport/Multiport Memory Synthesis in Data Path Design. In *Proc. Int'l Symp. on Circuits and Systems*, pages 1110–1113. IEEE, May 1990.

[CT90] R. J. Cloutier and D. E. Thomas. The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm. In *Proc. 27th Design Automation Conf.*, pages 71–76. ACM/IEEE, June 1990.

[CvE87] R. Camposano and J.T.J. van Eijndhoven. Partitioning a Design in Structural Synthesis. In *Proc. Int'l Conf. on Computer Design*, pages 564–566. IEEE, October 1987.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proc. 19th Design Automation Conf.*, pages 175–181. ACM/IEEE, June 1982.

[Fre82] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Ellis-Horwood Limited, West Sussex, England, 1982.

[GBK85] E. F. 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 Trans. on Computer-Aided Design*, CAD-4(2):134–142, April 1985.

[GE91] C. H. Gebotys and M. I. Elmasry. Optimal Synthesis of High Performance Architectures. In *Proc. Custom Integrated Circuits Conf.*, pages 11.6.1–11.6.4. IEEE, May 1991.

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

[Gir91] E. F. Girczyc, May 1991. Personal Communication.

[GK84] E. F. Girczyc and J. P. Knight. An ADA to Standard Cell Hardware Compiler Based on Graph Grammars and Scheduling. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 726–731. IEEE, November 1984.

[GM90] R. Gupta and G. De Micheli. Partitioning of Functional Models of Synchronous Digital Systems. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 216–219. IEEE, November 1990.

[GRVM90] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man. An Efficient Microcode Compiler for Application Specific DSP Processors. *IEEE Trans. on Computer-Aided Design*, CAD-9(9):925–937, September 1990.

[Gup90] P. Gupta. PLA and Wire Delay Analysis. Department of Electrical Engineering, University of Southern California, 1990. Internal Report.

[Hag91] J. W. Hagerman. A Fast and Accurate Technique for Function Unit Allocation Estimation. Technical Report CMUCAD-91-28, Carnegie Mellon University, April 1991.

[HC89] R. Hartley and A. Casavant. Tree-height Minimization in Pipelined Architectures. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 112–115. IEEE, November 1989.

[HCCd89] K. S. Hwang, A. E. Casavant, C. Chang, and M. A. d'Abreu. Scheduling and Hardware Sharing in Pipelined Data Paths. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 24–27. IEEE, November 1989.

[HCDd88] K. S. Hwang, A. E. Casavant, M. Dragomirecky, and M. A. d'Abreu. Constrained Conditional Resource Sharing in Pipeline Synthesis. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 52–55. IEEE, November 1988.

[HCLH90] C. Huang, Y. Chen, Y. Lin, and Y. Hsu. Data Path Allocation Based on Bipartite Weighted Matching. In *Proc. 27th Design Automation Conf.*, pages 499–504. ACM/IEEE, June 1990.

[Hil85] P. Hilfinger. A High-level Language and Silicon Compiler Digital Signal Processing. In *Proc. Int'l Symp. on Circuits and Systems*, pages 213–216. IEEE, May 1985.

[Hit83] C. Y. Hitchcock. Automated Synthesis of Data Paths. Master's thesis, Department of Electrical Engineering, Carnegie-Mellon University, January 1983.

[HLH91] C. Hwang, J. Lee, and Y. Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Trans. on Computer-Aided Design*, 10(4):464–475, April 1991.

[HP78] L. J. Hafer and A. C. Parker. Register-Transfer Level Digital Design Automation: The allocation process. In *Proc. 15th Design Automation Conf.*, pages 213–219. ACM/IEEE, June 1978.

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

[HP89] S. Hayati and A. C. Parker. Automatic Production of Controller Specifications from Control and Timing Behavioral Descriptions. In *Proc. 26th Design Automation Conf.*, pages 75–80. ACM/IEEE, June 1989.

[HS71] A. Hashimoto and J. Stevens. Wire Routing By Optimizing Channel Assignment Within Large Apertures. In *Proc. 8th Design Automation Workshop*, pages 155–169. ACM/IEEE, June 1971.

[Jai89] R. Jain. *High-Level Area-Delay Prediction with Application to Behavioral Synthesis.* PhD thesis, Department of Electrical Engineering, University of Southern California, 1989.

[Jai90] R. Jain. MOSP: Module Selection for Pipelined Designs with Multi-Cycle Operations. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 212–215. IEEE, November 1990.

[JKMP89] R. Jain, K. Küçükçakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proc. 26th Design Automation Conf.*, pages 55–61. ACM/IEEE, June 1989.

[JMP88] R. Jain, M. J. Mlinar, and A. C. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 48–51. IEEE, November 1988.

[JPP87] R. Jain, A. C. Parker, and N. Park. Predicting Area-Time Tradeoffs for Pipelined Designs. In *Proc. 24th Design Automation Conf.*, pages 35–41. ACM/IEEE, June 1987.

[JPP88] R. Jain, A. C. Parker, and N. Park. Module Selection for Pipelined Designs. In *Proc. 25th Design Automation Conf.*, pages 542–547. ACM/IEEE, June 1988.

[KL70] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(1):291–307, January 1970.

[Kna89] D. W. Knapp. Synthesis from Partial Structure. In *Design Methodologies for VLSI and Computer Architecture. D. A. Edwards, (Ed.)*, pages 35–51. North-Holland, 1989.

[Kod72] U. R. Kodres. Partitioning and Card Selection. In *Digital System Design Automation, vol.I: Theory and Techniques. Melvin A. Breuer (ed.)*, pages 173–212. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[KP85] D. W. Knapp and A. C. Parker. A Unified Representation for Design Information. In *Proc. 7th Int'l Conf. on CHDL*, pages 337–353. The Netherlands, North-Holland, August 1985.

[KP86] F. J. Kurdahi and A. C. Parker. PLEST: A Program for Area Estimation of VLSI Integrated Circuits. In *Proc. 23rd Design Automation Conf.*, pages 467–473. ACM/IEEE, June 1986.

[KP87] F. J. Kurdahi and A. C. Parker. REAL: A Program for REgister ALlocation. In *Proc. 24th Design Automation Conf.*, pages 210–215. ACM/IEEE, June 1987.

[KP89a] K. Küçükçakar and A. C. Parker. MABAL - A Software Package for Module and Bus ALlocation. Technical Report CRI-88-61, Department of Electrical Engineering, University of Southern California, March 1989.

[KP89b] F. J. Kurdahi and A. C. Parker. Techniques for Area Estimation of VLSI Layouts. *IEEE Trans. on Computer-Aided Design*, 8(1):81–92, January 1989.

[KP90a] K. Küçükçakar and A. C. Parker. BAD: Behavioral Area-Delay Predictor. Technical Report CENG-90-31, Department of Electrical Engineering, University of Southern California, November 1990.

[KP90b] K. Küçükçakar and A. C. Parker. CHOP: A Constraint-Driven System-Level Partitioner. Technical Report CENG-90-26, Department of Electrical Engineering, University of Southern California, November 1990.

[KP90c] K. Küçükçakar and A. C. Parker. Constraint-Driven System-Level Partitioning of Digital System Specifications. IFIP-GI/ITG Workshop on Partitioning, January 1990.

[KP90d] K. Küçükçakar and A. C. Parker. Data Path Tradeoffs using MABAL. In *Proc. 27th Design Automation Conf.* ACM/IEEE, June 1990.

[KP90e] K. Küçükçakar and A. C. Parker. MABAL: A Software Package for Module and Bus ALlocation. *Int'l Journal of Computer Aided VLSI Design*, 2(4):419–436, 1990.

[KP91] K. Küçükçakar and A. C. Parker. CHOP: A Constraint-Driven System-Level Partitioner. In *Proc. 28th Design Automation Conf.*, pages 514–519. IEEE/ACM, June 1991.

[KR91] F. J. Kurdahi and C. Ramachandran. LAST: Layout Area and Shape Function EsTimator. In *Proc. 1st European Design Automation Conf.*, pages 351–355. ACM/IEEE, February 1991.

[KT83] T. J. Kowalski and D. E. Thomas. The VLSI Design Automation Assistant: Prototype System. In *Proc. 20th Design Automation Conf.*, pages 479–483. ACM/IEEE, June 1983.

[Küç90] K. Küçükçakar. Constraint-Driven System-Level Partitioning of Behavioral Specifications. Ph. D. Proposal, Department of Electrical Engineering, University of Southern California, January 1990.

[Kun84] S. Y. Kung. On Supercomputing with Systolic/Wavefront Array Processors. *Proc. IEEE*, 72(7):867–884, July 1984.

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

[KWK85] S. Y. Kung, H. J. Whitehouse, and T. Kaliath. *VLSI and Modern Signal Processing.* Prentice-Hall, 1985.

[LK66] R. I. Levin and C. A. Kirkpatrick. *Planning and Control with PERT/CPM.* McGraw Hill, 1966.

[LT89] E. D. Lagnese and D. E. Thomas. Architectural Partitioning for System Level Design. In *Proc. 26th Design Automation Conf.*, pages 62–67. ACM/IEEE, June 1989.

[LT91] E. D. Lagnese and D. E. Thomas. Architectural Partitioning for System Level Synthesis of Integrated Circuits. *IEEE Trans. on Computer-Aided Design*, CAD-10(7):847–859, July 1991.

[LvMvdW+91] P. E. R. Lippens, J. L. van Meerbergen, A. van der Werf, W. F. J. Verhaegh, and B. T. McSweeney. Memory Synthesis for High Speed DSP Applications. In *Proc. Custom Integrated Circuits Conf.*, pages 11.7.1–11.7.4. IEEE, May 1991.

[McF78] M. C. McFarland. The VT: A Database for Automated Digital Design. Technical Report DRC-01-4-80, Department of Electrical Engineering, Carnegie-Mellon University, December 1978.

[McF83] M. C. McFarland. Computer-Aided Partitioning of Behavioral Hardware Descriptions. In *Proc. 20th Design Automation Conf.*, pages 472–478. ACM/IEEE, June 1983.

[McF86] M. C. McFarland. Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *Proc. 23rd Design Automation Conf.*, pages 474–480. ACM/IEEE, June 1986.

[McF91] M. C. McFarland, January 1991. Personal Communication.

[MK86] M. C. McFarland and T. J. Kowalski. Assisting DAA: The Use of Global Analysis in an Expert System. In *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pages 482–485. IEEE, October 1986.

[MK88] G. De Micheli and D. C. Ku. HERCULES - A System for High-Level Synthesis. In *Proc. 25th Design Automation Conf.*, pages 483–488. ACM/IEEE, June 1988.

[Mli91] M. J. Mlinar. *System Level Tradeoffs in VLSI Design.* PhD thesis, Department of Electrical Engineering, University of Southern California, May 1991.

[MOS90] MOSIS User Manual. USC-Information Sciences Institute, 1990.

[MPC88] M. C. McFarland, A. C. Parker, and R. Camposano. Tutorial on High-Level Synthesis. In *Proc. 25th Design Automation Conf.*, pages 330–336. ACM/IEEE, June 1988.

[MPC90] M. C. McFarland, A. C. Parker, and R. Camposano. The High-Level Synthesis of Digital Systems. *Proc. IEEE*, 78(2):301–318, February 1990.

[NT86] J. A. Nestor and D. E. Thomas. Behavioral Synthesis with Interfaces. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 112–115. IEEE, November 1986.

[Pan88] B. M. Pangrle. Splicer: A Heuristic Approach to Connectivity Binding. In *Proc. 25th Design Automation Conf.*, pages 536–541. ACM/IEEE, June 1988.

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

[Pen86] Z. Peng. Synthesis of VLSI Systems with the CAMAD Design Aid. In *Proc. 23rd Design Automation Conf.*, pages 278–284. ACM/IEEE, June 1986.

[PGH91] A. C. Parker, P. Gupta, and A. Hussain. The effects of Physical Design Characteristics on the Area-Performance Tradeoff Curve. In *Proc. 28th Design Automation Conf.*, pages 530–534. ACM/IEEE, June 1991.

[PK89a] N. Park and F. J. Kurdahi. Module Assignment and Interconnect Sharing in Register Transfer Synthesis of Pipelined Data Paths. In *Int'l Conf. on Computer Aided Design*, pages 16–19. IEEE, November 1989.

[PK89b] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Trans. on Computer-Aided Design*, 8(6):661–679, June 1989.

[PK90] C. A. Papachristou and H. Konuk. A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm. In *Proc. 27th Design Automation Conf.*, pages 77–83. ACM/IEEE, June 1990.

[PKG86] P. G. Paulin, J. P. Knight, and E. F. Girczyc. HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis. In *Proc. 23rd Design Automation Conf.*, pages 263–270. ACM/IEEE, June 1986.

[PKPW91] A. C. Parker, K. Küçükçakar, S. Prakash, and J. Weng. Unified System Construction (USC). In *High-level VLSI Synthesis. Edited by R. Camposano and W. Wolf*, pages 331–354. Kluwer Academic Publishers, 1991.

[PP88] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Trans. 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 *Proc. 23rd Design Automation Conf.*, pages 461–466. ACM/IEEE, June 1986.

[PR89] M. Potkonjak and J. Rabaey. A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs. In *Proc. 26th Design Automation Conf.*, pages 7–12. ACM/IEEE, June 1989.

[Pv82] T. S. Payne and W. M. vanCleemput. Automated Partitioning of Hierarchically Specified Digital Systems. In *Proc. 19th Design Automation Conf.*, pages 182–192. ACM/IEEE, June 1982.

[RCHP91] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast Prototyping of Datapath-Intensive Architectures. *IEEE Design and Test of Computers*, pages 40–51, June 1991.

[Res86] M. L. Resnick. SPARTA : A System Partitioning Aid. *IEEE Trans. on Computer-Aided Design*, CAD-5(4):490–498, October 1986.

[Rob85] E. A. Robinson. *Probability Theory and Applications.* Int'l Human Resources Development Corporation, 1985.

[RP90] J. Rabaey and M. Potkonjak. Resource Driven Synthesis in the HYPER System. In *Proc. Int'l Symp. on Circuits and Systems*, pages 2592–2595. IEEE, May 1990.

[RT85] J. V. Rajan and D. E. Thomas. Synthesis by Delayed Binding of Decisions. In *Proc. 22nd Design Automation Conf.*, pages 367–373. ACM/IEEE, June 1985.

[Sak83] T. Sakurai. Approximation of Wiring Delay in MOSFET LSI. *IEEE J. Solid-State Circuits*, SC-18(4):418–426, August 1983.

[Sch84] B. Schmult. Partitioning Strategies for Multi-chip VLSI Microprocessors. Technical Report CMUCAD-84-26, Department of Electrical and Computer Engineering, Carnegie-Mellon University, February 1984.

[SDD+89] W. Smith, D. Duff, M. Dragomirecky, J. Caldwell, M. Hartman, J. Jasica, and M. d'Abreu. FACE Core Environment: The Model and its Application in CAE/CAD Tool Development. In *Proc. 26th Design Automation Conf.*, pages 466–471. ACM/IEEE, June 1989.

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

[SR89] Y. Saab and V. Rao. An Evolution-Based Approach to Partitioning ASIC Systems. In *Proc. 26th Design Automation Conf.*, pages 767–770. ACM/IEEE, June 1989.

[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 *Proc. 25th Design Automation Conf.*, pages 337–343. ACM/IEEE, June 1988.

[TI86] 2 Micrometer CMOS Standard Cell Data Book. Texas Instruments, 1986.

[TR89] R. R. Tummala and E. J. Rymaszewski, editors. *Microelectronics Packaging Handbook.* Van Nostrand Reinhold, 1989.

[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 Trans. on Computer-Aided Design*, 6(2):259–269, March 1987.

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

[VHD88] IEEE Standard VHDL Language Reference Manual. The Institute of Electrical and Electronics Engineers Inc., March 1988.

[Wen89] J. Weng. CSG: Control Signal Generator. Department of Electrical Engineering, University of Southern California, 1989. Internal Report.

[WP91] J. Weng and A. C. Parker. 3D Scheduling. In *Proc. 28th Design Automation Conf.*, pages 668–673. IEEE/ACM, June 1991.

[WT89] R. A. Walker and D. E. Thomas. Behavioral Transformation for Algorithmic Level IC Design. *IEEE Trans. on Computer-Aided Design*, 9(9):1115–1128, October 1989.

[Zim88] G. Zimmerman. A New Area and Shape Function Estimation Technique for VLSI Layouts. In *Proc. 25th Design Automation Conf.*, pages 60–65. ACM/IEEE, June 1988.

Appendix A

A Sample MABAL Output

MABAL is run in batch mode. The dataflow graph, the library and the schedule are given to MABAL as files. The optional information specified on files includes pre-existing bindings (assignments) for operations and values, initial resource (operator and register) allocation, and interconnect restrictions. The names of these files and other program parameters are passed to MABAL from the command line. The following is a script which shows the list of command-line options to MABAL and a sample MABAL output. Text typed by the user is shown in boldface.

```
USC-> mabal -help


-----------------------------------------------------------------

MABAL v1.008    Module And Bus ALlocator

Design Automation Group
Electrical Engineering -- Systems
University of Southern California

-----------------------------------------------------------------

This is the list of options :
         -debug   prints out a file helping debugging
 -log     prints out a program log
 -lat     takes the initiation interval from the next argument
 -o       output_file_name              (default: stdout)
```

```
 -s       schedule_file_name            (default: a.timing)
 -bo      op_binding_file               (default: a.op_binding)
 -bv      val_binding_file              (default: a.val_binding)
 -m       module_file_name              (default: a.mod)
 -mux     mux_file_name                 (default: a.mux)
 -d       dataflow_file_name            (default: a.dfg)
 -l       library_file_name             (default: a.lib)
 -nl      netlist_file_name             (default: a.nl)
 -poval   prim_out_val_file_name        (default: a.po_val)
 -gen     generosity factor             (default: 1.0)
 -nobus   don't use any buses
 -nolatch don't latch inputs
 -help    prints this message
```

USC-> **mabal -lat 3 -nobus -nolatch**

```
-----------------------------------------------------------------

MABAL v1.008 Module And Bus ALlocator

Design Automation Group
Electrical Engineering -- Systems
University of Southern California

-----------------------------------------------------------------

Working Directory for this run : ~kayhan/thesis/mabal/fir
Files used for this run :
Dataflow file  : a.dfg
Library file  : a.lib
Module file  : a.mod
Schedule file  : a.timing
Operation binding file   : a.binding
Value binding file  : a.val_binding
Netlist output file  : a.nl
Mux file  : a.mux
Output-value specification file  : a.po_val
Output file  : stdout

Initiation Interval is 3 clock cycle(s).
```

```
Rate of generosity is 1.000000.
Bus prevention request is in effect.
No inputs except loop variables are latched.


Lifetime analysis has been completed.
Levelizing and ordering of data flow graph has been completed.
11 library module(s)
49 node(s)
0 constant(s) or already stored input(s)
47 edge(s) read
24 value(s) except ONLY_ONCE values need to be stored in registers


Post Processing has been completed.

OPERATORS


operator add_1 , 5
nodes assigned : add7(3) add4(2) add1(1)
input structure 1 : mux , 3 incoming wires(s)
Quality of Sharing: 1.00
input structure 2 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is 100 %

operator add_2 , 4
nodes assigned : add8(3) add5(2) add3(1)
input structure 1 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
input structure 2 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is 100 %

operator add_3 , 3
nodes assigned : addf(6) addd(5) addb(4)
input structure 1 : mux , 2 incoming wire(s)
Quality of Sharing: 1.50
```

```
input structure 2 : mux , 2 incoming wire(s)
Quality of Sharing: 1.50
output structure 1 : none
Utilization for module is 100 %

operator add_4 , 2
nodes assigned : addg(6) adde(5) addc(4)
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 3.00
input structure 2 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is 100 %

operator add_5 , 1
nodes assigned : adda(3) add6(2) add2(1)
input structure 1 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
input structure 2 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is 100 %


Utilization for module type add is 100 %

operator mul_1 , 8
nodes assigned : mul7(4) mul4(3) mul1(2)
input structure 1 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
input structure 2 : none , 1 incoming wire(s)
Quality of Sharing: 3.00
output structure 1 : none
Utilization for module is 100 %

operator mul_2 , 7
nodes assigned : mul8(5) mul5(4) mul3(3)
input structure 1 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
input structure 2 : mux , 3 incoming wire(s)
Quality of Sharing: 1.00
```

```
output structure 1 : none
Utilization for module is 100 %

operator mul_3 , 6
nodes assigned : mul6(4) mul2(2)
input structure 1 : mux , 2 incoming wire(s)
Quality of Sharing: 1.00
input structure 2 : none , 1 incoming wire(s)
Quality of Sharing: 2.00
output structure 1 : none
Utilization for module is  66 %


Utilization for module type mul is  88 %

operator register_1 , 26
values assigned :  add7_out  add4_out  add1_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 3.00
output structure 1 : none
Utilization for module is 100 %

operator register_2 , 25
values assigned :  add6_out  add2_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 2.00
output structure 1 : none
Utilization for module is 100 %

operator register_3 , 24
values assigned :  add3_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  66 %

operator register_4 , 23
values assigned :  add5_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
```

```
Utilization for module is  66 %

operator register_5 , 22
values assigned :  mul4_out  mul1_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 2.00
output structure 1 : none
Utilization for module is  66 %

operator register_6 , 21
values assigned :  mul6_out  mul2_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 2.00
output structure 1 : none
Utilization for module is  66 %

operator register_7 , 20
values assigned :  root_17
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  33 %

operator register_8 , 19
values assigned :  root_22
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  33 %

operator register_9 , 18
values assigned :  root_23
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  33 %

operator register_10 , 17
values assigned :  root_24
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
```

```
output structure 1 : none
Utilization for module is  66 %

operator register_11 , 16
values assigned :  add8_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  66 %

operator register_12 , 15
values assigned :  mul8_out  mul5_out  mul3_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 3.00
output structure 1 : none
Utilization for module is 100 %

operator register_13 , 14
values assigned :  adda_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  33 %

operator register_14 , 13
values assigned :  mul7_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 1.00
output structure 1 : none
Utilization for module is  66 %

operator register_15 , 12
values assigned :  addg_out  adde_out  addc_out
input structure 1 : none , 1 incoming wire(s)
Quality of Sharing: 3.00
output structure 1 : none
Utilization for module is 100 %


Utilization for module type register is  66 %
```

```
Total Module Cost :     42840.00
Total Interconnection Cost :      6624.00
Total Cost :     49464.00
Total Number of muxes : 13
Total Number of mux inputs : 36
Total Number of 2to1 1-bit muxes : 368
Total Number of 1 bit bus drivers : 0
Total Number of buses : 0
Total Number of 1-bit nets : 1088(1248)
Total Number of 2-point 1-bit nets : 1088(1248)
Total Hardware Utilization is  67 %

---------------------------------------------------------------
Upper Bound Area Calculations For the Final Resource Allocation

Multiplexer cost 18.00

Module Name      Area      Allocated # of Uses Mux Count
--------------- --------- --------- --------- ---------
add               1200.00       5         15      10 * 2
mul               9800.00       3          8       5 * 2
register           496.00      15         24       9 * 1

Upper bound for this resource allocation
MUX at all inputs : 11232.00
Number of mux trees at inputs : 39
Module cost : 42840.00
---------------------------------------------------------------

Resulting interconnect cost is 58 % of the upper bound.

Initialization Time : 180 msec
Data Input and Pre-processing Time : 60 msec
Computation Time : 360 msec
Host System : poisson.usc.edu
Host System Architecture : Sun 4/330
```

Appendix B

A Sample BAD Output

BAD is used interactively. The detailed information about the dataflow graph, the library, the partitioning, chip packaging information and the memory hierarchy are specified via files. The clocking scheme, the data path architecture style, and performance and delay constraints are specified interactively. BAD predicts possible implementations and lists the prediction results. Predictions can be performed on multiple user-specified partitions simultaneously. In addition to the summary output, there is a machine-readable output file which has the details of the prediction results reported. The following are the program options and a sample summary output. The user actions are shown in boldface.

USC-> **bad -help**

```
BAD v1.5 Behavioral Area-Delay Predictor

 Program options

-np      No pipelined data path predictions
-l       Long listing for a.dump file
-plot    Generates matlab files for prediction results
-ni      Non-inferior designs only
-fp      Feasible predictions only
-help    Prints this message
```

```
USC-> bad

BAD v1.5 Behavioral Area-Delay Predictor

Design Name:  AR FILTER
Clock Cycle:  300
Data Path Clock (as an integer multiple of the clock):  10
Data Transfer Clock (as an integer multiple of the clock):  1
Performance Constraint (Delay between initiations):  60000
Circuit Delay Constraint (Delay from input to output):  60000
Prob.  of Acceptance for Area:  1.0
Prob.  of Acceptance for Delay:  0.8
Prob.  of Acceptance for Latency:  1.0
Architecture (1:  single-cycle operations, m:  multi-cycle):  1
Options: Feasible Non-inferior Predictions

AR FILTER
16 operator types
16 module types
34 operations
62 edges
0 constants
1 chips
1 partitions
0 memory blocks

Number of prediction points attempted for 1 partitions : 207
Number of feasible prediction points : 3

Partition      Area    Latency  Delay    Clock Cycle
-----------------------------------------------------
      1        80968    4944     4944       309
      1        87036    4944     5253       309
      1        84584    4944     5562       309

Input Time : 160 msec
Computation Time : 170 msec
Host system : almaak.usc.edu
Host System Architecture : Sun 4/460
```

Appendix C

A Sample CHOP Output

CHOP is used interactively. The detailed information about the dataflow graph, the library, the partitioning, chip packaging information and the memory hierarchy are specified via files. The clocking scheme, the data path architecture, and performance and delay constraints are specified interactively. CHOP evaluates the given partitioning and asks the user for any modifications of the partitioning. In addition to the summary output, there is a machine-readable output file which has the details of prediction results reported for the final partitioning evaluated. The following are the program options and a sample summary output. The user actions are shown in boldface.

USC-> **chop -help**

```
CHOP v1.5 System-Level Interactive Partitioner

 Program options

-np      No pipelined data path predictions
-l       Long listing for a.dump file
-enum    Use enumeration in partitioning mode
-fenum   Use full enumeration in partitioning mode
         Default is heuristic
-2       Initial automatic 2-way partitioning
-help    Prints this message
```

```
USC-> chop

CHOP v1.5 System-Level Interactive Partitioner


Design Name:  AR FILTER
Clock Cycle:  300
Data Path Clock (as an integer multiple of the clock):  10
Data Transfer Clock (as an integer multiple of the clock):  1
Performance Constraint (Delay between initiations):  60000
Circuit Delay Constraint (Delay from input to output):  60000
Prob.  of Acceptance for Area:  1.0
Prob.  of Acceptance for Delay:  0.8
Prob.  of Acceptance for Latency:  1.0
Architecture (1:  single-cycle operations, m:  multi-cycle):  1



Options : Feasible Predictions
        : Partitioning by Iterative Heuristic

AR FILTER
16 operator types
16 module types
34 operations
62 edges
0 constants
2 chips
2 partitions
0 memory blocks


Number of prediction points attempted for 2 partitions : 207
Number of feasible prediction points : 84

Number of combinations tried for the partitioning : 41
Number of non-inferior feasible partitioning points : 2
```

```
------------------------------------------------------
Latency  Delay  Clock Cycle     Fraction of used area
                                Chip[ 1]    Chip[ 2]

     68     68          309        0.95        0.87
     40     78          309        0.88        0.87
------------------------------------------------------

Do you want to modify the partitioning ?  y

Enter node name, partition number to a line
Enter "end" to end the changes

>> add9 1
>> add10 1
>> end


Number of prediction points attempted for 2 partitions : 207
Number of feasible prediction points : 82

Number of combinations tried for the partitioning : 38
Number of non-inferior feasible partitioning points : 1

------------------------------------------------------
Latency  Delay  Clock Cycle     Fraction of used area
                                Chip[ 1]    Chip[ 2]

     40     67          309        0.93        0.87
------------------------------------------------------


Do you want to modify the partitioning ?  n

Do you want to save the partitioning ?  y


Input Time : 130 msec
Computation Time : 810 msec
Host system : almaak.usc.edu
Host System Architecture : Sun 4/460
```