High-Level Synthesis with Pin Constraints for Multiple-Chip Designs

Yung-Hua Hung

CENG Technical Report 92-22

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

HIGH-LEVEL SYNTHESIS WITH PIN CONSTRAINTS FOR MULTIPLE-CHIP DESIGNS

by

Yung-Hua Hung

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

December 1992

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

This dissertation, written by

Yung-Hua Hung

under the direction of h.i.s..... 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

Barbara Solomon

Dean of Graduate Studies

Date December 2, 1992

DISSERTATION COMMITTEE

Alice C. Parker

Chairperson

Sandeep Gupta

Ming-Deh Huang

Dedication

To my parents, my wife, and my son.

Acknowledgements

I take this opportunity to express my sincere appreciation to my advisor, Prof. Alice C. Parker, for her valuable guidance and continued encouragement. Without her, this thesis would not exist.

I would also like to thank Profs. Ming-Deh Huang and Sandeep Gupta for serving on my dissertation committee, and Profs. Sarma Sastry, Michel Dubois, and Viktor Prasanna for serving on my guidance committee.

I thank all my friends and colleagues in USC for their warm friendship and various help. In particular, I like to mention Chih-Tung Chen, Pravil Gupta, Jen-Pin Weng, Shiv Prakash, Atul Ahuja, Sen-Pin Lin, Charles Njinda and Mary Zittercob.

I have received support from Computer & Communication Research Laboratories (CCL), a division of Industrial Technology Research Institute (ITRI), and encouragement from my friends and colleagues, there, including my former managers, Dr. Ting-Hao Chao and Dr. Chih-Tien Hsing.

To my little honey - Elbert - your welcome interruptions kept my goal in perspective.

Finally, I would like to thank my wife, Lan-Hsiang, for her patience, support, and sacrifices throughout my graduate study.

Contents

List Of Figures

List Of Tables

Abstract

In this thesis, data-path synthesis techniques for multiple-chip pipelined systems from partitioned behavioral-level specifications are presented. Most digital designs are too large to fit into a single chip. Design of such systems is becoming harder and harder because the complexity of digital systems is increasing drastically and synthesis of multiple-chip digital systems will become crucial.

The problem of synthesizing multiple-chip digital systems is complicated by several aspects. First, output and input of a value between partitions must take place in the same transfer cycle. Also, the values transferred across chips may have a variety of bit widths, which makes it difficult to utilize I/O pins efficiently. Finally, no devices will be allowed on the off-chip connection, and this makes all traditional connection binding techniques no longer applicable.

A class of partitioning, called *simple partitioning*, is identified, in which it is proven that there exists an interchip connection without communication conflicts for a schedule without I/O pins conflicts. For the other class of partitioning, *general partitioning*, the problem is divided into two subproblems, namely *interchip connection synthesis* and *scheduling*. The problem is approached in either order: interchip connection synthesized before or after scheduling. The experimental results obtained by both approaches are compared. An Integer Linear Programming (ILP) formulation and a heuristic search technique are developed for the subproblem of interchip connection synthesis.

Chapter 1

Introduction

High-level synthesis [MPC88] is the process of transforming a behavioral description into a structural design that can implement the behavior. In general, the transformation is a one-to-many mapping. That is, there are a number of implementations performing the given behavior. One of the tasks of synthesis is to find the structural implementation that meets a set of constraints while achieving a design goal.

Because of the complexity of high-level synthesis, the task of synthesis is usually divided into two sequential sub-tasks: synthesis of the data path, and synthesis of the control path. The core of traditional data path synthesis is further divided into three simpler tasks, namely scheduling, module allocation, and resource binding. Scheduling assigns operations to appropriate control steps (states). Module allocation determines the number and types of hardware modules required. Resource binding assigns operations and data values to specific hardware resources. Control synthesis refers to the automatic generation of a controller which provides the control signals for sequencing events in the data path.

In general, the input to data-path synthesis tools is modeled in an intermediate format, such as the Value Trace (VT) [McF78], Design Data Structure (DDS) [KP85], dataflow/control-flow (dacon) [Tri87], and Sequencing Intermediate Form (SIF) [MK88], sometimes referred to as a Control Data Flow Graph (CDFG), which can be obtained by compiling a Hardware Description Language (HDL),

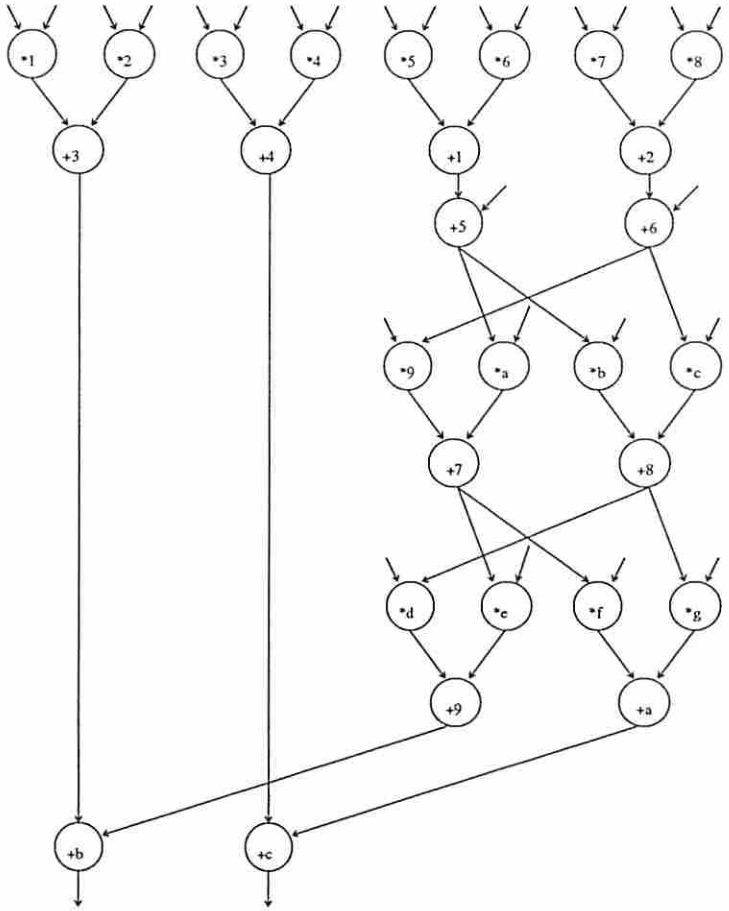


Figure 1.1: An example of a CDFG

such as ISPS [Bar81], VHDL [VHDL88], and HardwareC [MK88]. A CDFG is a directed graph, in which the nodes represent operations to be performed, and the arcs represent their dependencies. An example of a CDFG is shown in Figure 1.1. The output from data-path synthesis tools is usually a register transfer level (RTL) design. An RTL data path consists of operators and registers interconnected via multiplexers, buses, and wires.

1.1 Motivation

Most modern digital systems can hardly fit into a single chip, even though the number of components that can be fabricated on a single chip has been increasing drastically in recent years, because the complexity of digital systems has also increased significantly in the same period of time. In such a design composed of multiple chips, the interchip communication is an important issue and should be considered at an early stage of the design process. However, most current high-level synthesis research [PP88, PK89, PK91, HHL91] pays no attention to the issue of communication across chips.

Pin constraints should also be taken into account during the process of high-level synthesis for multi-chip designs, as all IC chips only have a finite number of pins available for off-chip communication, although this number has increased over the years. However, many current data path synthesis tools do not consider pin constraints. In the process of scheduling, they make an implicit assumption that every input value is ready whenever it is required[1]. In general, the assumption is not true because the number of input pins is limited and inputs cannot be assured to be available at any time. Some tools [CK86, NT86, NK90] can take pin limitations into account implicitly by the use of time constraints on inputs and outputs.

1.2 Problem Statement Overview

In our approach, partitioning of a design onto multiple chips is performed before the behavioral synthesis process. Using predictions, the behavioral partitioner, such as CHOP [KP91], partitions the behavioral specification into a number of clusters in such a way that the synthesized multi-chip design will likely be feasible with respect to the given constraints.

[1]It can be stated in another way that all input values are available before any functional operation can execute.

The intent of this research is to propose an approach to multi-chip data path synthesis given a partitioned behavioral specification. A pipelined schedule and pin allocation consisting of multiple chips is to be produced, with each chip corresponding to a cluster of operations in the behavioral specification. Also, a set of communication buses between the chips will be synthesized, and the values to be transferred will be scheduled and assigned to the pins and buses. The designs produced will satisfy user-supplied constraints, including the number of I/O pins on individual chips.

1.3 Related Work

List scheduling [Girc84, PP88, PG86] has been widely used, and attempts to minimize the execution time of the design where hardware resource constraints are specified. In the algorithm, *ready* operations, whose precedent operations have been scheduled, are sorted according a heuristic priority function, and scheduled in the next control step until a resource conflict occurs, and a control step is advanced. There are many variations of list scheduling. The variation is mainly due to how the heuristic priority function is defined.

Force-Directed Scheduling (FDS), developed by Paulin [PK89], tries to reduce the number of hardware resources by balancing the concurrency of the operations among control steps such that a global time constraint, expressed as the maximum number of control steps, is met. In each iteration, Distribution Graphs (DGs) for all *operation types* are computed, from which *force* can be derived. Forces are calculated for all operations at all feasible control steps. The operation - control step pair yielding the lowest (most negative) force, that is, most likely balanced, is selected and assigned.

An *Integer Linear Programming* (ILP) formulation has been applied to the data-path synthesis problem. Hafer and Parker [HP83] are the first ones who modeled the data-path synthesis problem with a formal method, and mapped a behavioral specification into a scheduled and allocated data path by using Mixed

Integer Linear Programming (MILP). Lee et al. [LHL89], Hwang et al. [HHL90], and Gebotys et al. [GE91] have applied ILP to the scheduling problem in data-path synthesis.

In the literature, only two research projects explicitly on high-level synthesis of multiple-chip designs are found. De Micheli et al. [KM90, GM90] applied an iterative approach consisting of three stages, namely resource binding, scheduling, and partitioning. In their approach, resource binding is performed before scheduling. In the binding stage, operations are bound to specific resources. There will be a large number of binding configurations. In each iteration, one of the binding configurations is selected, either by an exhaustive search or a heuristic search with some cost criteria. Once a resource binding is selected, the operations bound to the same resource are serialized to resolve resource conflicts. A branch-and-bound approach is used to explore the alternatives. Then, a SIF model, a variation of a CDFG containing binding and serialization information, is passed to a partitioning program to be partitioned into multiple chips. Their system can only handle non-pipelined designs. Moreover, pin sharing among I/O operations is not considered, as the pin cost, which is used as one of the constraints in the process of partitioning, is computed by simply adding the costs of all I/O operations in the partition. So, the design produced by this approach will require many more I/O pins than necessary.

Gebotys [Gebo92] approached the problem of synthesizing multi-chip designs by formulating partitioning, scheduling, and allocating hardware (functional units, I/O pins, and interchip buses) as an ILP. Since she made the assumptions that every interchip bus is connected to all of the chips and every value transferred off-chip has the same bit width, the interchip communication bus structure does not need to be determined. Instead, only the numbers of input ports, output ports, and interchip buses are to be determined. These assumptions will be fine for two-chip designs. However, it would require more I/O pins than necessary for systems which contain more than two chips. The larger number of chips in a system, the more I/O pins are likely to be wasted under the assumptions. Also, values transferred

across chips may have different bit widths. In addition, the problem size will be limited by the run time to solve the ILP. When the number of integer variables in an ILP reaches a few hundred, the run time will grow drastically with the number of integer variables.

1.4 Thesis Outline

Chapter 2 gives an overview of the research, and addresses the difficulties which arise during the multi-chip pipelined data-path synthesis process. In Chapter 3, a class of partitioning, called simple partitioning, is identified. The technique to synthesize multi-chip designs with a simple partitioning is presented. For systems with a general partitioning, the problem of synthesizing multi-chip designs is divided into 2 subproblems: (1) interchip connection synthesis, and (2) scheduling. Chapters 4 and 5 describes different techniques to solve these subproblems in different orders, interchip connection synthesis before and after scheduling, respectively. The experimental results produced by these approaches are compared. Chapter 6 presents a technique to deal with extended cases in which a communication bus can be used to transferred multiple values in a cycle by using different portions of the bus. In Chapter 7, some extensions to the research are discussed. Finally, conclusions and future research directions are summarized in Chapter 8.

Chapter 2

Research Overview

In this chapter, a brief definition of the research problem is given first. Then, some restrictive assumptions which will be made are discussed. Following this several primary problem issues of the research are addressed. Finally, the difficulties being encountered are shown with simple examples.

2.1 Problem Statement

The research problem is to synthesize synchronous multiple-chip pipelined designs from partitioned behavioral specifications, achieving a user-specified goal, while satisfying user-given constraints. The limited number of I/O pins will be taken into account. As shown in Figure 2.1, the problem inputs include

- a CDFG which has been partitioned into a number of partitions, each of which will be implemented on a separate chip,
- a set of hardware modules, with parameters of cost and delay, which will be used to implement the operations in the CDFG,
- constraints on cost and pin count for each individual chip, or a performance requirement of the whole design, and
- a specified goal of maximizing performance of the whole design, or minimizing the total cost of the design, depending on what constraints are given.

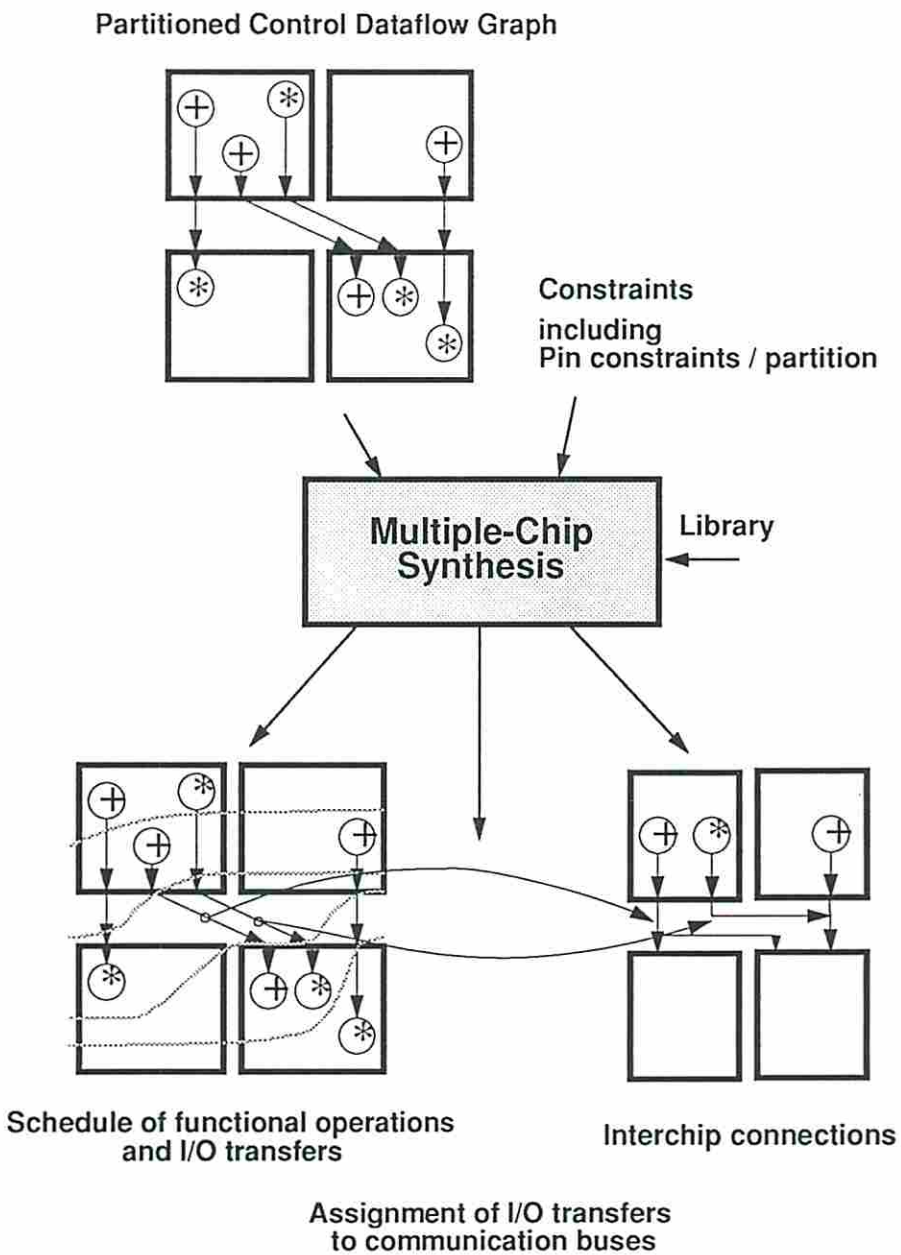


Figure 2.1: Multiple-chip Synthesis

The problem outputs include

- a schedule for the functional operations and I/O transfers,
- a set of interchip communication buses, and
- an assignment of I/O operations to communication buses.

For multiple-chip designs, it is beneficial to have functional units of all partitions operating concurrently to fully utilize multiple hardware resources. One of the effective ways to achieve this is to design such systems as pipelined systems, at least from the view of the system. Individual partitions can be implemented in either a non-pipelined or pipelined manner.

In this research, only synthesis of pipelined systems is discussed since scheduling of a non-pipelined design can be regarded as a special case of scheduling of a pipelined design with initiation rate equal to or greater than the pipe length.

2.2 Problem Assumptions

In order to focus on the major problem issues and to keep the complexity of the problem to a manageable level, several restrictive assumptions are made. These include selection of a clocking scheme, determination of a Control Data Flow Graph (CDFG) representation, and consideration of I/O transfers, functional operations, and the control mechanism.

- *Clocking Scheme*

  A global clock is used for all partitions, and to synchronize all functional operations and I/O transfers. The period of the clock will be set by the user or by prior software and input to the program. Two minor clocks, one for data manipulation and one for I/O transfers, can also be handled. In this case, both clocks must be derived from a global clock, and the initiation interval of the design must be a multiple of the period of the global clock.

- *Control Data Flow Graph*

  A flat CDFG without internal loops is assumed. A hierarchical CDFG must be flattened beforehand. Internal loops with a fixed or maximum iteration count should be unwound. Internal loops with indeterminate iteration counts cannot be unwound, and so will not be covered in the research since they make the design of a pipelined system very difficult and for some real-time systems, impossible. The CDFG can contain nested conditionals as well as data recursive edges.

- *I/O Transfers*

  Interchip communication is performed synchronously. A value produced in a partition is transferred directly to the requesting partition at a predetermined time. In other words, a value is not transferred to the requesting partition through any other partition(s). I/O transfers are activated at the beginning of a clock cycle, and must be completed in a single cycle. When two minor clocks are used, I/O transfers is activated at the beginning of a minor clock cycle and must be completed in the same minor cycle.

- *Functional Operations*

  For each operation type, there exists exactly one functional unit that can perform the operation in the module set. In other word, module selection has been done before scheduling. Each partition can have its own module set. All functional units are combinational. The delays of functional units can be greater than the period of a clock cycle. Chaining of operations and multi-cycle operations are allowed. I/O transfer and functional operations can also be chained. The module set can be specified by a behavioral-level partitioning tool [KP91].

- *Control Mechanism*

  A distributed control mechanism is assumed. There are several disadvantages to applying a central controller approach. The bandwidth of data transfers across chips would be reduced significantly because a large number of I/O

pins would have to be reserved for control signals. Also, control signals traveling off-chip would cause long delays in the control path. Both of these could result in designs with poor performance.

2.2.1 I/O Operation Model

For a system consisting of a number of partitions, a value produced in a partition may be used in other partitions. So, an I/O transfer is required when an operation in a partition requires a value produced in another partition. The criteria of an I/O transfer are

- that the output operation of the value from a partition and the input operation of the value to the other partition must take place in the same control step,[1] and
- that there is a *free*[2] communication path between these partitions in the control step.

In our system, a value can be output once or several times when a value produced in a partition is required by two or more partitions. For the case that a value is to be output once, the value can be output to a communication bus and all of the requesting partitions can input that value in the same control step. This requires that all of requesting partitions have their input ports connected to the communication bus. In this case, only one communication bus will be allocated for only one cycle. On the other hand, for the case that a value is to be output several times, the value can be output to different communication buses in the same or different control steps and the requesting partitions can input the value from different communication buses in the corresponding control steps. This has the flexibility that the requesting partitions are not required to have their input ports connected to the same communication bus. In this case, more than one communication bus

[1] Recall that we have assumed that values are transferred directly.

[2] A communication path is said to be *free* in a control step if it has not been allocated for data transfer in the control step.

will be allocated to transfer the value. Note that, for each partition, a value needs to be input only once and stored even it is used by more than one operation in a partition. In this case, I/O pins can be saved at the expense of chip area due to an extra register used to store the input value. However, the extra register does not have a first-order effect on the chip area.

An I/O operation is modeled as a single I/O operation node which consists of both one output operation for a partition and one input operation for another partition, since an output operation for a partition is always accompanied in the same control step by an input operation for another partition. Scheduling an I/O operation node in a certain control step means that the output and input operations it denotes will take place in that control step. For a value required by more than one partition, there are several I/O operation nodes, one for each requiring partition, since a value is allowed to be output several times if necessary. In order to be able to identify the I/O operations transferring the same value, each I/O operation is associated with the name of the value it will transfer.

A cost, called *bit-width*, is associated with each I/O operation to indicate the bit width of the value it will transfer. There is also a delay associated with each I/O operation. The I/O operation delay is defined as the duration from the beginning of the clock cycle where the I/O operation is activated to the time the value becomes valid at the destination. An estimated delay is assumed for all I/O operations, because the actual delays of output drivers and interchip connections are unknown *a priori*.

2.3 Primary Problem Issues

A number of major problem issues are to be explored in this research. These include

- pin constraints,
- interchip connections, and

- time division I/O multiplexing.

Each of the research topics will be described in the following sections.

2.3.1 Pin Constraints

Values produced in a chip and used in another chip require I/O pins for data transfers. Pins are resources, much like functional units are resources, and I/O transfers are modeled as operations allocated for the pins. However, the problem is complicated by the fact that a value is generally composed of several bits, so more than one pin will be allocated for transferring a value. In a pipelined design with an initiation rate of L, the operations which are scheduled in the same *control step group*[3] will execute overlapped in time, and cannot share the same hardware resources. So, an I/O pin can be allocated at most L times, once in each control step group. This can be modeled as a multiple bin packing problem, where the bin capacity represents the number of available pins and the number of bins is the initiation rate of the design.

2.3.2 Interchip Connections

There must be interconnection among chips to allow interchip communication. The synthesis of interchip connections is quite different from and harder than the binding of connections within a chip. The complication arises from the fact that no switching devices, such as MUX's or tri-state buffers, can be used on the off-chip connections. All traditional datapath connection binding techniques [KP90a, WS89, HCLH90, PK90] can no longer be applied to the problem since all of them assume that MUX's and/or tri-state buffers can be used on connections. The difficulty of the problem will be discussed in Section 2.4.

[3]For a pipelined design with an initiation rate of L, the set of control steps $k, k+L, \ldots$ will be referred to as control step group k.

2.3.3 Time Division I/O Multiplexing

A value transferred across chips may have a large bit width. A large number of pins will be required to transfer the value as a whole. Sometimes, it may be beneficial to split a value into several sub-values, each having a smaller number of bits, and to have these sub-values transferred in a number of cycles. This will be referred to as *time division I/O multiplexing.*

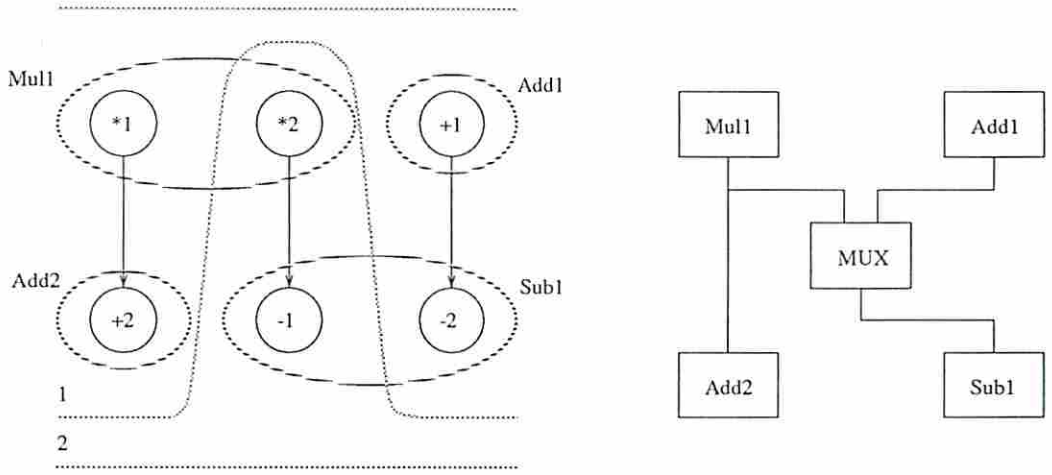
Although the number of I/O pins required may be reduced with time division I/O multiplexing, larger chip area and degraded performance can result because more register control signals are required to latch the multiplexed values and/or more control steps are needed. This kind of tradeoff should be considered at an early stage in the synthesis process.

2.4 Research Difficulties

Most of the research difficulties in multi-chip synthesis are caused by interchip communications. These difficulties will be described in detail in this section. For the ease of discussion, some simple examples will be used to demonstrate these difficulties.

Scheduling I/O operations is quite different from and harder than scheduling functional operations because no switching elements are allowed on interchip connections, while switching devices can be used on intrachip connections. The discrepancy will be illustrated by two CDFG fragments which have the same data dependence, one for functional operations and the other for I/O operations. For the ease of comparison, each value is assumed to be one bit wide.

Figure 2.2(a) shows the portion of a CDFG containing only functional operations, which can be scheduled in 2 control steps, if there are 1 multiplier, 1 subtractor, and 2 adders available for these operations. The numbers on the left of the figure are control steps, and the operations between two shaded lines are scheduled in that control step. The operation binding for the schedule is indicated by dotted ellipses, and the operations within an ellipse are assigned to the same



(a) A portion of scheduled CDFG

(b) The corresponding RTL structure

Figure 2.2: The scheduling and binding of a CDFG containing only functional operations

functional unit. The corresponding RTL design is shown in Figure 2.2(b). Note that a multiplexor has been inserted before the input of *Sub1*.

On the other hand, a similar CDFG with I/O operations is shown in Figure 2.3. Assume that each of the chips P_a, and P_b has 1 output pin, and each of the chips P_c, and P_d has 1 input pin, which is similar to the assumption made in Figure 2.2. If I/O operations *OP1, IP1, OP3*, and *IP3* are scheduled in control step 1, as shown in Figure 2.4(a), I/O operations *OP2*, and *IP2* are unable to be scheduled in any control step under the pin constraints given. This can be illustrated by the partial RTL design, as shown in Figure 2.4(b), corresponding to the partial schedule. Transferring value V_2 requires a connection between the output pin of partition P_a, and the input pin of partition P_d, as indicated by a dotted line in Figure 2.4(b). However, the connection would cause the transfers of the values V_1, and V_3, which occur in the same control step to conflict even though partitions P_a, and P_d have 1 output pin, and 1 input pin, respectively, unallocated in control step 2. For the given constraints, the design requires 3 control steps, each for transferring one value. This leads to the following result:

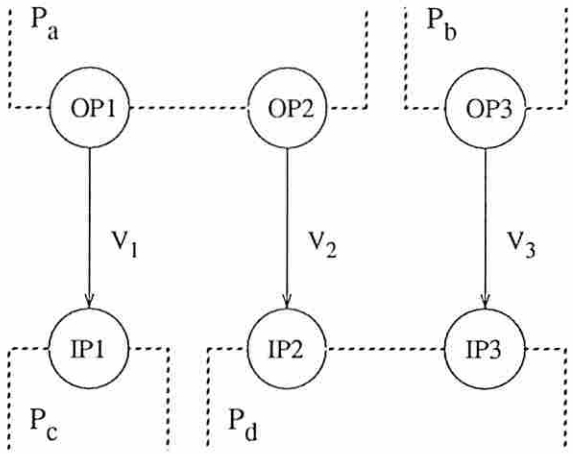


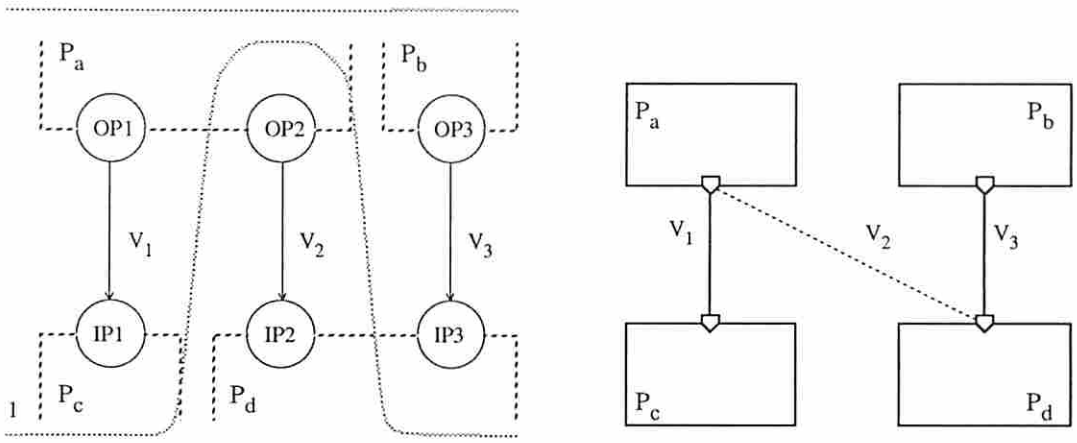
Figure 2.3: A portion of CDFG containing I/O operations

> Scheduling an I/O operation in a control step in which there are unallocated I/O pins does not guarantee that there will be no communication conflict in the schedule. Moreover, scheduling an I/O operation in a control step, even with no communication conflict in that control step, can make achieving a completed schedule impossible.

Even a restricted case, a *simple* partitioning,[4] where scheduling an I/O operation in a control step in which there are unallocated I/O pins guarantees no communication conflict in the control step, may not lead to a schedule (as we show later). The problem is due to the requirement that the input and output operations for a value must occur in the same control step.

Consider the partial CDFG shown in Figure 2.5. The assumption of one bit per value is made. Suppose that partition P_a has 2 output pins, and partitions P_b, and P_c have 2, and 1 input pins, respectively. Also, an initiation rate of 2 is assumed. If the I/O operations for transferring values V_1, and V_2 are scheduled in control step s, we will never be able to complete the schedule. The I/O operations for transferring values V_3, and V_4 cannot be scheduled in any of the control steps $s, s+2, \ldots$ because partition P_a has no unallocated output pins in those control

[4]See Definition 3.2



(a) Partially scheduled CDFG

(b) Partial inter-chip connection

Figure 2.4: A partial schedule and interchip connection for Figure 2.2

steps, whereas one of them must be scheduled in one of those control steps because partition P_c has only one input pin. So, one of the I/O operations for transferring values V_1, and V_2 should be scheduled in one of control steps $s, s+2, \ldots$, while the other should be scheduled in one of the control steps $s+1, s+3, \ldots$. One possible schedule is that the I/O operations for transferring values V_1, V_2, V_3, and V_4 are performed in control steps $s, s+1, s+1$, and $s+2$, respectively.

Values transferred across chips may have a variety of bit widths. How to utilize I/O pins efficiently makes the problem even harder because transferring different values may require using different numbers of I/O pins.

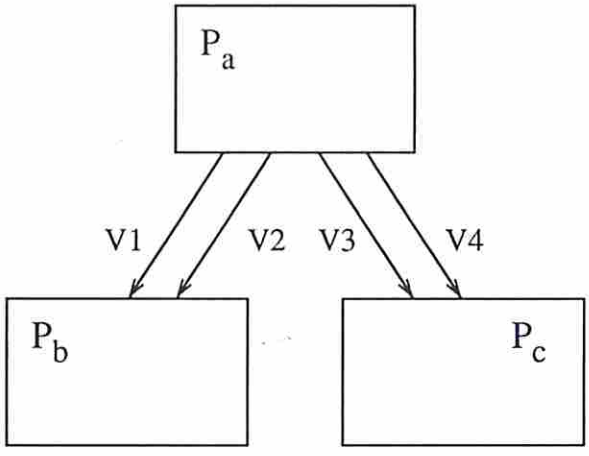


Figure 2.5: A portion of *simple* partitioning of a CDFG

Chapter 3

Synthesis for Designs with a Simple Partitioning

In this chapter, the definition of a *simple partitioning* is given. For a system with a simple partitioning, the interchip communication problem can be reduced to a pin allocation problem. That is, there exists an interchip connection with no communication conflicts for a schedule without I/O pin conflicts. An integer-linear programming (ILP) formulation for this pin allocation subproblem is given in Section 3.1. Section 3.2 describes how *list scheduling* is adapted to schedule a system with a simple partitioning and pin constraints. Before each I/O operation is scheduled, the feasibility of the ILP is checked to determine whether a feasible pin allocation still exists. The feasibility of the ILP is determined by applying the Dual Simplex algorithm to the ILP tableau. Section 3.3 shows how the feasibility of the ILP is determined and shows how the ILP tableau can be updated incrementally as the scheduling progresses. Last, experimental results are given in Section 3.4.

Definition 3.1 In a partitioning, partition P_a is said to *drive* partition P_b if a value required by an operation in partition P_b is produced by an operation in partition P_a. In this case, partition P_a is a *driver* of partition P_b. Partition P_b is *driven* by partition P_a.

Definition 3.2 A partitioning is said to be *simple* if it satisfies all of the following conditions:

1. Every partition drives at most two partitions.

2. Every partition is driven by at most two partitions.

3. If a partition is driven by two partitions, its drivers drive no other partitions.

4. If a partition drives two partitions, it is the only driver of these two partitions.

An example of a simple partitioning is given in Figure 3.1.

3.1 Pin Allocation Subproblem

A pin allocation is an allocation of I/O pins for all I/O operations in some control step groups. Recall that an I/O operation is composed of an output operation of a partition and an input operation of another partition. So, two sets of I/O pins will be allocated for an I/O operation in the same control step.

Definition 3.3 *Pin Allocation Problem*
Given

- the number of I/O pins (used for data transfers) for each partition,
- initiation rate L (L control step groups),
- a set of I/O operations, and
- a partial pin allocation.

The pin allocation problem is to determine if a valid pin allocation still exists. That is, I/O pins can be allocated for every I/O operation in some control step group.

3.1.1 The ILP Formulation

Suppose the CDFG is divided into N partitions $1, \ldots, N$, and that a pipelined system with an initiation rate of L is to be designed. An additional pseudo partition, partition 0, reflects the outside world and is used to model the pin constraints of

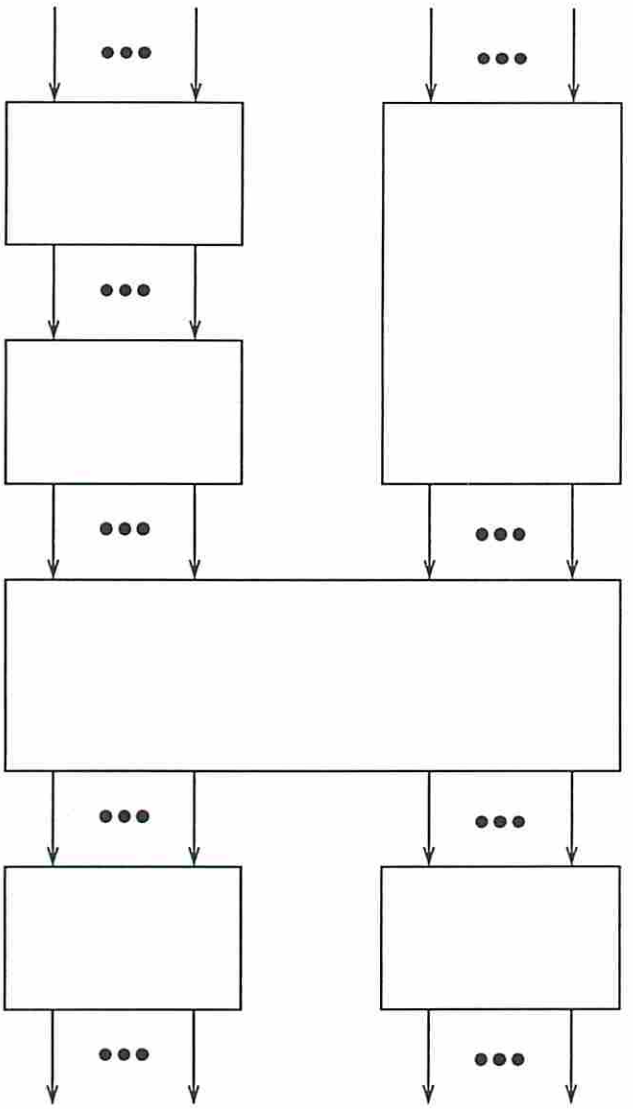


Figure 3.1: A *simple* partitioning example

the whole system. It contains only input operations and output operations. The input operations are used for outputs from the system, and the output operations are used for inputs to the system. The numbers of input pins, and output pins available for partition 0 are the output pins and input pins, respectively, of the system. The notations and variables to be used in the formulation and discussion are

- L : Initiation rate of the design.
- N : Number of partitions.
- $W = \{ w \mid$ every I/O operation $w \}$.
- $W_v = \{ w \mid$ every I/O operation w used to transfer value $v \}$.
- $IS_i = \{ w \mid$ every I/O operation w used to input a value to partition $i \}$.
- $OS_j = \{ v \mid v$ is a value output from partition $j \}$.
- B_w : Bit width of I/O operation w.
- B_v : Bit width of value v.
- I_i : Total number of input pins in partition i.
- O_j : Total number of output pins in partition j.
- T_i : Total number of pins used for data transfers in partition i. Pins used for power and control lines are excluded.
- o_j : An integer variable specifying the number of pins used for output from partition j.
- $x_{w,k}$: A binary variable denoting that I/O pins can be allocated for I/O operation w in control step group k.
  $x_{w,k} = 1$ if I/O pins can be allocated for I/O operation w in control step group k;
  $x_{w,k} = 0$ otherwise.

The pin allocation subproblem can be formulated as follows:

$$\text{Maximize } 0 \tag{3.1}$$

subject to

$$\sum_{w \in IS_i} B_w x_{w,k} \leq I_i, \qquad \text{for } 0 \leq i \leq N, 0 \leq k < L; \tag{3.2}$$

$$\sum_{v \in OS_j} B_v \max_{w \in W_v} \{x_{w,k}\} \leq O_j, \qquad \text{for } 0 \leq j \leq N, 0 \leq k < L; \tag{3.3}$$

$$\sum_{k=0}^{L-1} x_{w,k} \geq 1, \qquad \text{for } w \in W. \tag{3.4}$$

It is assumed in the formulation that the pins of each partition have been divided into input pins and output pins. A trivial objective function 3.1 is used because only the feasibility of the constraints is of interest. Constraint 3.2 states that, for each partition, the total number of bit widths of the input operations scheduled in control step group k should not exceed the number of input pins available. Constraint 3.3 states that the total number of bit widths of the output operations scheduled in control step group k should not exceed the number of output pins available. The maximum of $x_{w,k}$ is used instead of the summation of $x_{w,k}$ because only one output slot is required for the I/O operations transferring the same value being scheduled in the same control step. Constraint 3.4 ensures that I/O pins can be allocated for every I/O operation in a certain control step group. The *greater-or-equal-to* rather than *equal-to* is used because I/O pins can be allocated for an I/O operation in several control step groups, and the initial ILP tableau[1] can be made dual feasible.[2] To get linear constraints, the maximum function in Constraint 3.3 is eliminated by replacing $\max_{w \in W_v}\{x_{w,k}\}$ by $y_{v,k}$, a new binary variable, and introducing a new constraint which captures the relation

[1] A matrix form composed of the objective function and constraints. The inequalities are converted into equality form by introducing slack or surplus variables.

[2] A tableau is said to be dual feasible if all elements in the reduced cost row are nonnegative.

between $y_{v,k}$ and $x_{w,k}$. Then, Constraint 3.3 can be replaced with the following equations:

$$\sum_{v \in OS_j} B_v y_{v,k} \leq O_j, \qquad \text{for } 0 \leq j \leq N, 0 \leq k < L; \tag{3.5}$$

and

$$\sum_{w \in W_v} x_{w,k} - |W_v| y_{v,k} \leq 0, \qquad \text{for } |W_v| > 1, 0 \leq k < L; \tag{3.6}$$

where $|W_v|$ denotes the cardinal of set W_v. The maximum of $x_{w,k}$ can be expressed as Constraint 3.6, because $x_{w,k}$, and $y_{v,k}$ are binary variables. Note that $\max_{w \in W_v}\{x_{w,k}\}$ needs to be replaced by $y_{v,k}$ only if $|W_v| > 1$.

If the partition of total pins into input pins and output pins is not specified, that is, constants I_i, and O_j are not unknown in advance, introducing new integer variables o_j, substituting $I_i = T_i - o_i$ and $O_j = o_j$ into Constraints 3.2 and 3.3, and moving terms o_j to the left hand side yields

$$\sum_{w \in IS_i} B_w x_{w,k} + o_i \leq T_i, \qquad \text{for } 0 \leq i \leq N, 0 \leq k < L; \tag{3.7}$$

$$\sum_{v \in OS_j} B_v \max_{w \in W_v}\{x_{w,k}\} - o_j \leq 0, \qquad \text{for } 0 \leq j \leq N, 0 \leq k < L. \tag{3.8}$$

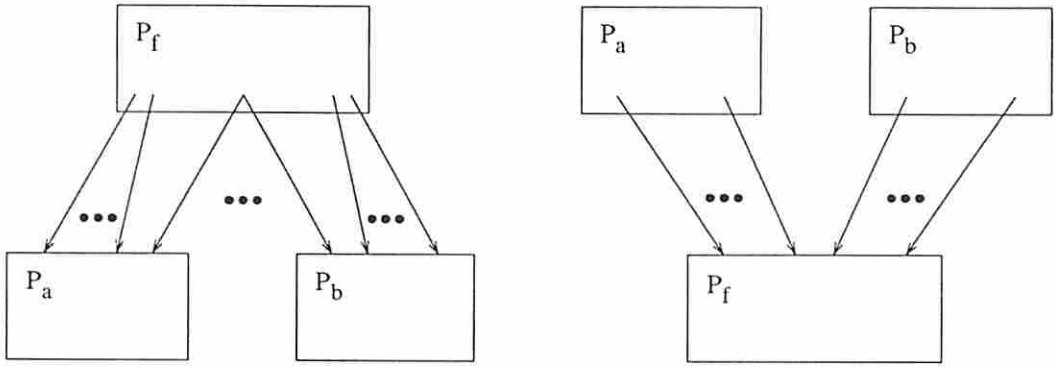
Theorem 3.1 For a simple partitioning, there exists an interchip connection configuration without communication conflict for a schedule satisfying Constraints 3.7, 3.8, and 3.4.

Proof: Since the connections at the input and output ends of a partition can be constructed independently, the communication among partitions in a simple partitioning must follow one of the two forms shown in Figure 3.2. Note that the connection between just two partitions is a special case of either one shown in the figure.

First, consider the case shown Figure 3.2(a).

Let

$$X_k = \{\text{Components of } w \,|\, w \in IS_a \wedge x_{w,k} = 1\},$$



(a) (b)

Figure 3.2: Possible communications in a *simple* partitioning

$$
\begin{aligned}
Y_k &= \{\text{Components of } w \mid w \in IS_b \wedge x_{w,k} = 1\}, \\
Z_k &= \{\text{Components of } v \mid v \in OS_f \wedge |W_v| = 2 \wedge (\forall w \in W_v, x_{w,k} = 1)\}, \\
a_k &= |X_k|, \\
b_k &= |Y_k|, \\
c_k &= |Z_k|, \\
M_a &= \max_k\{a_k\}, \\
M_b &= \max_k\{b_k\}, \\
O_i &= o_i, \text{ and} \\
I_i &= T_i - o_i.
\end{aligned}
$$

X_k, and Y_k are sets of bits of values which are to be input to partitions P_a and P_b, respectively, at control step group k. Z_k is a set of bits of values which is to be output from partition P_f to P_a and P_b at the same control step group k. a_k and b_k are the numbers of bits transferred from partition P_f to P_a and P_b, respectively, at control step group k. c_k is the number of bits transferred from partition P_f to both P_a and P_b at the same control step group k. M_a and M_b are the maximum numbers of bits transferred from partition P_f to P_a and P_b, respectively, at any

control step. Since $\{x_{w,k}\}$ is a feasible solution of Constraints 3.7, 3.8, and 3.4, we have

$$a_k \leq M_a \leq I_a, \tag{3.9}$$

$$b_k \leq M_b \leq I_b, \tag{3.10}$$

$$a_k + b_k - c_k \leq O_f. \tag{3.11}$$

The above relations state that the numbers of bits of values input and output at any control step group k must not be greater than the numbers of input and output pins, respectively. The minus term in the last inequality reflects the fact that every value output to both partitions during the same control step can share output pins.

Case (i): If $M_a + M_b \leq O_f$, that is, the number of output pins of partition P_f is greater than or equal to the sum of the maximum numbers of bits transferred from partition P_f to P_a and P_b at any control step, we can construct the interchip connection for these partitions, as shown in Figure 3.3(a), with $N_a = M_a$, $N_b = M_b$, and $N_c = 0$, where N_a, N_b, and N_c are the numbers of links in connections A, B, and C, respectively. Then, all values from P_f to partitions P_a and P_b can be transferred through connections A and B, respectively. So, no communication conflict results.

Case (ii): If $M_a + M_b > O_f$, that is, the sum of the maximum numbers of bits transferred from partition P_f to P_a and P_b at any control step is greater than the number of output pins available in partition P_f, the interchip connection shown in Figure 3.3(a) with $N_c = M_a + M_b - O_f$, $N_a = I_a - N_c$, and $N_b = I_b - N_c$ can also be used.

When $c_k \leq N_c$, we can make all bits in Z_k transferred through connection C. After c_k links of connection C are allocated to Z_k, $N_c - c_k$ links of connection C remain unused and can be used to transfer bits in X_k and/or Y_k. Only in the case of $a_k - c_k \geq N_a$, and $b_k - c_k \geq N_b$ is necessary to examine communication conflict. In this case, N_a values in X_k, and N_b values in Y_k can be transferred

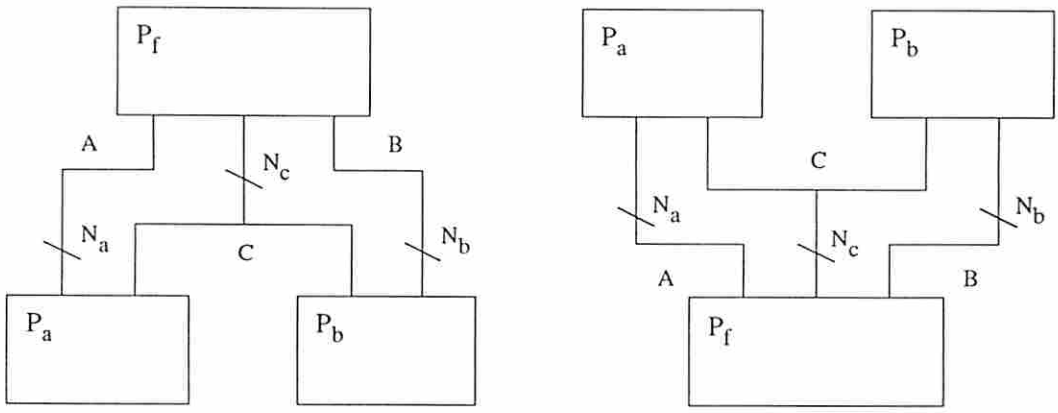


Figure 3.3: Possible interchip connection in a design of *simple* partitioning

through connections A and B, respectively. The unallocated portion of connection C is used to transfer the rest. No communication conflict occurs if the number of bits transferred does not exceed the number of links available. That is, $(a_k - c_k - N_a) + (b_k - c_k - N_b) \leq (N_c - c_k)$.

$$\begin{aligned} & (a_k - c_k - N_a) + (b_k - c_k - N_b) - (N_c - c_k) \\ = \; & a_k + b_k - c_k - (N_a + N_b + N_c) \\ = \; & a_k + b_k - c_k - (I_a + I_b - M_a - M_b + O_f) \\ = \; & (a_k + b_k - c_k - O_f) + (M_a - I_a) + (M_b - I_b) \\ \leq \; & 0. \end{aligned}$$

Since each term of the above expression is less than or equal to zero according to Constraints 3.9, 3.10, and 3.11, the entire expression is less than or equal to zero. So, no communication conflict will happen.

When $c_k > N_c$, N_c bits of Z_k are transferred through connection C, and the rest are transferred through connections A and B. It is required to check whether

there are enough connections A and B. That is, $(a_k - c_k) + (c_k - N_c) \leq N_a$, and $(b_k - c_k) + (c_k - N_c) \leq N_b$.

$$\begin{aligned}(a_k - c_k) + (c_k - N_c) - N_a &= a_k - I_a \\ &\leq 0, \\ (b_k - c_k) + (c_k - N_c) - N_b &= b_k - I_b \\ &\leq 0.\end{aligned}$$

So, no communication conflict will happen.

Now, consider the case shown in Figure 3.2(b).
Let

$$\begin{aligned}X_k &= \{\text{Components of } w \,|\, w \in OS_a \wedge x_{w,k} = 1\}, \\ Y_k &= \{\text{Components of } w \,|\, w \in OS_b \wedge x_{w,k} = 1\}, \textit{ and} \\ Z_k &= \phi.\end{aligned}$$

It can be proven in a similar way that the connection shown in Figure 3.3(b) can be used for communication without conflict. □

Note that the above formulation is only required for designs with an initiation rate greater than 1. For designs with an initiation rate of 1, the constraints for the pin allocation subproblem are reduced to

$$\sum_{w \in IS_i} B_w + \sum_{v \in OS_i} B_v \leq T_i, \qquad \text{for } 0 \leq i \leq N.$$

The feasibility of pin allocation can be guaranteed if the summation of the bit widths of all input and output values does not exceed the total number of pins available for data transfers for each partition.

3.1.2 The Size of the ILP Tableau

In the above ILP formulation, there are $|W| * L$ variables of $x_{w,k}$, and $N + 1$ variables of o_j. The numbers of constraints in Constraints 3.7, 3.8, and 3.4 are $(N+1)*L$, $(N+1)*L$, and $|W|$, respectively. For the best case, in which there is no value going to more than one partition, the number of variables and constraints are $|W| * L+(\mathrm{N}+1)$ and $2(N+1)*L+|W|$, respectively. For the worst case, in which all of values are going to exactly two partitions, there are extra $|W|*L/2$ variables of $y_{v,k}$, and $|W|*L/2$ constraints in Constraint 3.6. Summing up results in $|W|*3L/2+(N+1)$ variables, and $2(N+1)*L+|W|(1+L/2)$ constraints. However, the size of the ILP formulation can be reduced in the following way. Let $w_1, w_2, \ldots, w_q$ be single-fanout I/O operations transferring values that have same bit width from partition i to j. Define $x_{w_1,k} + x_{w_2,k} + \cdots + x_{w_q,k}$ as $x_{w,k}$, and substitute into Constraints 3.7 and 3.8. In addition, the inequalities in Constraint 3.4 which correspond to I/O operations $w_1, w_2, \ldots, w_q$ are replaced by

$$\sum_{k=0}^{L-1} x_{w,k} \geq q.$$

In practice, most of the values have the same bit width. So, the tableau size can be reduced quite a lot.

3.2 Scheduling under Resource Constraints

In the scheduling approach described in this section, all partitions of the system are processed at the same time, since there is strong interdependence among partitions. Scheduling an output or input operation of a partition will also restrict the schedule of other input or output operation(s), and so allocation of input or output pins of other partition(s). In addition, scheduling a partition can affect the quality of the schedule of other partitions, and so the whole system.

List scheduling is used in our prototype program to schedule the system under resource constraints, in term of the numbers of modules for each type. I/O

operations, each of which requires an input operation of a partition and an output operation of another partition to occur in the same control step, can cause a problem when scheduling a design consisting of multiple partitions, each having pin constraints. Scheduling can be impossible to complete if I/O operation nodes are scheduled inappropriately, as discussed in Section 2.4.

A solution to avoid these situations is to determine whether there still exists a valid pin allocation if I/O pins are allocated for the I/O operation in a certain control step before each I/O operation is actually scheduled. If allocating I/O pins for (or scheduling) an I/O operation in a certain control step results in no feasible pin allocation, the I/O operation will be postponed to a later control step. The procedure for scheduling under resource constraints is shown in Figure 3.4.

A pin allocation feasibility checker has been incorporated into the scheduling procedure to determine the feasibility of pin allocation before scheduling an I/O operation in a control step, as shown by the bold boxes in the flow chart.

3.3 Determining Feasibility of an ILP

Certain ILPs, such as max-flow and weighted bipartite matching problems, can be solved as linear programs with the Simplex algorithm. In such problems, the optimal solution to the corresponding LP is guaranteed to be integer. For most ILPs, however, the solution obtained usually does not satisfy the integral constraint imposed by ILP. Unfortunately, the problem we have here does not exhibit the integral property. General algorithms for ILP fall into two categories [PS82]: *cutting plane algorithms*, which are derived from the simplex algorithm, and *enumerative algorithms*, which are based on enumerating, either explicitly or implicitly, all possible solutions.

The main idea of the cutting plane algorithm is to add inequalities or "cuts" that do not exclude integer feasible points to the corresponding LP until the solution to the LP is integer.

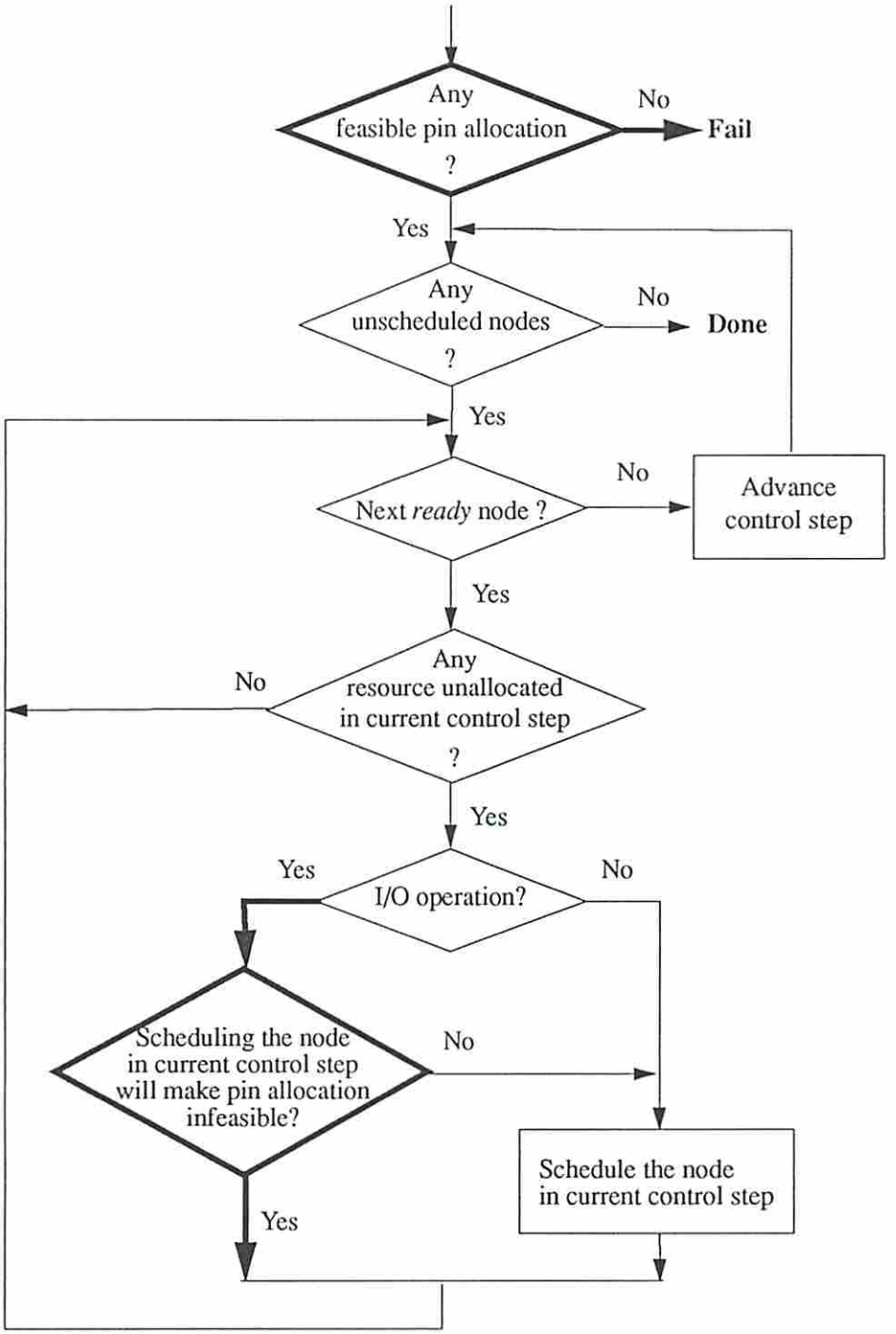


Figure 3.4: A *list scheduling* algorithm with pin allocation feasibility checker

The cutting plane algorithm with *Dual All-Integer Cuts* was developed by Ralph Gomory [Gomo60] in 1960. In the algorithm, the initial tableau is assumed to be all-integer and dual feasible. At each iteration of the dual simplex algorithm, an all-integer inequality is generated and used as the pivot row and ensures a -1 pivot. So, the tableau remains all-integer after pivoting.

After the initial tableau is constructed, the Dual All-Integer Cutting Plane algorithm is called. Infeasibility of the initial formulation means that there is no feasible schedule under the pin constraints.

This technique can be applied to our problem. Every time before scheduling an I/O operation node w in a control step s, constraint $x_{w,k} \geq 1$, where $k = s \bmod L$, is added to the ILP and the cutting plane algorithm is called to check whether it is feasible. The initial tableau of the new ILP is obtained by updating the result tableau of the previous step. That is, the solution obtained in the previous step is used as the start point for the current step. The cutting plane algorithm usually terminates in a few iterations since the feasible solution space defined by the constraints is modified only slightly.

The initial tableau of the updated tableau can be obtained in the following ways. Let

$$\left(A_1 \cdots A_j \cdots A_m \right) \begin{pmatrix} z_1 \\ \vdots \\ z_j \\ \vdots \\ z_m \end{pmatrix} = b, \tag{3.12}$$

where A_j, and b are column vectors and z_j is a single variable,[3] be the result tableau of the previous step. Let z_j, and z'_j denote $x_{w,k}$, and $x'_{w,k}$, respectively. Constraint

[3] z_j rather than $x_{w,k}$ is used since z_j can be $x_{w,k}$, or $x_{w,k} - 1$.

$x_{w,k} \geq 1$ can be added into the tableau simply by letting $x'_{w,k} = x_{w,k} - 1 \geq 0$. Substituting $x_{w,k} = x'_{w,k} + 1$ into Equation 3.12 and rearranging it yields

$$\left(A_1 \cdots A_j \cdots A_m \right) \begin{pmatrix} z_1 \\ \vdots \\ z'_j \\ \vdots \\ z_m \end{pmatrix} = b' = b - A_j. \tag{3.13}$$

The modified tableau with the additional constraint can be obtained simply by subtracting the *j'th* column, A_j, of the original matrix from the constant vector, b.

3.4 Experimental Results

The AR Lattice Filter [Kun84] is used as a test example. It is to be implemented on 4 chips. Two chips have 48 pins used for data transfer, while the other two have 32 pins. A simple partitioning of the AR filter into 4 blocks is shown in Figure 3.5, in which I/O operation nodes, indicated as shaded nodes, have been inserted on the arcs across partition boundaries. All I/O operations are 8 bits wide. In addition, the following assumptions are stated:

- The stage time is 250 ns.
- Inputs arrive every 2 cycles.
- I/O operations take 10 ns.
- Additions are performed by adders with 30 ns. delay.
- Multiplications are performed by multipliers with 210 ns. delay.
- Minimum number of functional units are used.
- Chained operations are allowed.

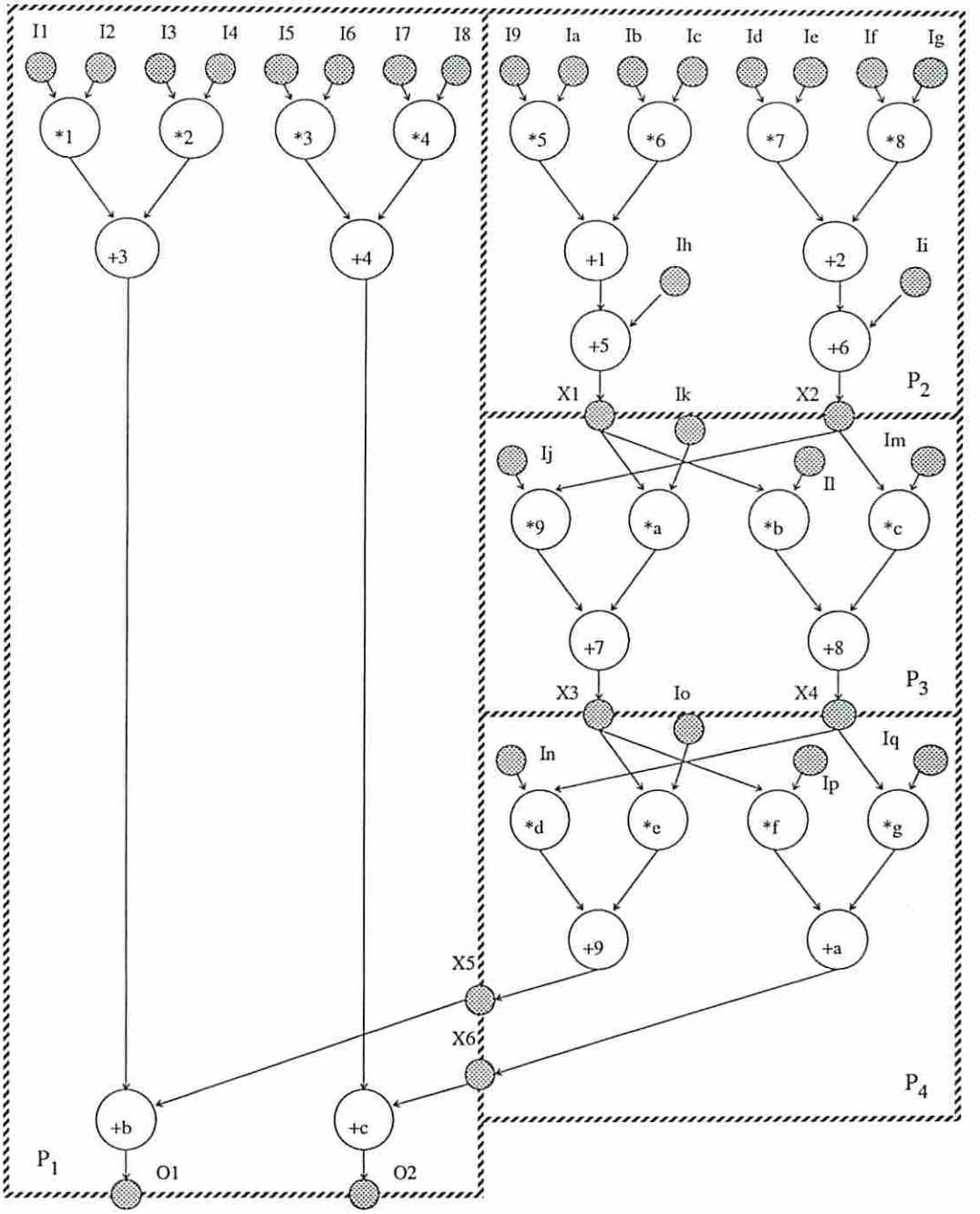


Figure 3.5: A simple-partition AR filter

The minimum functional operators required for partitions P_1, P_2, P_3, and P_4 are (2+,2*), (2+,2*), (1+,2*), and (1+,2*), respectively, under the given initiation rate. With these resource constraints, our program scheduled the partitioned AR filter as shown in Figure 3.6 by using *list scheduling* incorporated with the safety check mechanism. It takes 0.5 sec. on Sun 3/280.

Each of the partitions P_1 and P_2 has 10 input operations, and 2 output operations, and each of the partitions P_3 and P_4 has 6 input operations, and 2 output operations. With the initiation rate of 2, the 48 pins have to be grouped into 6 bundles of 8 bits, 5 for input and 1 for output, and the 32 pins have to be grouped into 4 bundles of 8 bits, 3 for input and 1 for output. Note that I/O operations I1, I2, I3, and I4 were assigned in control step 1 even though there was still one group of input pins unallocated after I/O operations I5, I6, I7, and I8 were assigned in control step 0. The safety checker in the program predicted that the schedule could not be completed if any of I/O operations I1, I2, I3, and I4 was assigned to control step 0 after I/O operations I5, I6, I7, and I8 were scheduled. One of I/O operations X5, and X6 must be assigned in control step group 0, and the other in control step group 1 because partition P_4 has only 1 output pin. At least one group of pins should be reserved for I/O operations X5, and X6. I/O operations Ij and Ik of partition P_3, and In and Io of partition P_4 were postponed to control step 1 for the same reason.

The interchip connection for the schedule, which is obtained by following the procedure in the proof of Theorem 3.1, is shown in Figure 3.7.

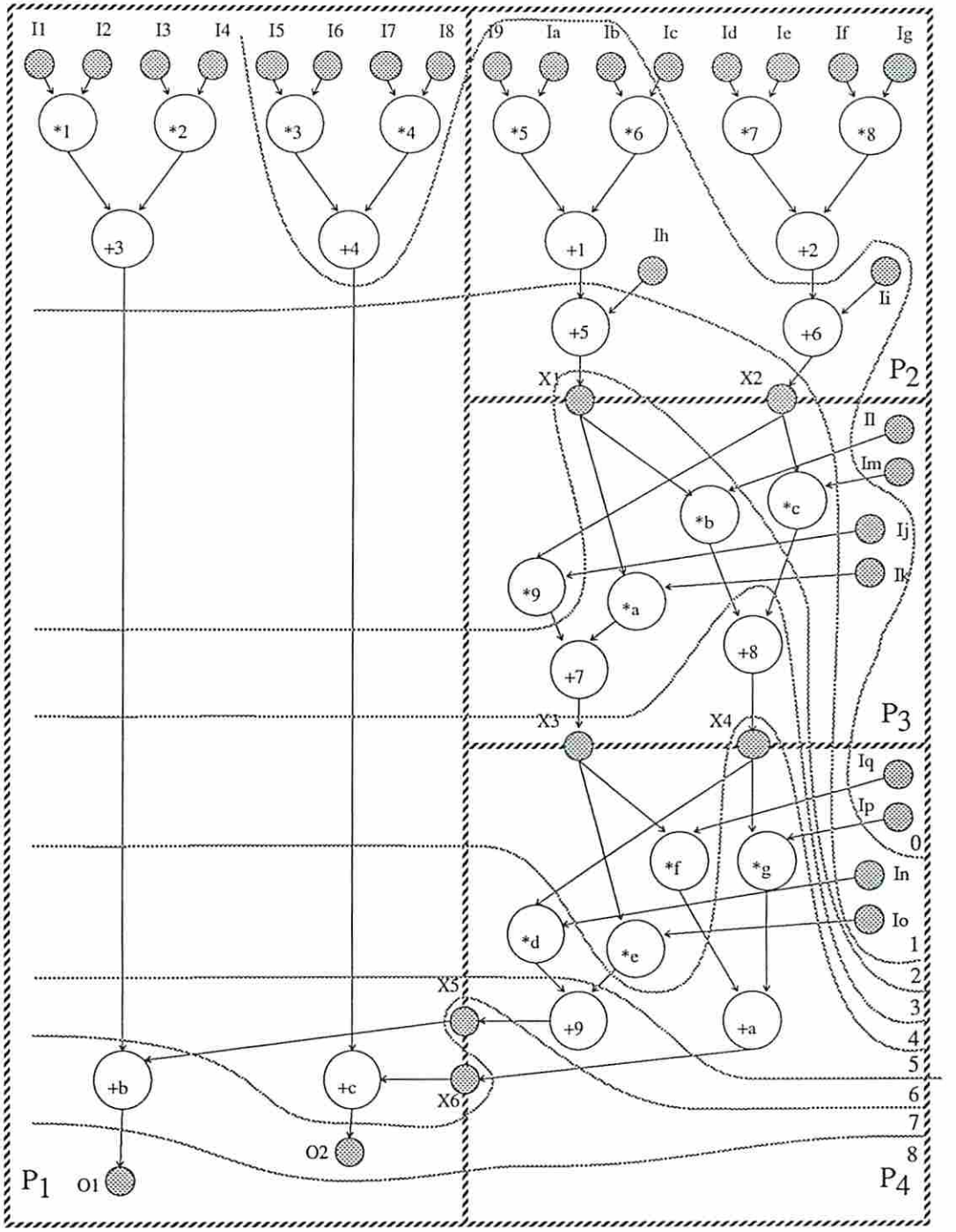


Figure 3.6: Scheduling of the simple-partition AR filter

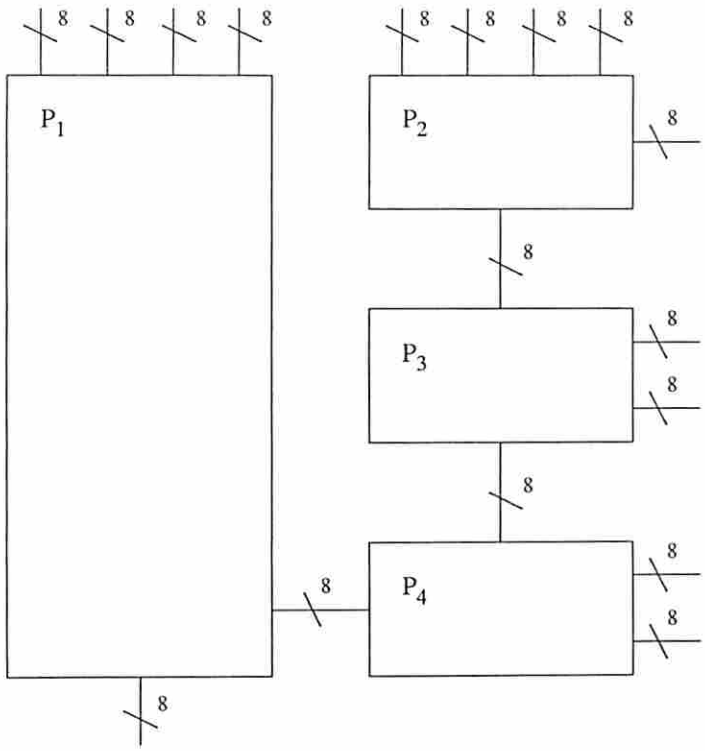


Figure 3.7: Interchip connection for the partitioned AR filter in Figure 3.5

Chapter 4

Synthesis for Designs with a General Partitioning

In Section 2.4, we discussed the fact that assigning an I/O operation in a control step in which there are unallocated input and output pins in the corresponding partitions does not guarantee that communication conflicts will not occur, because no switching devices are allowed outside partitions. For designs with simple partitioning, it is guaranteed, as discussed in Chapter 3, that there will be no communication conflict if an I/O operation is assigned in a control step in which there are unallocated input and output pins in the corresponding partitions. The technique discussed in the previous chapter can be used to predict the feasibility of a schedule with simple partitioning before an I/O operation is scheduled. For general partitioning, however, we have to develop new methods. In this chapter, a technique to produce designs which have a general partition structure is described. Here, the problem has been simplified so that every communication bus can only be used to transfer at most one value in one control step. That is, values with smaller bit width are not grouped and transferred on a communication bus with a larger bit width. Also, values have to be transferred as a whole. In other words, a communication bus does not carry a portion of a value.

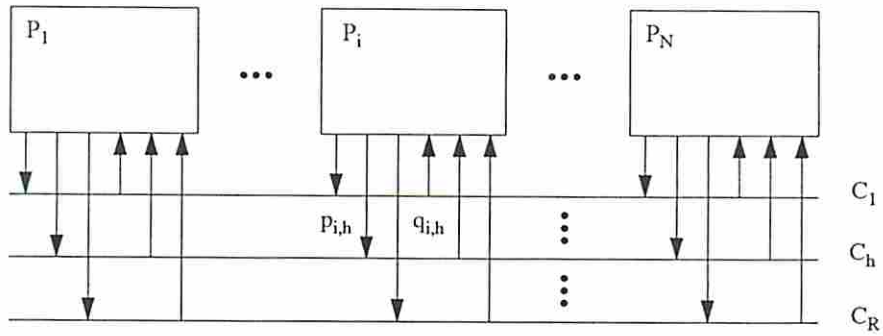


Figure 4.1: Interchip connection model

Due to its complexity, the problem will be approached in two phases,

1. determining the interchip connections before scheduling, and
2. scheduling the design given the interchip connections.

4.1 Determining Interchip Connections

The interchip connection model has the bus structure shown in Figure 4.1. In the figure, $p_{i,h}$ ($q_{i,h}$) is the width of the output (input) ports of partition P_i connected to bus C_h. In our model, each bus can be connected to more than one partition at its input port, and more than one partition at its output port. A partition can be connected to more than one bus. However, an input or output port can be connected to one and only one bus. The width of the input or output port of a partition connected to a bus is not necessarily the same as the width of the input or output port of other partitions connected to the bus. For example, the interchip connection shown in Figure 4.2 can be used to transfer values with less than or equal to 16 bits from partition P_a to partition P_c, and values with no more than 12 bits from partition P_b to partition P_c. Only 12 instead of 16 output pins of partition P_b are connected to the bus if only values with bit width less than or equal to 12 are to be transferred from partition P_b to partition P_c through the bus.

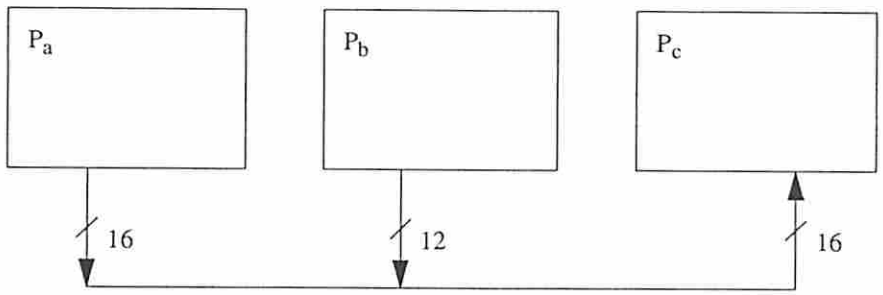


Figure 4.2: An interchip connection example

4.1.1 The ILP Formulation

The communication among partitions can be denoted by a directed multigraph[1] $G = (P, W)$, where P is a set of nodes,

$$P = \{P_0, P_1, \ldots, P_N\},$$

and W is a set of edges,

$$W = \{w_1, w_2, \ldots, w_q\}.$$

Each node represents a partition, and each edge $w_k = (P_i, P_j)$ represents an I/O transfer (operation) from partition P_i to partition P_j.

The following notation will be used in the discussion.

- L : Initiation rate of the design.
- N : Number of partitions.
- R : Maximum number of communication buses.
- B_w : Bit width of I/O operation w.
- T_i : Total number of I/O pins used for data transfers (pins used for power and control lines excluded) in partition P_i.

[1]In a multigraph, there can be several edges between a pair of nodes.

- $C = \{C_1, C_2, \ldots, C_r\}$: A set of communication buses.
  A communication bus is connected to one or more partitions at their output ports and one or more partitions at their input ports. A communication bus is capable of performing I/O transfer $w = (P_i, P_j)$ if it connected to partition P_i at its output port and partition P_j at its input port, and the widths of both ports of the partitions connected to the bus are at least B_w lines.

- $OW_i = \{w \mid w = (P_i, P) \wedge P \text{ is any partition}\}$.

- $IW_i = \{w \mid w = (P, P_i) \wedge P \text{ is any partition}\}$.

- $W_v = \{\ w \mid \text{every I/O operation } w \text{ used to transfer value } v\ \}$.

- o_i : An integer variable specifying the number of pins to be used for output from partition P_i.

- $y_{w,h}$: A binary variable denoting I/O operation w is assigned to communication bus C_h.
  $y_{w,h} = 1$ if I/O operation w is assigned to communication bus C_h;
  $y_{w,h} = 0$ otherwise.

- $p_{i,h}$: An integer variable denoting the width of the output port of partition P_i connected to communication bus C_h.
  $p_{i,h} = 0$ if C_h is not connected to the output of P_i.

- $q_{i,h}$: An integer variable denoting the width of the input port of partition P_i connected to communication bus C_h.
  $q_{i,h} = 0$ if C_h is not connected to the input of P_i.

The problem of determining interchip connection can be formulated as an objective function subject to a set of constraints.

The *assignment* constraint

$$\sum_{h=1}^{R} y_{w,h} = 1, \qquad \text{for } w \in W; \tag{4.1}$$

states that every I/O operation must be assigned to one and only one communication bus.

The *data transfer* constraints

$$p_{i,h} \geq \max_{w \in OW_i} B_w y_{w,h}, \quad \text{for } 0 \leq i \leq N, 1 \leq h \leq R; \tag{4.2}$$

$$q_{i,h} \geq \max_{w \in IW_i} B_w y_{w,h}, \quad \text{for } 0 \leq i \leq N, 1 \leq h \leq R; \tag{4.3}$$

specify that the width of the output (input) port of partition P_i connected to communication bus C_h must be greater than or equal to the bit widths of values output from (input to) partition P_i via the bus.

The *resource* constraints

$$\sum_{h=1}^{R} p_{i,h} + \sum_{h=1}^{R} q_{i,h} \leq T_i, \qquad \text{for } 0 \leq i \leq N \tag{4.4}$$

ensure that the total number of input and output pins connected to any bus does not exceed the total number of pins available.

For each communication bus, only one value can be transferred in a control step because communication buses are connected components. So, for a pipelined design with an initiation rate of L, each communication bus can be used to transfer at most L values. This is specified by *capacity* constraints

$$\sum_{v \in V} \max_{w \in W_v} y_{w,h} \leq L, \qquad \text{for } 1 \leq h \leq R. \tag{4.5}$$

The heuristic objective function

$$\text{Maximize} \sum_{h=1}^{R} \max_{w \in W} y_{w,h} \tag{4.6}$$

tries to maximize the number of buses actually used since a higher I/O bandwidth is more likely to result.

The maximum function of binary variables in Constraint 4.5 can be eliminated by introducing a new binary variable as discussed in Section 3.1. The maximum

function of integer variables in Constraints 4.2 and 4.3 can be eliminated in the following way. The inequality

$$p_{i,h} \geq \max_{w \in OW_i} B_w y_{w,h}$$

can be replaced by

$$p_{i,h} \geq B_w y_{w,h} \qquad \text{for } w \in OW_i.$$

In the above formulation, there are approximately $(|W| + 2N + 2)R$ variables, and $(2R+1)|W|+(N+1)+R$ constraints. The variables are composed of $|W| * R$ variables of $y_{i,h}$, $(N+1) * R$ variables of $p_{i,h}$, and $(N+1) * R$ variables of $q_{i,h}$. Constraints 4.1, 4.2, 4.3, 4.4, and 4.5 have $|W|$, $|W| * R$, $|W| * R$, $N+1$, and R constraints, respectively. Linearizing $\max_{w \in W_v} y_{w,h}$ in Constraint 4.5 will introduce extra binary variables and constraints.

The above formulation requires a maximum number of communication buses, R, to be specified. A naive way of determining the upper bound of R is to count the total number of I/O operations. However, a tighter upper bound can be obtained. The concept is based on the observation that every communication bus must be connected to at least one input port and one output port, and no input port or output port can be connected to more than one communication bus. For each chip, the maximum number of input (output) ports is estimated according to the number of I/O pins available for input (output), and the number and bit widths of input (output) operations.

Let

- $B_1 < B_2 < \ldots < B_T$: A increasing sequence of bit widths.
- T_i : Total number of I/O pins used for data transfers in partition P_i.
- $nI_{i,k}$: Number of input operations in P_i whose bit width is equal to B_k.
- $Iub_{i,k}$: Upper bound on the number of input ports of P_i having a width of B_k.

- $Ilb_{i,k}$: Lower bound on the number of input ports of P_i having a width of B_k assuming that a minimum number of input ports of P_i are used for input operations with widths greater than B_k.

- $IP_{i,k}$: Upper bound on the number of input pins left unallocated in P_i after pin allocation for input values with bit width greater than B_k.

- $IS_{i,k}$: Number of input slots[2] in partition P_i available for input values with bit width less than or equal to B_k when a minimum number of input pins are used for input values with bit width greater than B_k. ($IS_{i,T} = 0$.)

- IPl_i : Lower bound on the number of input pins of P_i required for all input operations in P_i.

- OPl_i : Lower bound on the number of output pins of P_i required for all output operations in P_i.

Before the upper bounds on the numbers of input ports and output ports are computed, the lower bounds on the numbers of input pins and output pins are determined because the values produced during the computation of the lower bounds on the numbers of input pins and output pins are required to compute the upper bounds on the numbers of input ports and output ports.

The lower bound on the number of input ports of P_i with a width of B_k assuming that a minimum number of input ports of P_i are used for input operations with widths greater than B_k, $Ilb_{i,k}$, can also be given as

$$Ilb_{i,k} = \lceil \frac{nI_{i,k} - IS_{i,k}}{L} \rceil.$$

The unallocated input slots due to not fully utilizing input ports with bit widths greater than B_k can be used to input values with B_k or less bits. This is reflected by the minus term in the numerator.

[2]For an initiation rate of L, an input port can be used to input L values, each value at a control step. We say there are L input slots for each input port.

The number of input slots in partition P_i available for input values with bit widths less than or equal to B_{k-1} when a minimum number of input pins is used for input values with bit widths greater than B_{k-1} can be obtained by adding the number of additional slots of an input port with a width of B_k required to the number of those already available for input values with bit widths less than or equal to B_k and subtracting those actually used for input operations with a width of B_k. That is,

$$IS_{i,k-1} = IS_{i,k} + Ilb_{i,k} * L - nI_{i,k}.$$

So, a lower bound on the number of input pins of P_i, IPl_i, can be computed by

$$IPl_i = \sum_{k=1}^{T} Ilb_{i,k} * B_k.$$

A lower bound on the number of output pins of P_i, OPl_i, can be computed in a similar way.

For each partition P_i, the maximum number of I/O pins available for input can be obtained by subtracting the minimum number of I/O pins required for output from the total number of I/O pins available for data transfers. That is,

$$IP_{i,T} = T_i - OPl_i.$$

The upper bound on the number of input ports of P_i having a width of B_k, $Iub_{i,k}$, is given as

$$Iub_{i,k} = \min(\lfloor \frac{IP_{i,k}}{B_k} \rfloor, nI_{i,k}).$$

The first term gives the maximum number of input ports having a width of B_k which can be formed from the remaining input pins. However, the upper bound should not exceed the number of input operations with B_k bits, $nI_{i,k}$.

The maximum number of input pins available for input values with bit widths less than or equal to B_{k-1} can be obtained by subtracting the minimum number of input pins allocated for input values with B_k bits from the maximum number

of those available for input values with bit widths less than or equal to B_k. That is,

$$IP_{i,k-1} = IP_{i,k} - Ilb_{i,k} * B_k.$$

So, an upper bound on the number of input ports of P_i, uI_i, can be computed by summing up all upper bound numbers of input ports with different bit widths. That is,

$$Iub_i = \sum_{k=1}^{T} Iub_{i,k}.$$

An upper bound on the number of output ports of P_i, Oub_i, can be computed in a similar way. So, a upper bound on the number of communication buses, called R, which is tighter than that obtained by counting the total number of I/O operations, can be given by

$$R = \min(\sum_{i=0}^{N} Iub_i, \sum_{i=0}^{N} Oub_i).$$

The reason for taking the minimum is that every communication bus must be connected to at least one input port and one output port.

4.1.2 A Heuristic Search Procedure

The above formulation can be submitted to an ILP package to get a solution, and we have implemented a program to automatically generate the ILP formulation, and have submitted the formulation to two ILP packages, *Bozo* [HH90] and *LINDO* [LIN87]. However, for practical size problems, the size of the formulation is too large to obtain a solution within a reasonable time. Still, the ILP formulation is useful for verification of synthesized results. So, a heuristic search technique has been developed as shown in Figure 4.3.

At the beginning, the I/O operations are sorted. An I/O operation with a higher bit width requires more I/O pins, and is assigned before the others. It is believed that the search tree can more likely be pruned at an earlier stage. At each

```
1:  determine_interchip_connection
2:  build a sorted list of I/O operations L in descending order of bit width
3:  C = assign_nodes ( L )

4:  assign_nodes ( L )
5:  if ( L = φ ) then return φ
6:  w = first element of L
7:  L' = L - w
8:  select a small number of communication buses for w having best gain
9:  for ( each communication bus C_h ) do
10:     if ( there still are I/O pins available to assign w to C_h ) then
11:         assign w to C_h tentatively and update connection table
12:         C' = assign_nodes ( L' )
13:         if ( C' ≠ FAIL ) then return C_h + C'
14:         restore connection table
15:     endif
16: endfor
17: return FAIL
```

Figure 4.3: A heuristic search procedure to find an interchip connection structure

level of the search tree, only a small number of communication buses having the best gain will be explored. For the sake of run time efficiency, the communication buses to be explored do not have the same *topology*. Two communication buses are said to have same topology if they are connected to same partitions even though they can have different bit widths. The number of branches to be explored at each node is controlled by an branching factor, which is given by users to trade off the execution time and the chance of finding a solution. The search time is still exponential in term of the number of I/O operations in the worst case if the branching factor is greater than 1.

The gain, g, of assigning I/O operation $w = (P_i, P_j)$ to communication bus C_h is composed of three factors g_1, g_2, and g_3 shown as follows:

$$g = 10000 * g_1 + 100 * g_2 + g_3.$$

The above weighting constants are used to order factors g_1, g_2 and g_3 according to their priorities. The values of 10000, 100, and 1 are chosen arbitrarily.

Factor g_1 is used to make the I/O operation try to utilize an already existing communication path in order to save pin resources, and is given as

$$g_1 = \begin{cases} 0 & \text{if } p_{i,h} = 0 \wedge q_{j,h} = 0 \\ wf_i & \text{if } p_{i,h} \neq 0 \wedge q_{j,h} = 0 \\ wf_j & \text{if } p_{i,h} = 0 \wedge q_{j,h} \neq 0 \\ wf_i + wf_j & \text{otherwise} \end{cases} \tag{4.7}$$

where

$$wf_i = \frac{\text{number of bits of unassigned I/O operations in } P_i}{\text{number of unallocated pins in } P_i}$$

Priority is given to the partitions with tight pin resources. This is reflected by the weighting factor wf_i.

The factor g_2 given below is used to force the I/O operations transferring the same value to be assigned to a common communication bus. So, no extra communication slot is consumed.

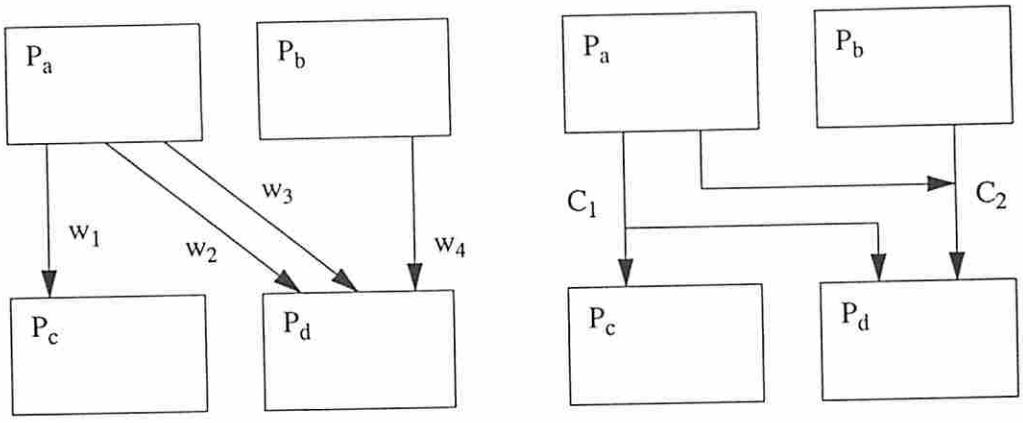
$$g_2 = \begin{cases} 1 & \text{if } w \text{ and a } w' \text{ assigned to } C_h \text{ transfer same value} \\ 0 & \text{otherwise} \end{cases}$$

Factor g_3 tries to balance the utilization of communication buses, and is given as follows:

$$g_3 = \text{number of unassigned slots in } C_h$$

4.2 Scheduling with Given Interchip Connections

After the interchip connection has been determined, data path and I/O scheduling using this interconnection can be performed. A list scheduling technique is used. Before scheduling each I/O operation w in control step s, any communication bus available in the control step for this I/O operation is checked. Recall that every I/O operation is also assigned to a communication bus in the procedure of determining interchip connection. A direct way of checking availability of communication buses for the I/O operation is to check whether the communication bus previously assigned to the I/O operation is free in the control step. However, the resulting schedule using static assignment can be too pessimistic. For the simple example shown in Figure 4.4(a), the interchip connection can be constructed as shown in Figure 4.4(b). Assume that initially I/O operations w_1 and w_2 are assigned to communication bus C_1, and I/O operations w_3 and w_4 are assigned to communication bus C_2. After w_1 is scheduled in control step s, w_2 cannot be scheduled in the same control step since C_1 has been allocated for w_1. However, w_2 can actually be scheduled in the same control step by using C_2. In this case, w_3 can be reassigned to C_1. When assigning w_2 to C_2, it must be ensured that w_3



(a) Communication between partitions (b) Inter-chip connection

Figure 4.4: An example of an interchip connection

or w_4 can be reassigned to another communication bus. Therefore, it is of benefit to reassign communication buses dynamically as the scheduling proceeds.

Reassignment can be done in the following way. If the communication bus currently assigned to the I/O operation being checked has been allocated to another I/O operation in the current control step group, the I/O operation tries to preempt another I/O operation which is currently assigned to a communication bus that is unallocated in the current control step group. The preempted I/O operation, in turn, tries to preempt another I/O operation until a preempted I/O operation can find a *free*[3] communication slot, and be assigned to it. Before the preemption process starts, the communication slot currently assigned to the I/O operation being checked is released.

The process of checking any free communication bus to assign an I/O operation at a control step and reassigning I/O operations to communication buses if necessary is similar to the one of finding an *augmentation path* in a *bipartite matching problem*, which can be solved in polynomial time [PS82]. A bipartite

[3]A *free* communication slot is a communication slot not assigned to any I/O operation

graph $G = (V, U, E)$ is a graph in which every edge in E has one terminal in V, and the other terminal in U. A matching of a graph is a subset of the edges with the property that no two edges in the subset share the same node. The matching problem is to find a maximum matching of a graph. Given a graph and a matching, *matched* (*free*) edges are edges (not) in the matching. *Exposed* nodes are not incident upon any matched edge. An *alternating* path is a path $p = [v_1, v_2, \ldots, v_k]$, where $[v_1, v_2], [v_3, v_4], \ldots$ are free, and $[v_2, v_3], [v_4, v_5], \ldots$ are matched. An augmentation path is an alternating path $p = [v_1, v_2, \ldots, v_k]$, where v_1, and v_k are exposed.

The problem of reassigning I/O operations to communication buses can be represented by a bipartite graph $G = (V, U, E)$, in which V is a set of nodes representing I/O operations, U is a set of nodes representing communication slots, which are 2-tuples (C_h, s_k), where C_h is a communication bus, and s_k is a control step group, and E is a set of edges representing the capability of a communication bus able to be used to perform an I/O operation. The reassignment of I/O operations to communication slots within a communication bus can be done by simply switching the assignment. So, the communication slots within a communication bus are grouped to eliminate unnecessary search.

The reassignment of I/O operations for the example is shown in Figure 4.5. A thin line indicates that the communication bus is capable of performing the I/O operation. A heavy line indicates that the I/O operation is currently assigned to the communication bus. A shaded line indicates that the assignment has been fixed because the communication bus has been allocated for an I/O operation at a control step. Circles in ellipses show the status of communication buses at certain control steps. An empty one indicates that the bus is free at the control step. The dotted line in Figure 4.5(b) corresponds to an augmentation path in the matching problem.

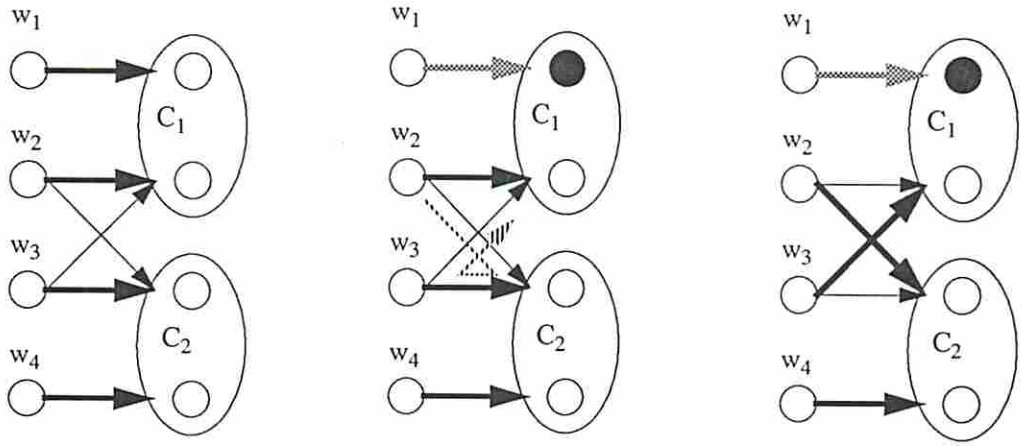


Figure 4.5: Snapshots of communication bus allocation

4.3 Bidirectional I/O Ports

In the previous section, we have assumed that all I/O ports are unidirectional. That is, an I/O port is either an input port or output port, but not both. However, I/O pins can be utilized more efficiently if bidirectional I/O ports are used.

Interchip connection with bidirectional I/O ports can be modeled as shown in Figure 4.6. The ILP formulation for the model is similar to the formulation for the model with unidirectional I/O ports except that data transfer constraints 4.2 and 4.3 are replaced by

$$r_{i,h} \geq \max_{w \in IW_i \cup OW_i} B_w y_{w,h}, \quad \text{for } 0 \leq i \leq N, 1 \leq h \leq R$$

where $r_{i,h}$ is an integer variable denoting the width of the I/O port of partition P_i connected to communication bus C_h.

Resource constraint 4.4 is replaced by

$$\sum_{h=1}^{R} r_{i,h} \leq T_i, \quad \text{for } 0 \leq i \leq N.$$

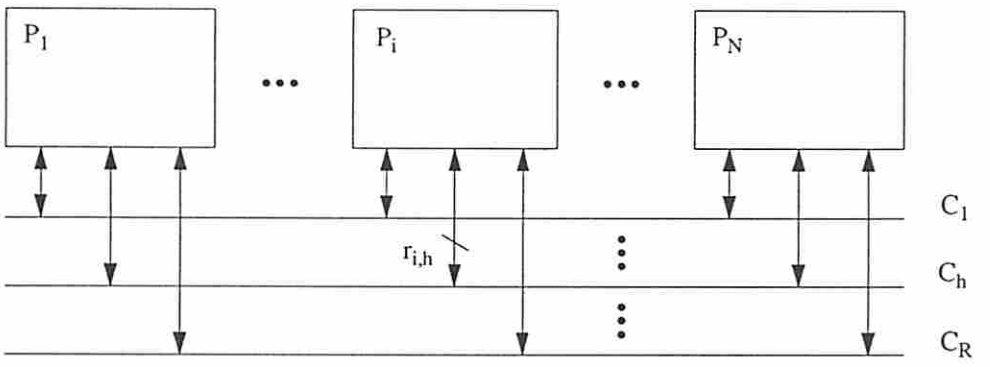


Figure 4.6: Interchip connection model with bidirectional I/O ports

The upper bound of communication buses can be estimated similarly to the procedure described in the previous section. However, the maximum number of communication buses is half of total number of I/O ports since every bus must have at least two I/O ports connected to it.

The heuristic search technique developed for designs with unidirectional I/O ports can be applied to designs with bidirectional I/O ports, except that Equation 4.7 for factor g_1 is replaced by

$$g_1 = \begin{cases} 0 & \text{if } r_{i,h} = 0 \wedge r_{j,h} = 0 \\ wf_i & \text{if } r_{i,h} \neq 0 \wedge r_{j,h} = 0 \\ wf_j & \text{if } r_{i,h} = 0 \wedge r_{j,h} \neq 0 \\ wf_i + wf_j & \text{otherwise} \end{cases}$$

4.4 Experimental Results

It would be difficult to directly compare the work to De Micheli's work [GM90] because they only consider nonpipelined designs.

Initiation Rate	P_0	P_1			P_2			P_3		
3	120P	135P	2+	2*	90P	2+	2*	90P	2+	2*
4	120P	135P	1+	1*	90P	1+	2*	90P	1+	2*
5	120P	135P	1+	1*	90P	1+	2*	90P	1+	2*

P: Pins, +: Adders, *: Multipliers

Table 4.1: Resource Constraints for the AR filter designs with unidirectional I/O ports

4.4.1 AR Lattice Filter

The AR Lattice Filter is partitioned in a general-partition structure as shown in Figure 4.7, and a variety of bit widths are assumed. The number next to the I/O operation is the bit width. The I/O operations without numbers next to them have 8 bits. Other assumptions are stated as follows:

- The stage time is 250 ns.
- I/O operations take 10 ns.
- Additions are performed by adders with 30 ns delay.
- Multiplications are performed by multipliers with 210 ns delay.
- Chained operations are allowed.

4.4.1.1 Designs with Unidirectional I/O Ports

The partitioned AR filter has been synthesized with initiation rates of 3, 4, and 5. The hardware resources available for designs with different initiation rates are shown in Table 4.1.

First, the interchip connections are synthesized by using the heuristic search technique described in this chapter. Interchip connections for designs with different initiation rates are shown in Figures 4.8, 4.9, and 4.10. The actual numbers of I/O pins allocated for the interchip connections are given in the second column of

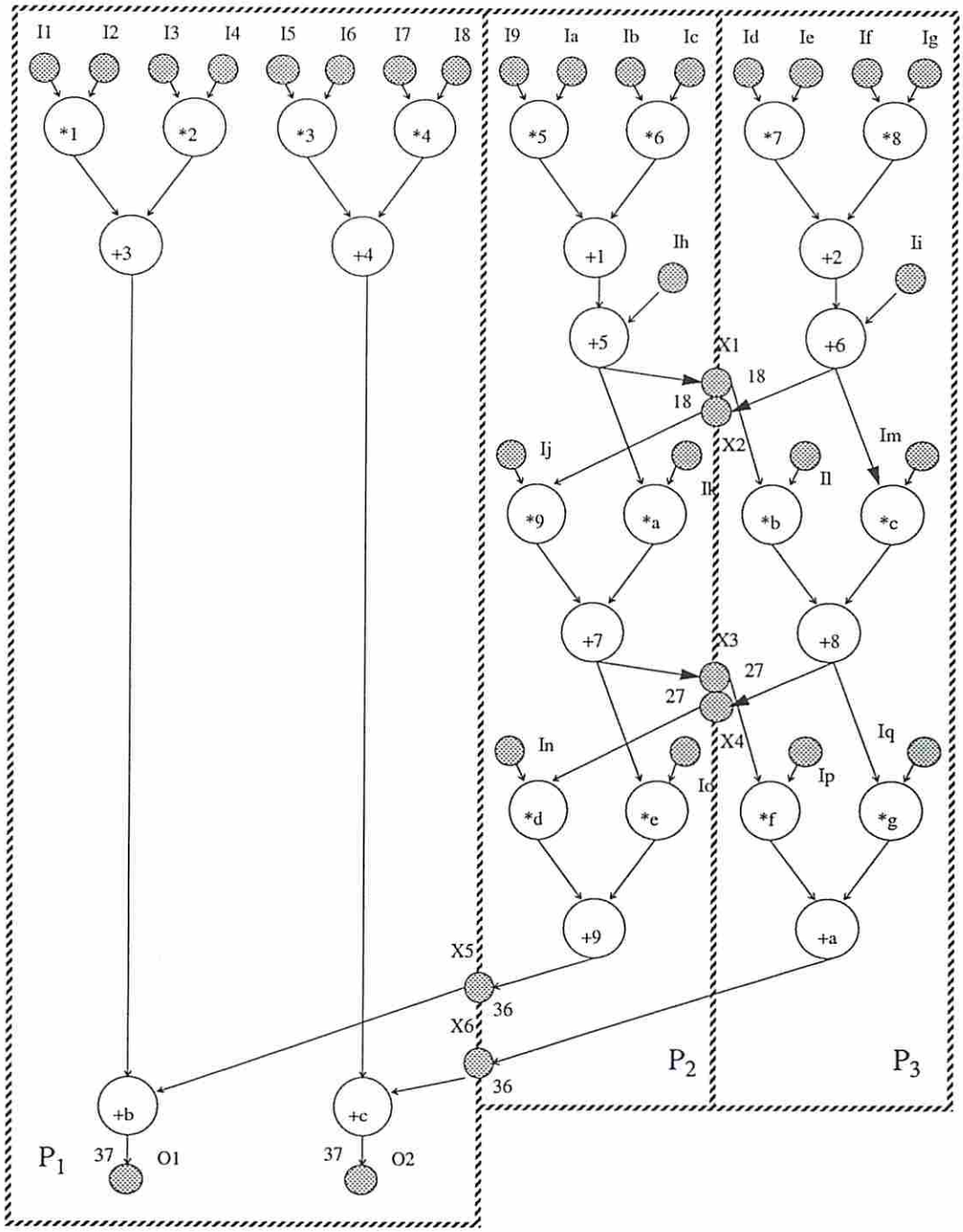


Figure 4.7: A general-partition AR filter

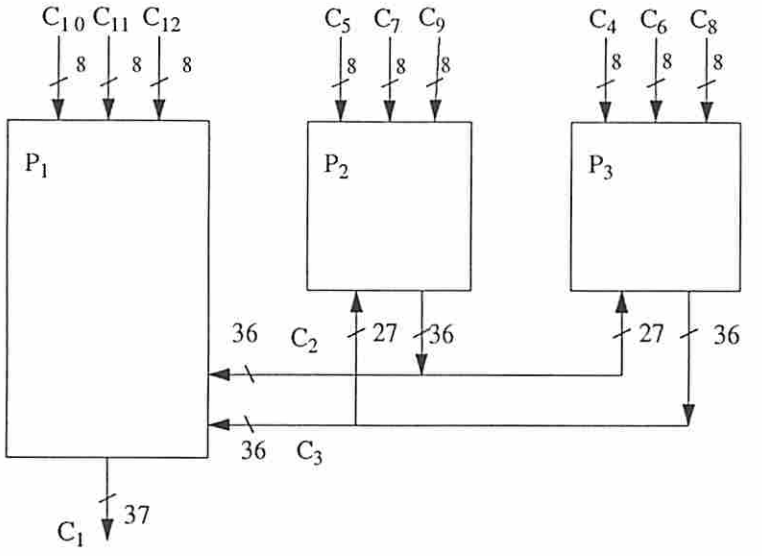


Figure 4.8: Interchip connection for the AR filter design with unidirectional I/O ports and an initiation rate of 3

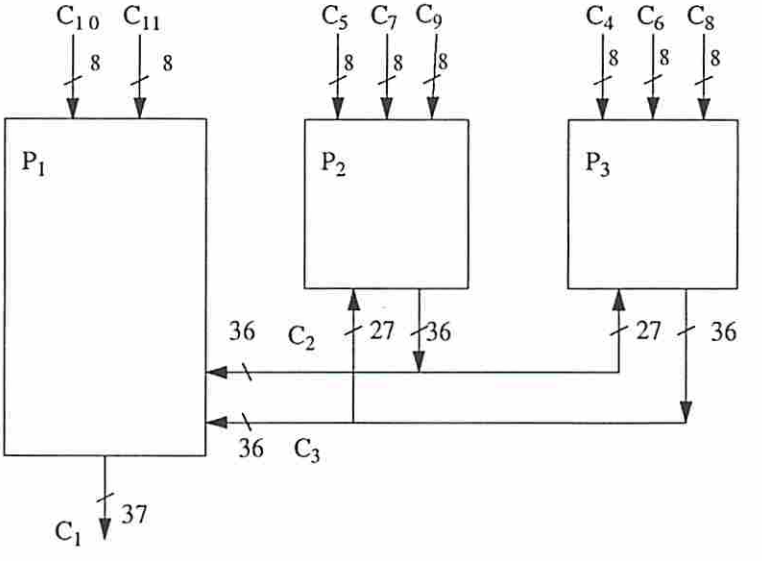


Figure 4.9: Interchip connection for the AR filter design with unidirectional I/O ports and an initiation rate of 4

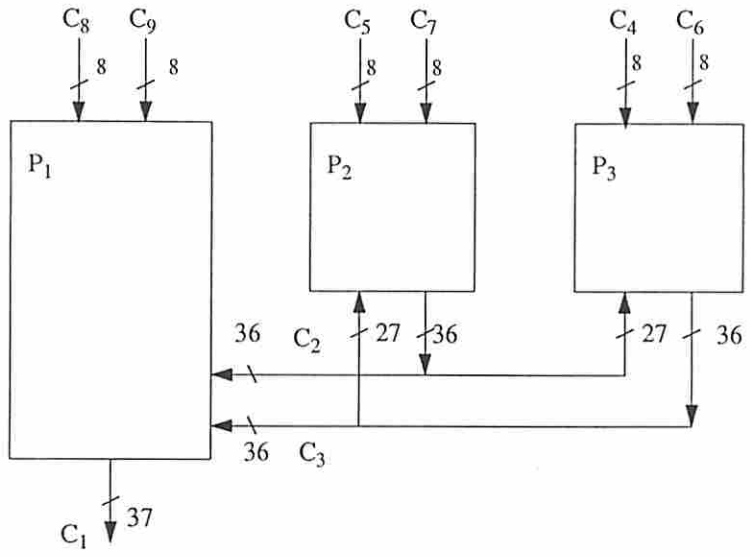


Figure 4.10: Interchip connection for the AR filter design with unidirectional I/O ports and an initiation rate of 5

Table 4.2. The schedules for the designs with these interchip connections and the resource constraints given are shown in Figures 4.11, 4.12, and 4.13. During the scheduling process, I/O operations have been reassigned to communication buses. The numbers of control steps required for the schedules with and without bus reassignment are given in the third and fourth columns of Table 4.2, respectively. From the results, it can be seen that the schedules with bus reassignments have smaller numbers of control steps than the schedules with static bus assignment.

Initiation Rate	#Pins used				#Control steps	
	P_0	P_1	P_2	P_3	w/ reassignment	w/o reassignment
3	109	133	87	87	11	12
4	101	125	87	87	14	15
5	85	125	79	79	9	13

Table 4.2: Summarized results for the AR filter designs with unidirectional I/O ports

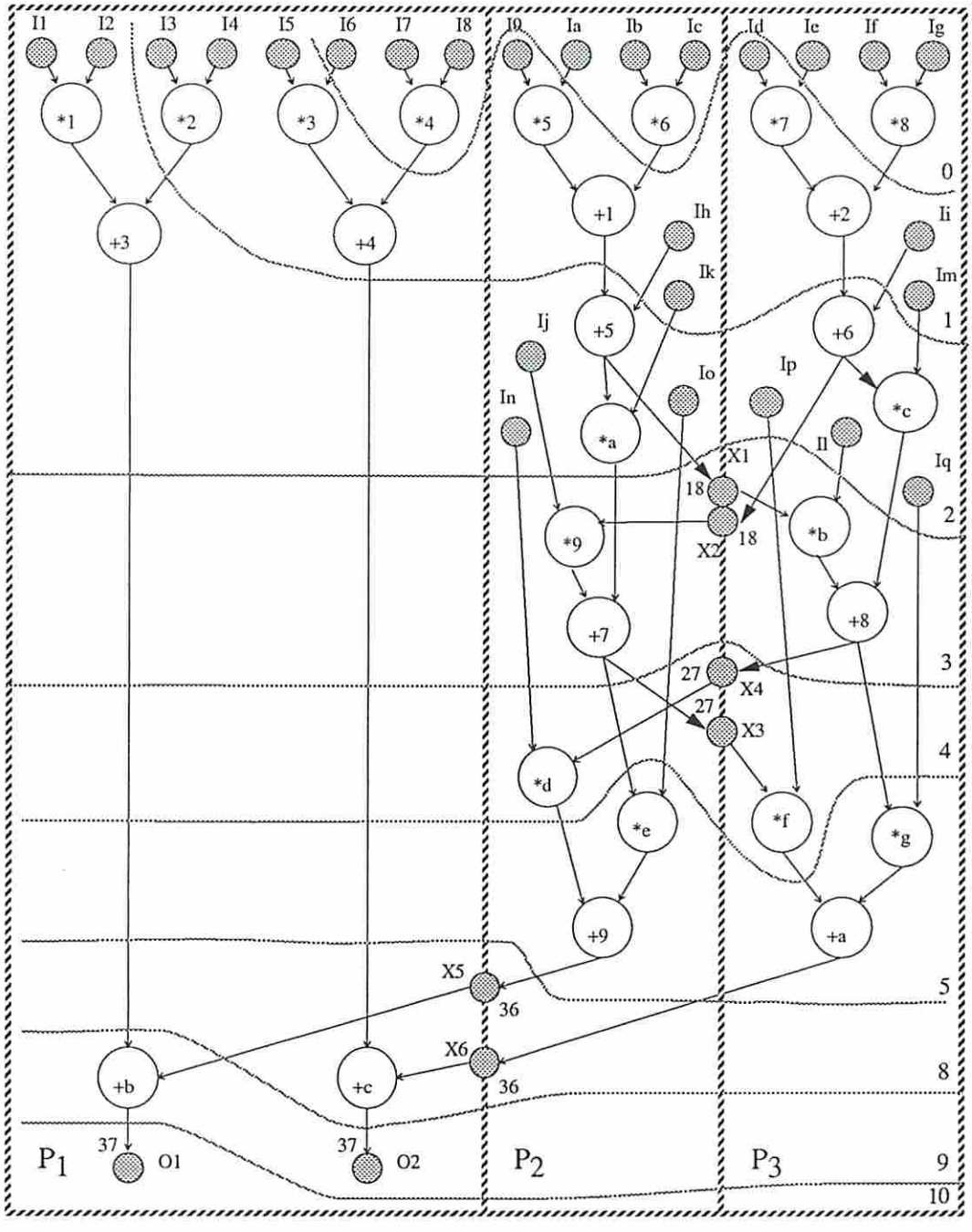


Figure 4.11: Schedule for the AR filter design with unidirectional I/O ports and an initiation rate of 3

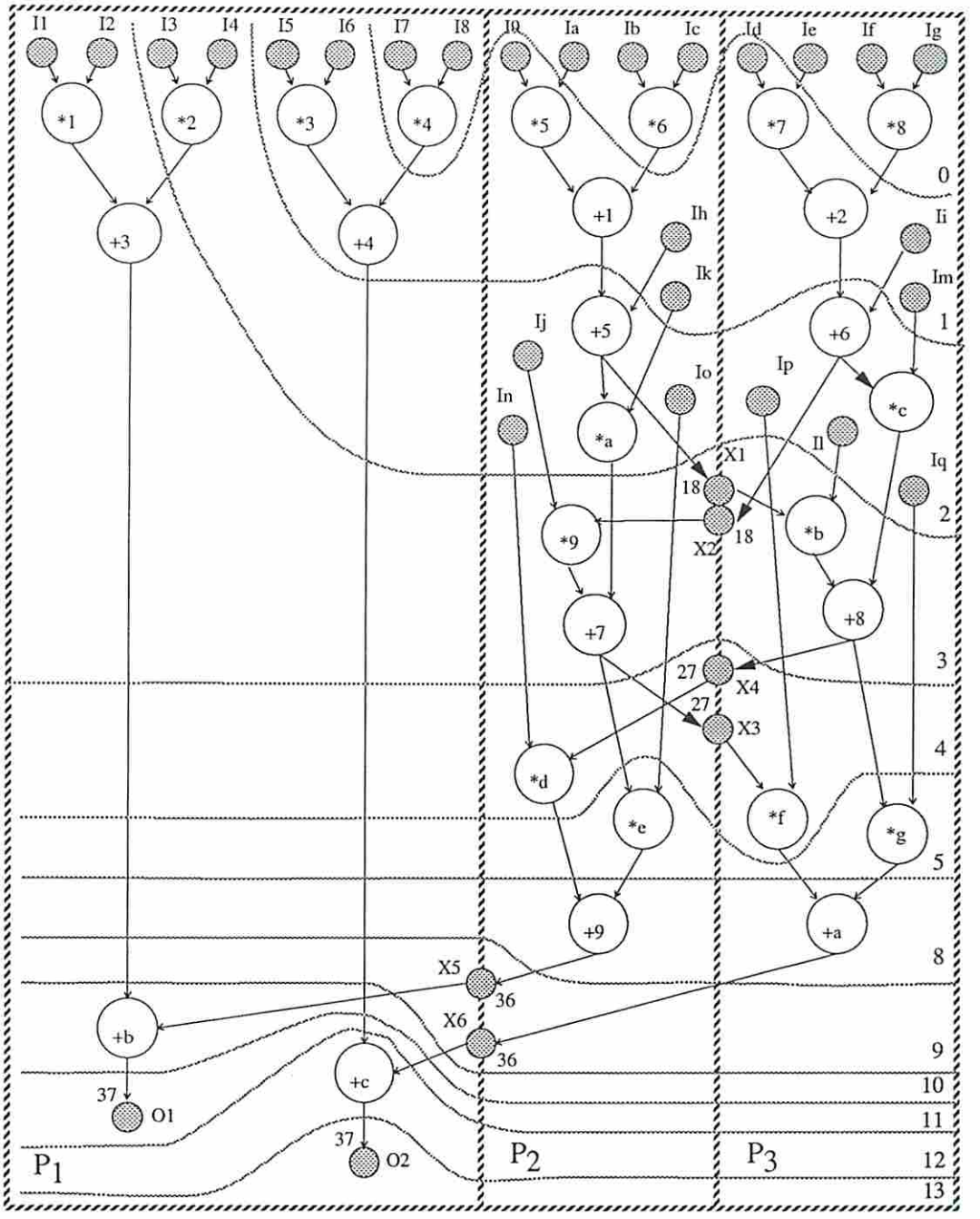


Figure 4.12: Schedule for the AR filter design with unidirectional I/O ports and an initiation rate of 4

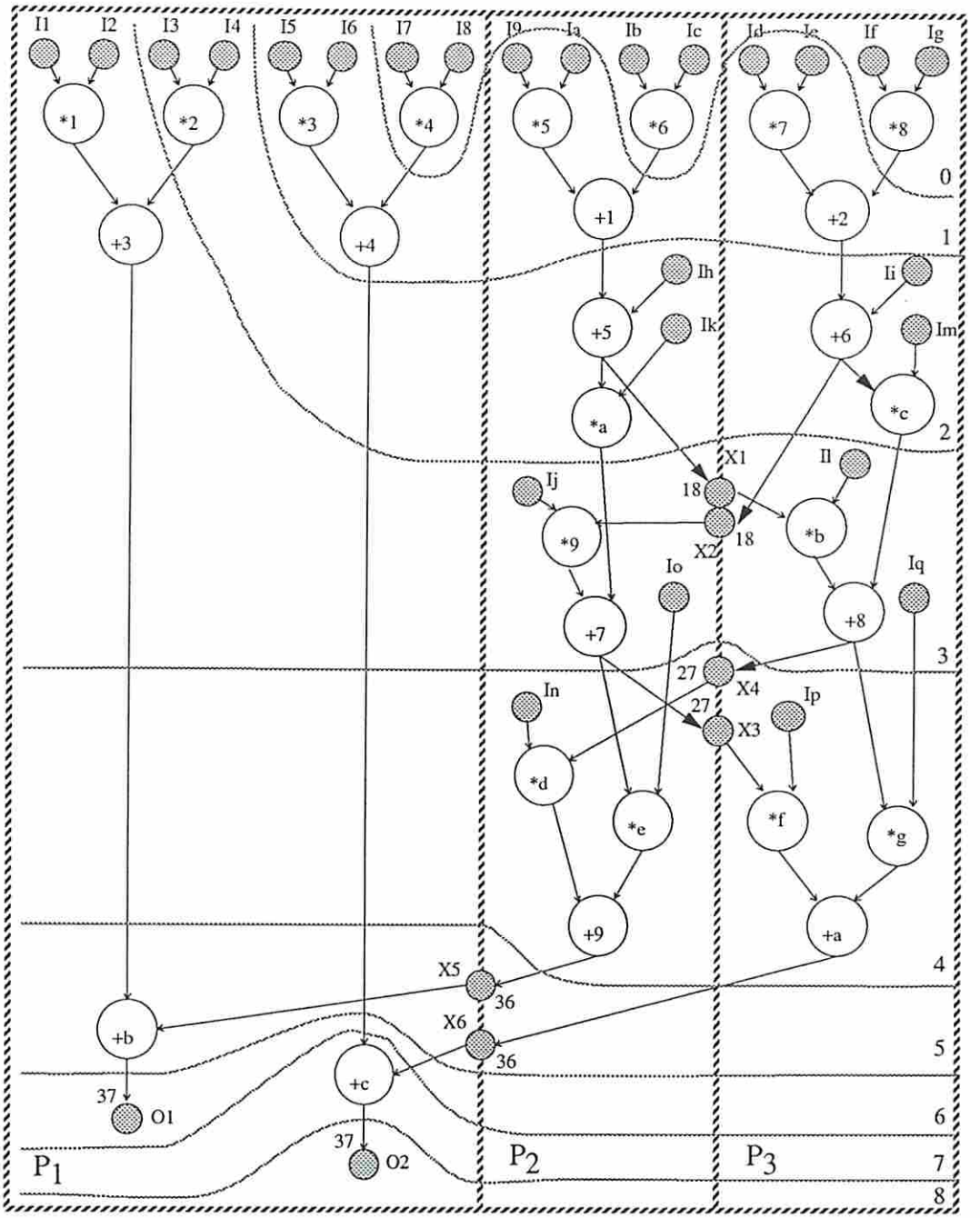


Figure 4.13: Schedule for the AR filter design with unidirectional I/O ports and an initiation rate of 5

Communication Bus	Initial Assignment	Final Assignment
C_1	O1 O2	O1 O2
C_2	X1 X3 X5	X1 X3 X5
C_3	X2 X4 X6	X2 X4 X6
C_4	Im Ip Iq	Ie Im Ip
C_5	Ik In Io	Ia Ik In
C_6	Ig Ii Il	Ig Ii Il
C_7	Ic Ih Ij	Ic Ih Ij
C_8	Id Ie If	Id If Iq
C_9	I9 Ia Ib	I9 Ib Io
C_{10}	I6 I7 I8	I1 I3 I8
C_{11}	I3 I4 I5	I4 I6
C_{12}	I1 I2	I2 I5 I7

Table 4.3: Bus assignment for the AR filter design with unidirectional I/O ports and an initiation rate of 3

Control Steps	Bus Allocation											
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
$0, 3, \ldots$	O2	X1	X2	Ie	Ia	Ig	Ic	If	Ib	I8	I6	I7
$1, 4, \ldots$	O1	X3	X4	Im	Ik	Ii	Ih	Id	I9	I3	I4	I5
$2, 5, \ldots$		X5	X6	Ip	In	Il	Ij	Iq	Io	I1		I2

Table 4.4: Bus allocation for the AR filter design with unidirectional I/O ports and an initiation rate of 3

The initial and final assignments of I/O operations to communication buses for the schedules are shown in Tables 4.3, 4.5, and 4.7. The communication bus allocations for these schedules and interchip connections are shown in Tables 4.4, 4.6, and 4.8.

4.4.1.2 Designs with bidirectional I/O ports

The partitioned AR filter has also been synthesized with the assumption that bidirectional I/O ports are allowed. The hardware resources available for designs with

Communication Bus	Initial Assignment	Final Assignment
C_1	O1 O2	O1 O2
C_2	X1 X3 X5	X1 X3 X5
C_3	X2 X4 X6	X2 X4 X6
C_4	Il Im Ip Iq	Ie Il Im
C_5	Ij Ik In Io	Ia Ij Ik
C_6	Ie If Ig Ii	Ig Ii Ip
C_7	Ia Ib Ic Ih	Ic Ih In
C_8	Id	Id If Iq
C_9	I9	I9 Ib Io
C_{10}	I5 I6 I7 I8	I2 I4 I6 I8
C_{11}	I1 I2 I3 I4	I1 I3 I5 I7

Table 4.5: Bus assignment for the AR filter design with unidirectional I/O ports and an initiation rate of 4

Control Steps	Bus Allocation										
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
$0, 4, \ldots$		X3	X4	Ie	Ia	Ig	Ic	If	Ib	I8	I7
$1, 5, \ldots$	O2	X5	X6	Im	Ik	Ii	Ih	Id	I9	I6	I5
$2, 6, \ldots$				Il	Ij	Ip	In	Iq	Io	I4	I3
$3, 7, \ldots$	O1	X1	X2							I2	I1

Table 4.6: Bus allocation for the AR filter design with unidirectional I/O ports and an initiation rate of 4

Communication Bus	Initial Assignment	Final Assignment
C_1	O1 O2	O1 O2
C_2	X1 X3 X5	X1 X3 X5
C_3	X2 X4 X6	X2 X4 X6
C_4	Ii Il Im Ip Iq	Id If Ii Il
C_5	Ih Ij Ik In Io	I9 Ib Ih Ij
C_6	Id Ie If Ig	Ie Ig Im Ip Iq
C_7	I9 Ia Ib Ic	Ia Ic Ik In Io
C_8	I4 I5 I6 I7 I8	I1 I4 I6 I8
C_9	I1 I2 I3	I2 I3 I5 I7

Table 4.7: Bus assignment for the AR filter design with unidirectional I/O ports and an initiation rate of 5

Control Steps	Bus Allocation								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$0, 5, \ldots$		X5	X6	If	Ib	Ig	Ic	I8	I7
$1, 6, \ldots$	O1			Id	I9	Ie	Ia	I6	I5
$2, 7, \ldots$				Ii	Ih	Im	Ik	I4	I3
$3, 8, \ldots$	O2	X1	X2	Il	Ij	Iq	Io	I1	I2
$4, 9, \ldots$		X3	X4			Ip	In		

Table 4.8: Bus allocation for the AR filter design with unidirectional I/O ports and an initiation rate of 5

Initiation Rate	P_0	P_1			P_2			P_3		
3	110P	100P	2+	2*	90P	2+	2*	90P	2+	2*
4	110P	100P	1+	1*	90P	1+	2*	90P	1+	2*
5	110P	100P	1+	1*	90P	1+	2*	90P	1+	2*

P: Pins, +: Adders, *: Multipliers

Table 4.9: Resource Constraints for the AR filter designs with bidirectional I/O ports

Initiation Rate	#Pins used				#Control steps	
	P_0	P_1	P_2	P_3	w/ reassignment	w/o reassignment
3	109	97	87	87	11	12
4	101	89	78	78	15	16
5	77	81	70	70	10	14

Table 4.10: Summarized results for the AR filter designs with bidirectional I/O ports

different initiation rates are shown in Table 4.9. The interchip connections for the different initiation rates are shown in Figures 4.14, 4.15, and 4.16. The actual numbers of I/O pins allocated for interchip connections are given in the second column of Table 4.10. The schedules for the designs with these interchip connections and the resource constraints given are shown in Figures 4.17, 4.18, and 4.19.

During the scheduling process, I/O operations have been reassigned to communication buses. The number of control steps required for the schedules with and without bus reassignment are given in the third and fourth columns in Table 4.10. From the results, it can be seen that the schedules with bus reassignments have smaller numbers of control steps than the schedules not allowed bus reassignment. The initial and final assignments of I/O operations to communication buses for the schedules are shown in Tables 4.11 - 4.13.

As we expected, the designs with bidirectional I/O ports require less I/O pins than the corresponding designs with only unidirectional I/O ports.

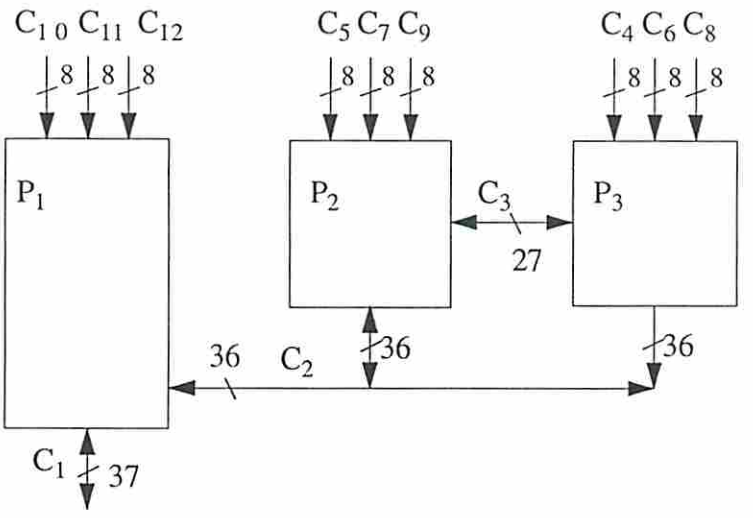


Figure 4.14: Interchip connection for the AR filter design with bidirectional I/O ports and an initiation rate of 3

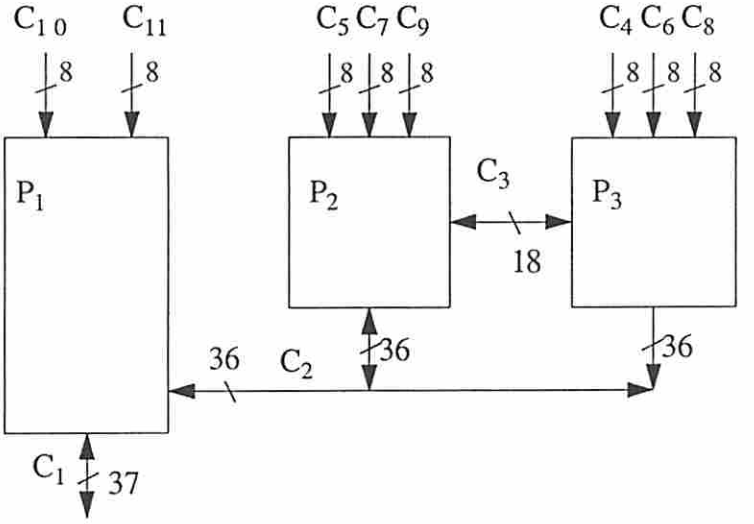


Figure 4.15: Interchip connection for the AR filter design with bidirectional I/O ports and an initiation rate of 4

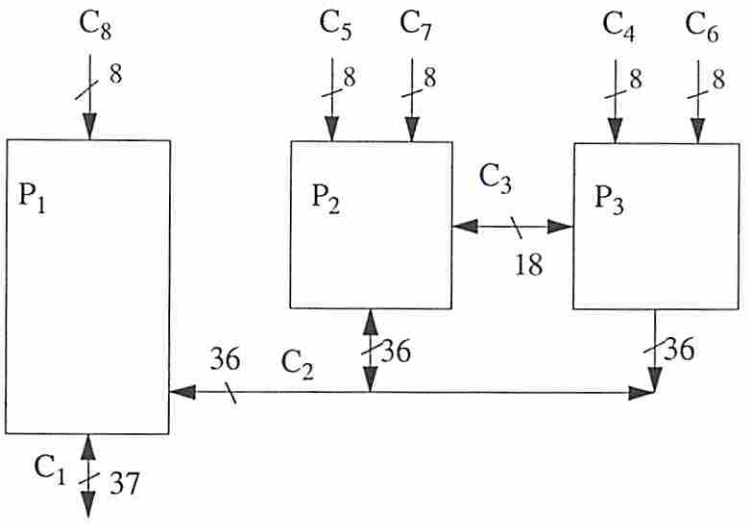


Figure 4.16: Interchip connection for the AR filter design with bidirectional I/O ports and an initiation rate of 5

Communication Bus	Initial Assignment	Final Assignment
C_1	I8 O1 O2	I8 O1 O2
C_2	X3 X5 X6	X1 X5 X6
C_3	X1 X2 X4	X2 X3 X4
C_4	Im Ip Iq	Ie Im Ip
C_5	Ik In Io	Ia Ik In
C_6	Ig Ii Il	Ig Ii Il
C_7	Ic Ih Ij	Ic Ih Ij
C_8	Id Ie If	Id If Iq
C_9	I9 Ia Ib	I9 Ib Io
C_{10}	I5 I6 I7	I2 I7
C_{11}	I2 I3 I4	I3 I5
C_{12}	I1	I1 I4 I6

Table 4.11: Bus assignment for the AR filter design with bidirectional I/O ports and an initiation rate of 3

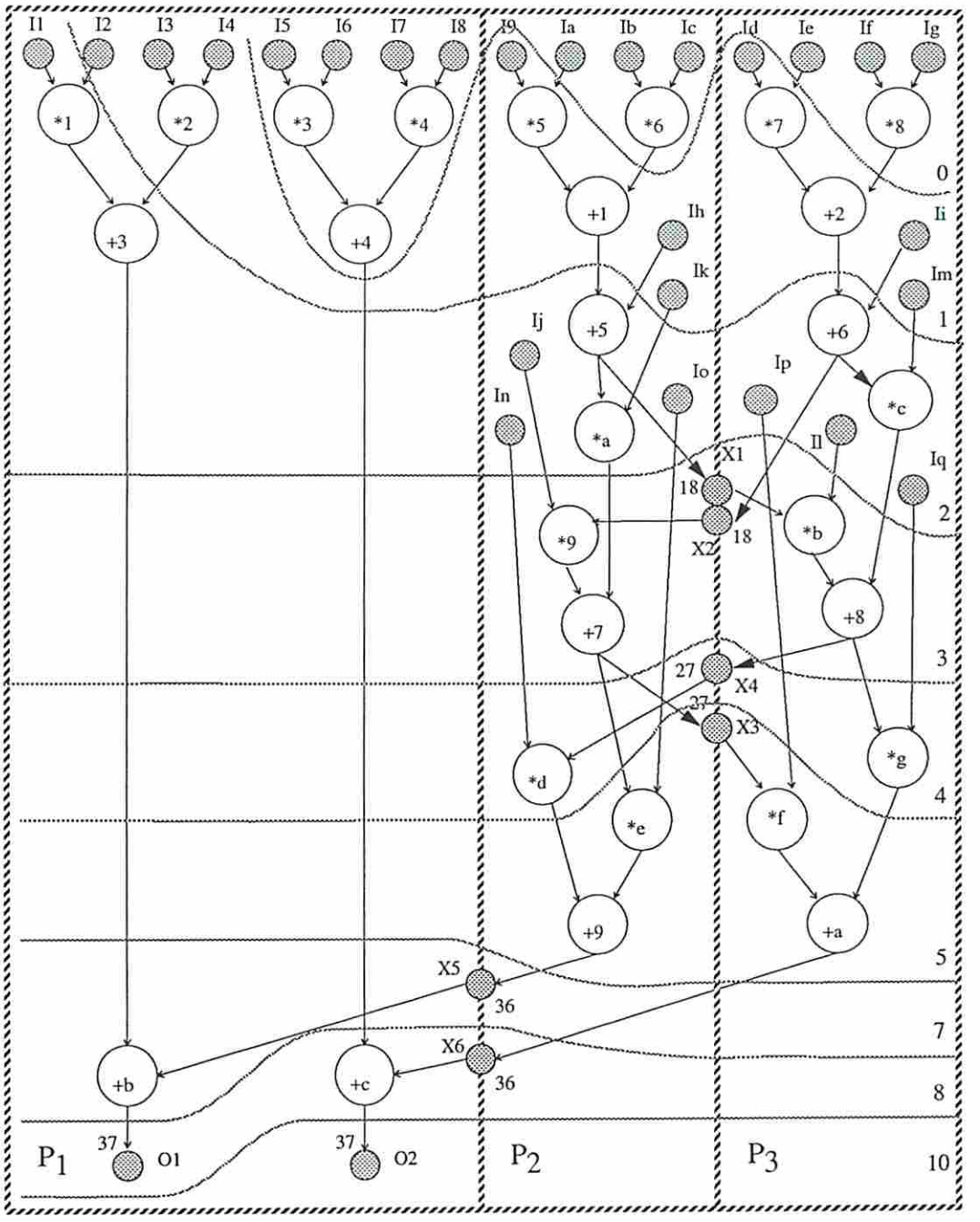


Figure 4.17: Schedule for the AR filter design with bidirectional I/O ports and an initiation rate of 3

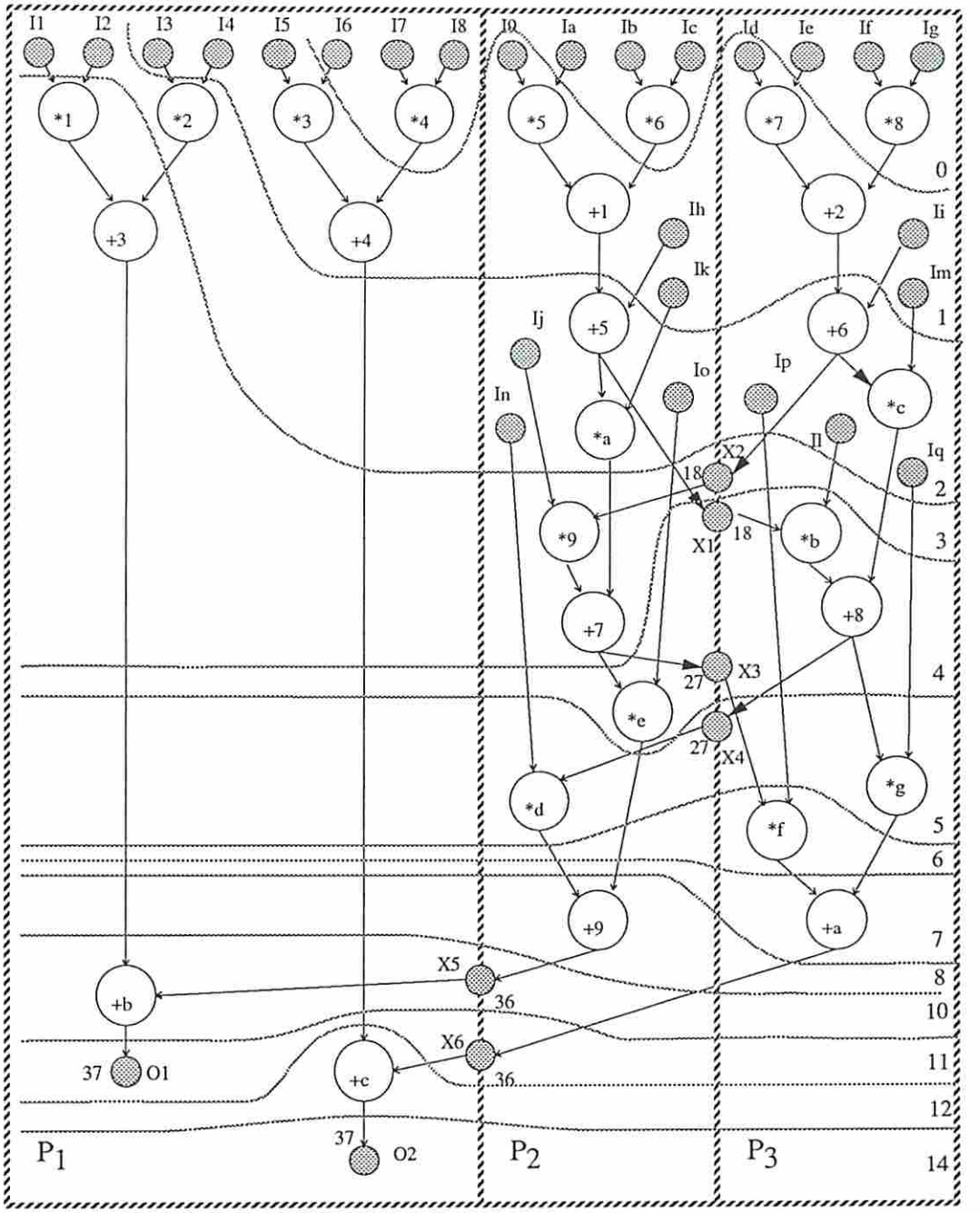


Figure 4.18: Schedule for the AR filter design with bidirectional I/O ports and an initiation rate of 4

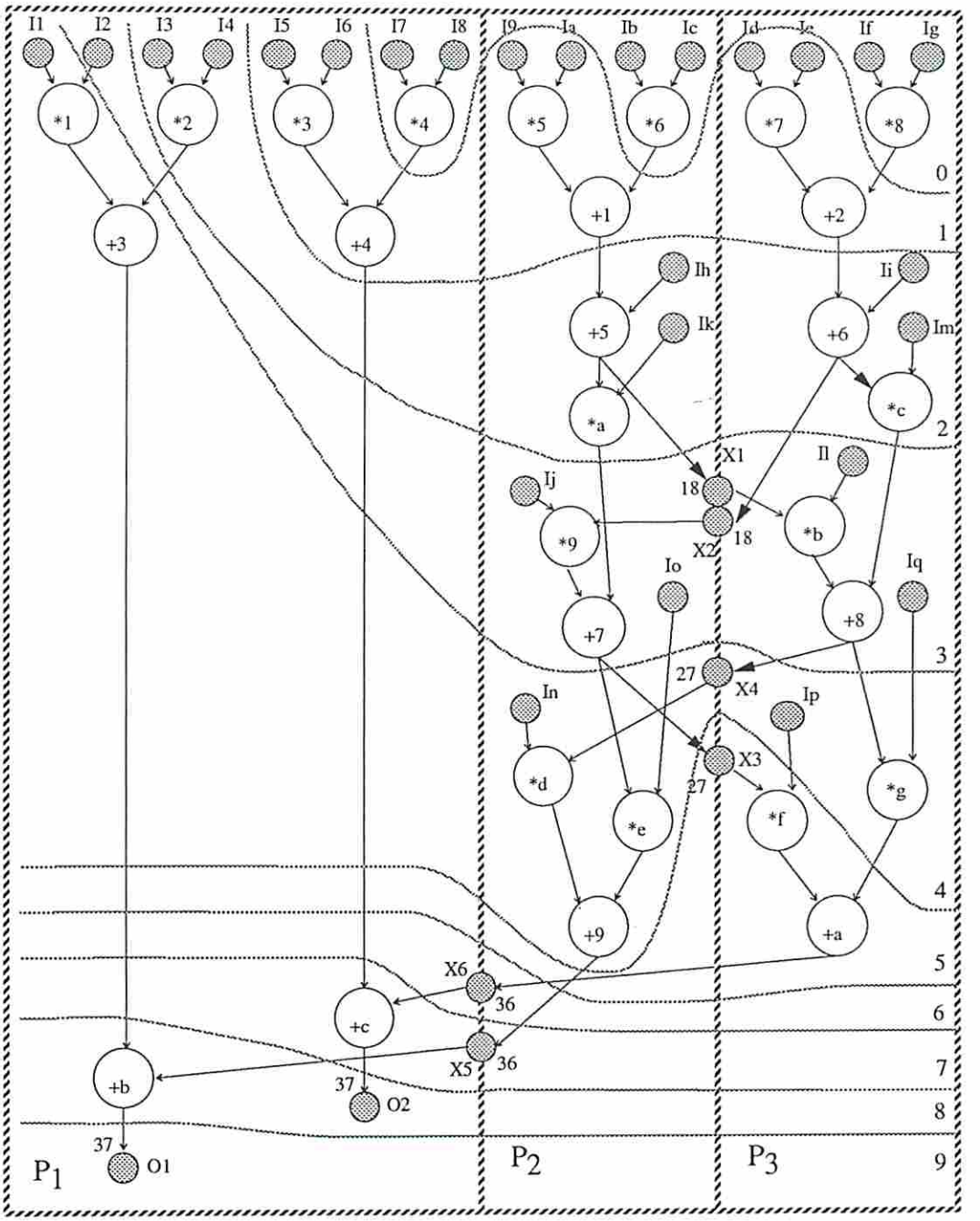


Figure 4.19: Schedule for the AR filter design with bidirectional I/O ports and an initiation rate of 5

Communication Bus	Initial Assignment	Final Assignment
C_1	I7 I8 O1 O2	I3 I8 O1 O2
C_2	X3 X4 X5 X6	X3 X4 X5 X6
C_3	X1 X2	X1 X2
C_4	Il Im Ip Iq	Ie Il Im
C_5	Ij Ik In Io	Ia Ij Ik
C_6	Ie If Ig Ii	Ig Ii Ip
C_7	Ia Ib Ic Ih	Ic Ih In
C_8	Id	Id If Iq
C_9	I9	I9 Ib Io
C_{10}	I3 I4 I5 I6	I1 I5 I6
C_{11}	I1 I2	I2 I4 I7

Table 4.12: Bus assignment for the AR filter design with bidirectional I/O ports and an initiation rate of 4

Communication Bus	Initial Assignment	Final Assignment
C_1	I6 I7 I8 O1 O2	I4 I6 I8 O1 O2
C_2	X1 X3 X4 X5 X6	X1 X3 X4 X5 X6
C_3	X2	X2
C_4	Ii Il Im Ip Iq	Id If Ii Il
C_5	Ih Ij Ik In Io	I9 Ib Ih Ij
C_6	Id Ie If Ig	Ie Ig Im Ip Iq
C_7	I9 Ia Ib Ic	Ia Ic Ik In Io
C_8	I1 I2 I3 I4 I5	I1 I2 I3 I5 I7

Table 4.13: Bus assignment for the AR filter design with bidirectional I/O ports and an initiation rate of 5

L	P_0	P_1	P_2	P_3	P_4	P_5
5	32P	(64P,3+,1*)	(48P,1+,1*)	(48P,2+,2*)	(64P,3+,2*)	(32P,1+,2*)
6	32P	(64P,2+,1*)	(48P,1+,1*)	(32P,1+,1*)	(48P,2+,1*)	(32P,1+,1*)
7	32P	(48P,1+,1*)	(48P,1+,1*)	(48P,1+,1*)	(48P,2+,1*)	(32P,1+,1*)

L: Initiation rate, P: Pins, +: Adders, *: Multipliers

Table 4.14: Resource Constraints for the elliptic filter designs with unidirectional I/O ports

4.4.2 Fifth-order Wave Elliptic Filter

The fifth-order wave elliptic filter [KWK85] is partitioned as shown in Figure 4.20, and all values are assumed to have 16 bits. It is also assumed that I/O operations and additions take 1 cycle, while multiplications take 2 cycles and are not pipelined. In the elliptic filter, there are a number of data recursive edges[4] with *degree* 1. A degree d on an edge denotes that the value produced by the source operation will be consumed by the destination operation d iterations (execution instances) later. With a single data stream, the initiation rate for the filter must be at least 20 cycles without retiming or transformation [PR91, LP91] to optimize the filter, because the length of the critical loop (X33, +2, Xf, +5, Xe, ..., Xh, ..., Xj, ..., +28) is 20 cycles. In order to demonstrate the sharing of multiple buses and let more I/O operations execute concurrently, the degree on each data recursive edge has been modified to 4. Thus, the design for the modified filter can operate on four independent multiplexed streams of data, and the minimum initiation rate becomes 5 cycles.

4.4.2.1 Designs with Unidirectional I/O Ports

The partitioned elliptic filter has been synthesized with initiation rates of 5, 6, and 7. The hardware resources available for designs with different initiation rates are shown in Table 4.14. The schedule for the design with an initiation rate of

[4]Data recursive edges will be discussed more detail in Chapter 7.

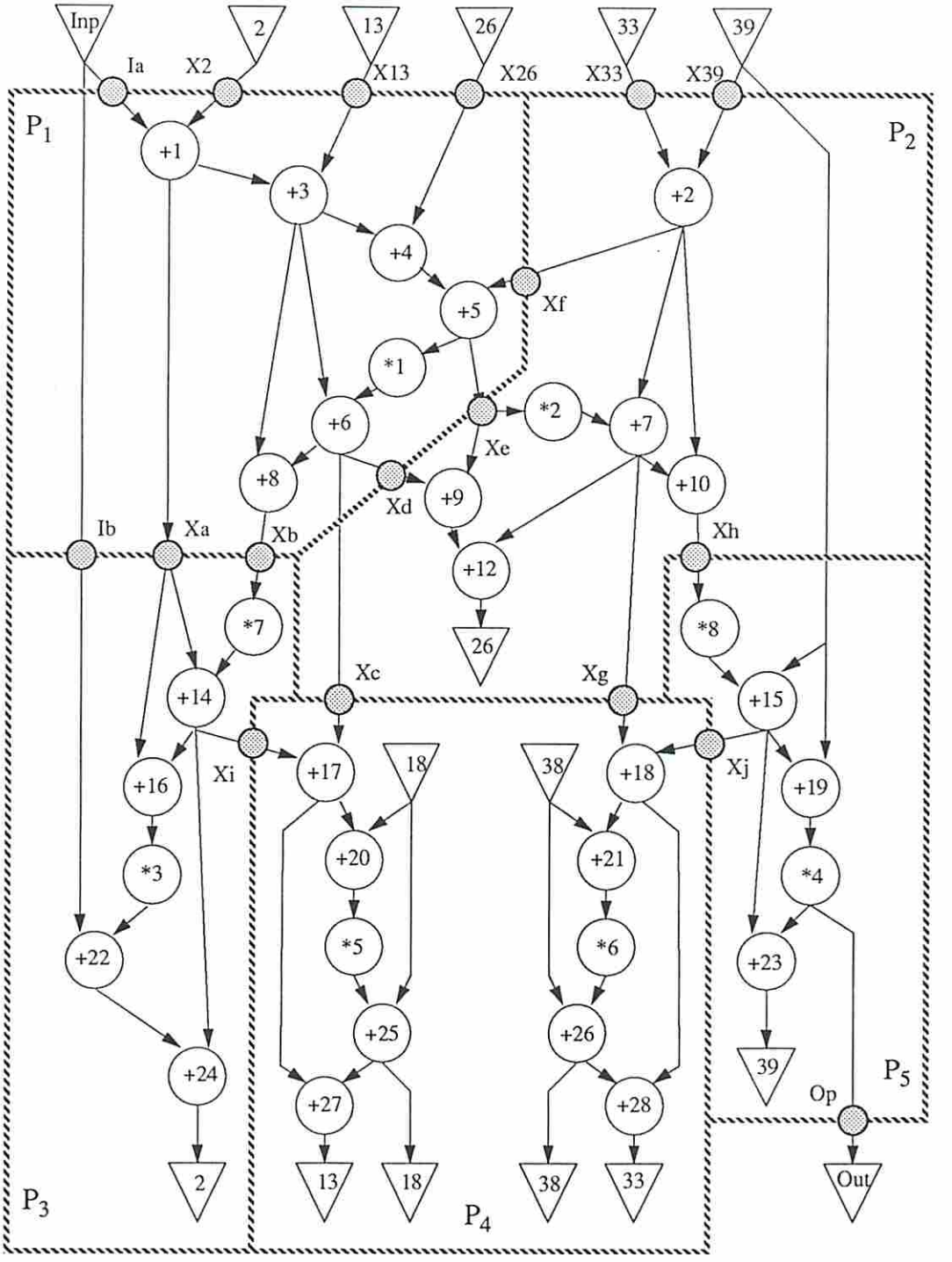


Figure 4.20: A partitioned elliptic filter

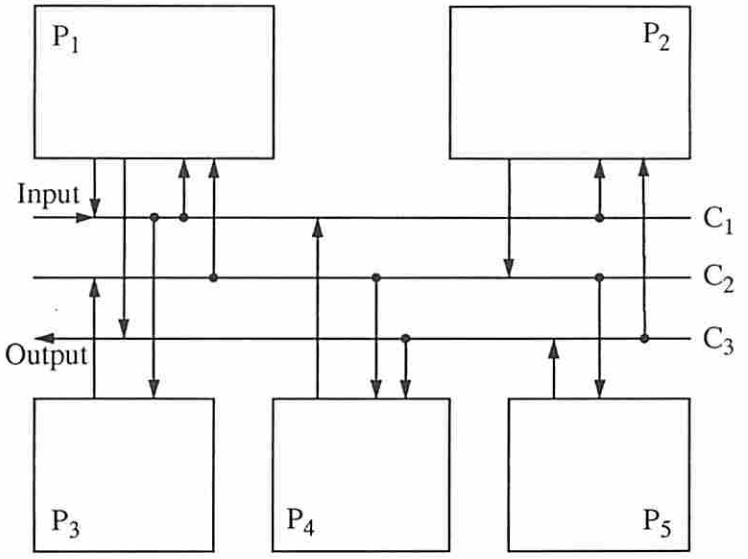


Figure 4.21: Interchip connection for the elliptic filter design with unidirectional I/O ports and an initiation rate of 6

5 cannot be obtained under the resource constraints even if one exists because of the very tight time constraints imposed by data dependencies between execution instances and the greedy heuristic of the list scheduling.

Interchip connections for designs with different initiation rates are shown in Figures 4.21, and 4.22. The actual numbers of I/O pins used for interchip connections are the same as the I/O pin constraints. The schedules for the designs with these interchip connections and the resource constraints given are shown in Figures 4.23 and 4.24. Note that, in these schedules, the I/O operations with negative indexes are scheduled in the control steps with negative numbers. This means that the values produced by previous execution have been input and stored before the current execution instance is initiated. No bus reassignments are done during scheduling because the initial assignment is the only valid bus assignment for the given interchip connections. The communication bus allocations for these schedules and interchip connections are shown in Tables 4.15 and 4.16. Note that I/O operations which transfer the same value and are scheduled in the same

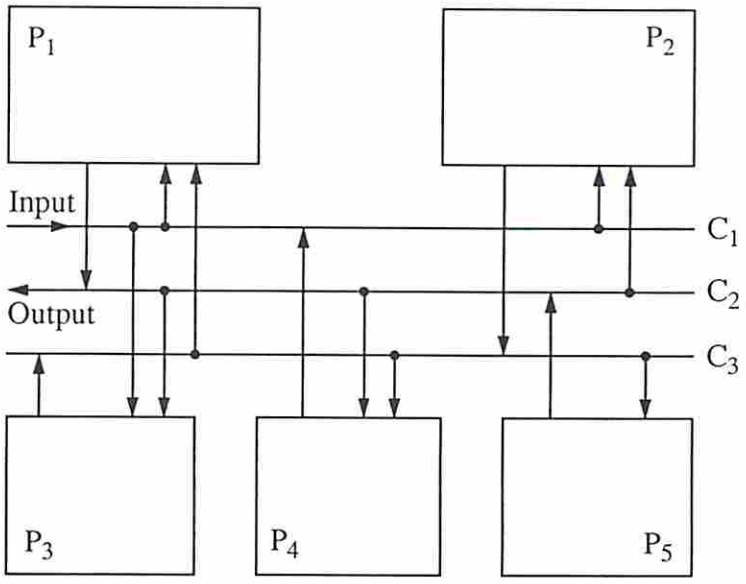


Figure 4.22: Interchip connection for the elliptic filter design with unidirectional I/O ports and an initiation rate of 7

Control Steps	Bus Allocation		
	C_1	C_2	C_3
$0, 6, \ldots$	(Ia,Ib)	X26	X39
$1, 7, \ldots$		Xf	Op
$2, 8, \ldots$	Xa	Xi	(Xc,Xd)
$3, 9, \ldots$	Xb	Xg	Xj
$4, 10, \ldots$	X33	Xh	
$5, 11, \ldots$	X13	X2	Xe

Table 4.15: Bus allocation for the elliptic filter design with unidirectional I/O ports and an initiation rate of 6

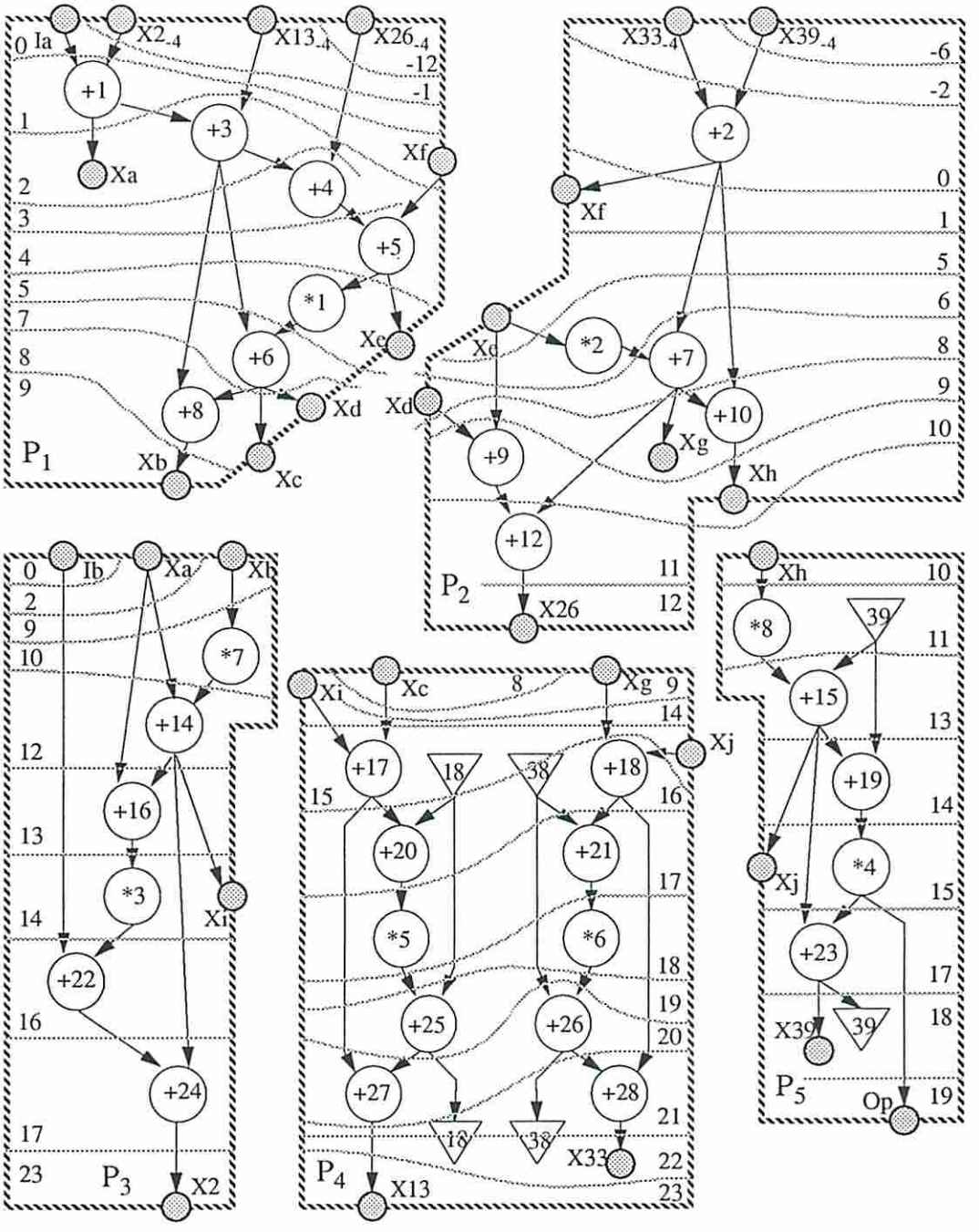


Figure 4.23: Schedule for the elliptic filter design with unidirectional I/O ports and an initiation rate of 6

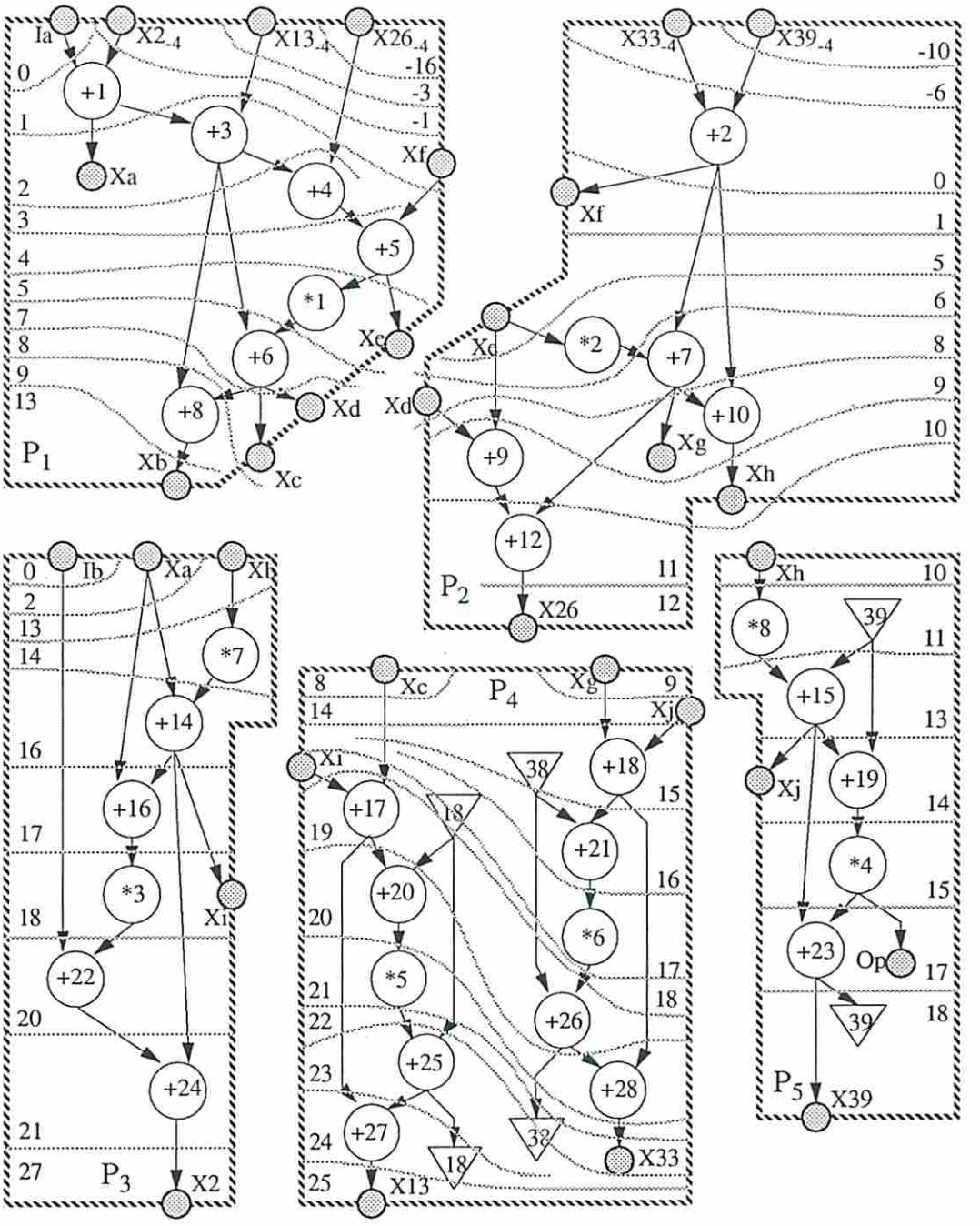


Figure 4.24: Schedule for the elliptic filter design with unidirectional I/O ports and an initiation rate of 7

Control Steps	Bus Allocation		
	C_1	C_2	C_3
$0, 7, \ldots$	(Ia,Ib)	Xj	
$1, 8, \ldots$	X33	(Xc,Xd)	Xf
$2, 9, \ldots$		Xa	Xg
$3, 10, \ldots$		Op	Xh
$4, 11, \ldots$	X13	X39	Xi
$5, 12, \ldots$		Xe	X26
$6, 13, \ldots$		Xb	X2

Table 4.16: Bus allocation for the elliptic filter design with unidirectional I/O ports and an initiation rate of 7

L	P_0	P_1	P_2	P_3	P_4	P_5
5	32P	(48P,2+,1*)	(48P,1+,1*)	(32P,2+,2*)	(32P,3+,2*)	(32P,1+,1*)
6	32P	(32P,2+,1*)	(32P,1+,1*)	(32P,1+,1*)	(32P,2+,1*)	(32P,1+,1*)
7	32P	(32P,1+,1*)	(32P,1+,1*)	(16P,1+,1*)	(32P,2+,1*)	(16P,1+,1*)

L: Initiation rate, P: Pins, +: Adders, *: Multipliers

Table 4.17: Resource Constraints for the elliptic filter designs with bidirectional I/O ports

control step can be assigned to the same communication bus without bus conflicts. As the design with an initiation of 7, assigning I/O operations Ia and Ib, which are scheduled in control step 0, to communication bus C_1 does not result in bus conflict.

4.4.2.2 Designs with Bidirectional I/O Ports

The partitioned elliptic filter has also been synthesized with the assumption that bidirectional I/O ports are allowed. The hardware resource constraints are shown in Table 4.17. List scheduling cannot find any schedule for design with an initiation rate of 5 under the resource constraints even one exists for the same reasons given above.

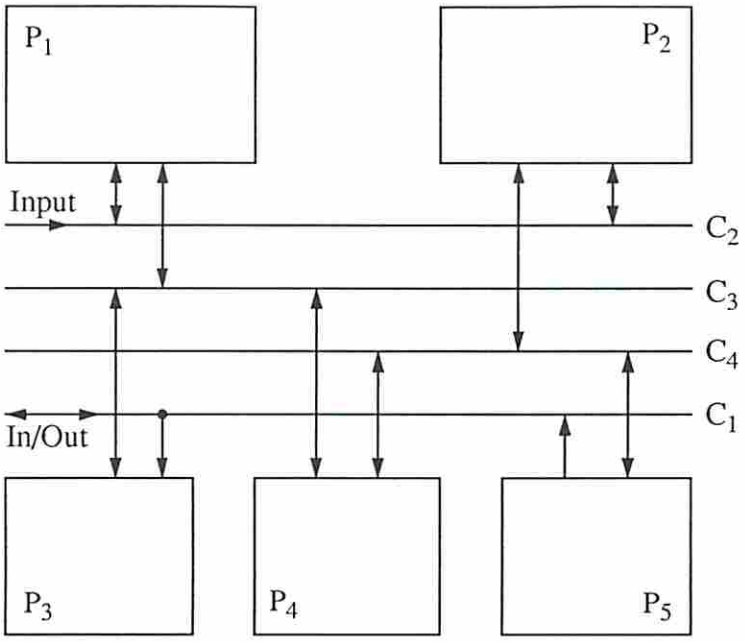


Figure 4.25: Interchip connection for the elliptic filter design with bidirectional I/O ports and an initiation rate of 6

The interchip connections for the designs with initiation rates of 6 and 7 are shown in Figures 4.25, and 4.26, respectively. The actual numbers of I/O pins used for interchip connections are the same as the I/O pin constraints. The schedules for the designs with these interchip connections and the resource constraints given are shown in Figures 4.27 and 4.28. In these schedules, the I/O operations with negative indexes are scheduled in the control steps with negative numbers. The meaning of this is the same given above. No bus reassignments are done during scheduling. The communication bus allocations for these schedules and interchip connections are shown in Tables 4.18 and 4.19. Note that I/O operations which transfer the same value and are scheduled in the same control step can be assigned to the same communication bus without bus conflicts. Also, I/O operations transferring the same value are not restricted to be scheduled in the same control step using the same communication bus. As the design with an initiation of 7, assigning I/O operations Ia and Ib, which are scheduled in control

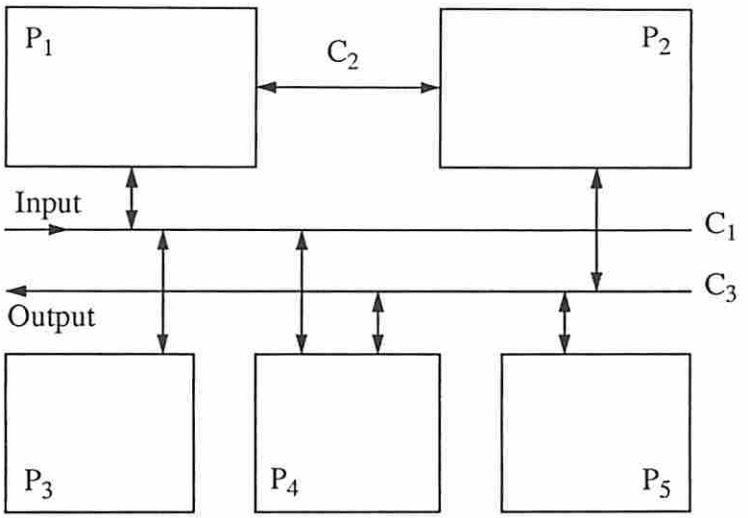


Figure 4.26: Interchip connection for the elliptic filter design with bidirectional I/O ports and an initiation rate of 7

Control Steps	Bus Allocation			
	C_1	C_2	C_3	C_4
$0, 6, \ldots$	Ib	Ia	X2	X39
$1, 7, \ldots$		Xf	Xi	
$2, 8, \ldots$		Xd	Xa	Xj
$3, 9, \ldots$		X26	Xb	Xg
$4, 10, \ldots$			Xc	Xh
$5, 11, \ldots$	Op	Xe	X13	X33

Table 4.18: Bus allocation for the elliptic filter design with bidirectional I/O ports and an initiation rate of 6

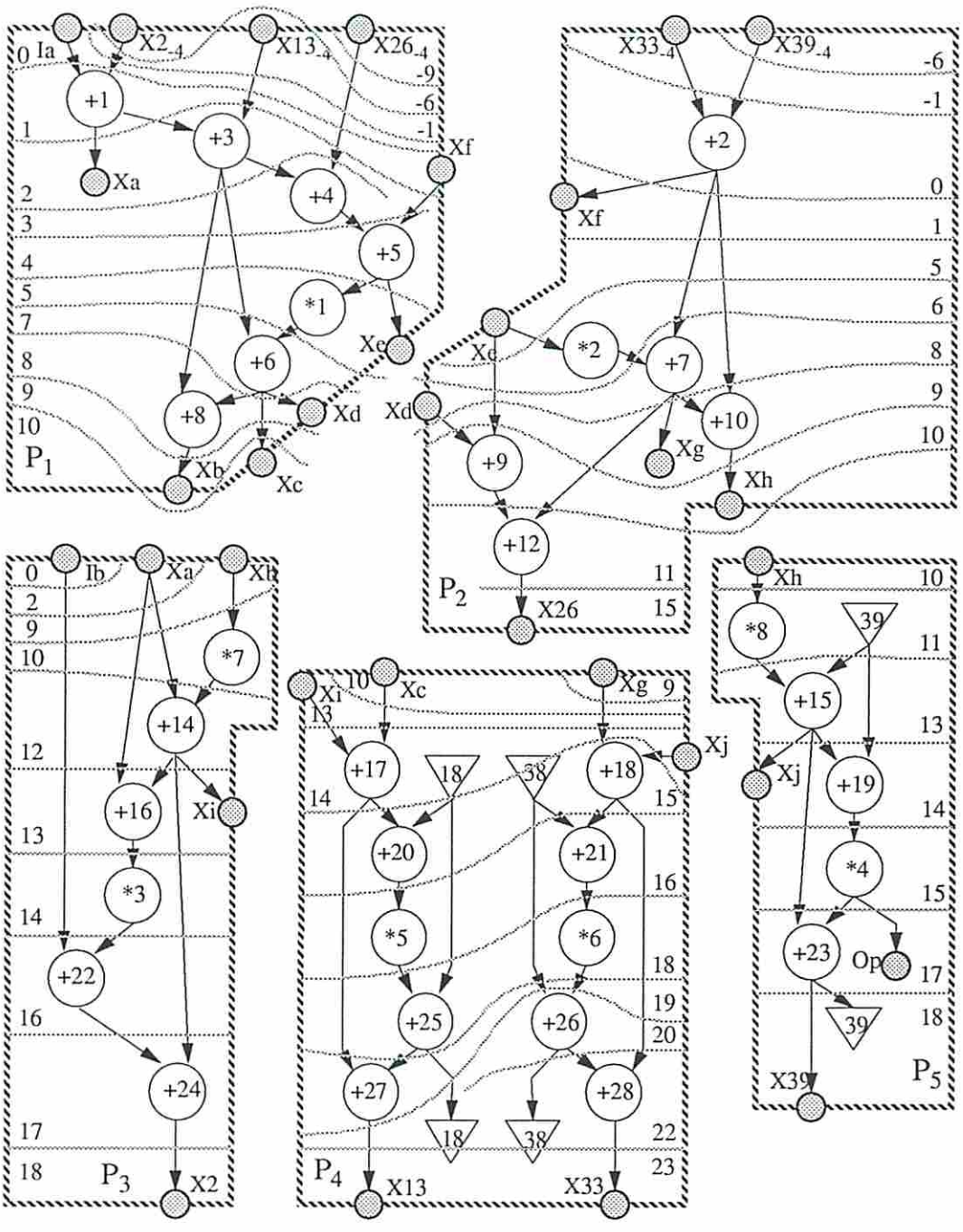


Figure 4.27: Schedule for the elliptic filter design with bidirectional I/O ports and an initiation rate of 6

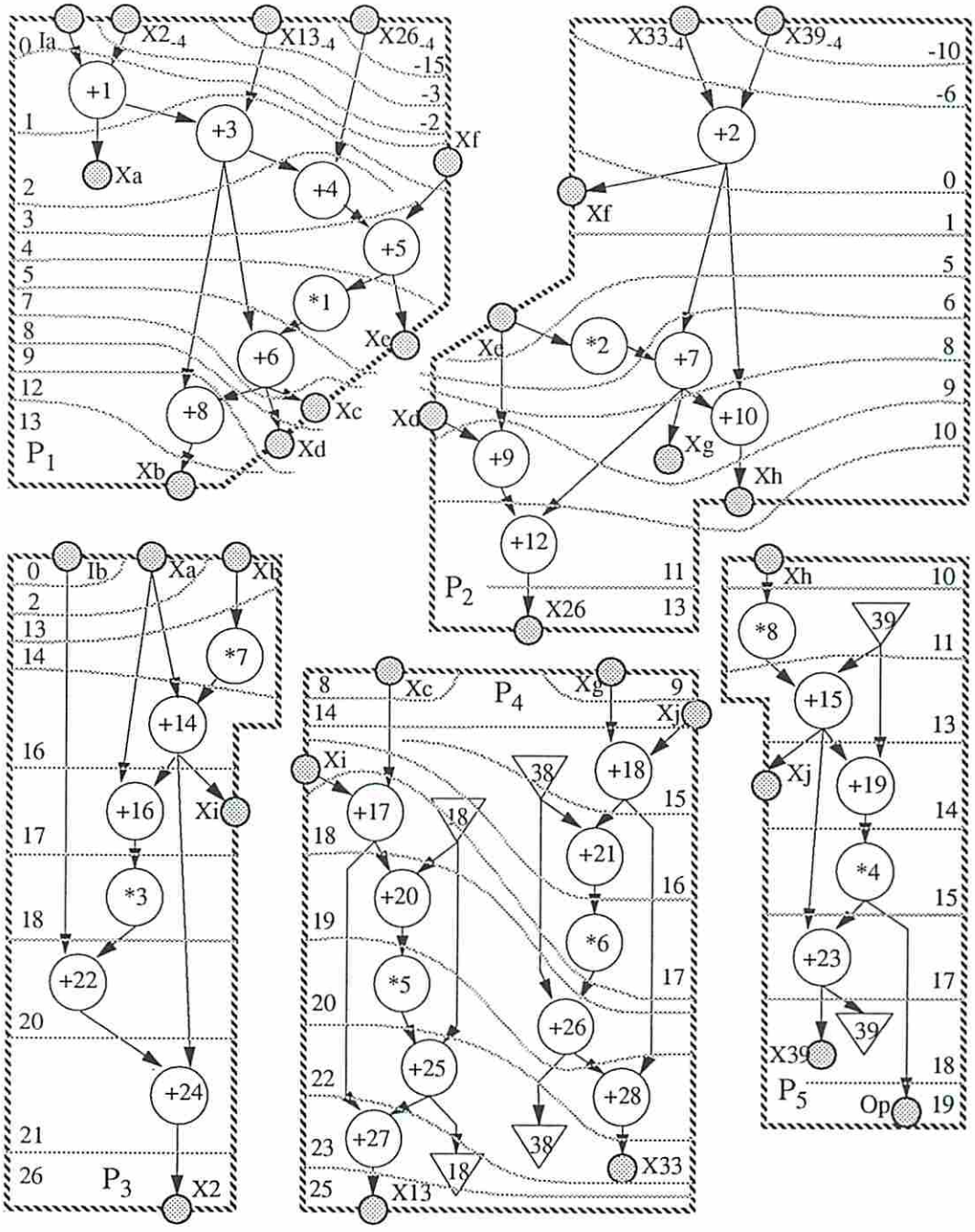


Figure 4.28: Schedule for the elliptic filter design with bidirectional I/O ports and an initiation rate of 7

Control Steps	Bus Allocation		
	C_1	C_2	C_3
$0, 7, \ldots$	(Ia,Ib)		Xj
$1, 8, \ldots$	Xc	Xf	X33
$2, 9, \ldots$	Xa	Xd	Xg
$3, 10, \ldots$	Xi		Xh
$4, 11, \ldots$	X13		X39
$5, 12, \ldots$	X2	Xe	Op
$6, 13, \ldots$	Xb	X26	

Table 4.19: Bus allocation for the elliptic filter design with bidirectional I/O ports and an initiation rate of 7

step 0, to communication bus C_1 does not result in bus conflict. I/O operations Xc and Xd are scheduled in control step 8 using bus C_1 and control step 9 using bus C_2, respectively.

Also, the designs with bidirectional I/O ports require less I/O pins than the corresponding designs with only unidirectional I/O ports.

Chapter 5

Interchip Connection Synthesis After Scheduling

In Chapter 4, an approach to multiple-chip design synthesis, in which the interchip connections are synthesized before scheduling, has been discussed. This chapter contains another approach to the synthesis problem, in which scheduling is performed before interchip connection synthesis. We also make the same assumptions that every communication bus can be used to transfer at most one value in a cycle, and that values are transferred as a whole. Then, the experimental results are compared to those obtained by the previous approach.

5.1 Scheduling

In the prototype program, Force-Directed Scheduling [PK89] is used to schedule a CDFG that has been partitioned into a number of partitions, each of which will be implemented in a chip, given an initiation rate and pipe length. All partitions are scheduled at the same time because of the interdependency among partitions imposed by the constraints that output and inputs of each value between partitions must take place in the same cycle. FDS tries to minimize the number of hardware resources required by balancing the utilization of the resources. In FDS, the distribution graphs for all operation types are required to derive the force of assigning an operation to a control step. Intuitively, the distribution graph of I/O

operation types can be obtained by combining the distribution graphs of input operation types and output operation types of the corresponding partitions since an I/O operation consists of an output operation from a partition and an input operation to another partition, and we try to do so. Unfortunately, unlike the cases of functional units, balancing the distribution graph for I/O operation types does not necessarily lead to the minimum number of I/O pins required because no switching devices are allowed off-chip.

5.2 Interchip Connection Synthesis

In pipelined designs, an execution instance will be initiated before the previous execution instance is completed. So, the operations scheduled in the same control step group are executed overlapped, and cannot share resources. The operations scheduled in different control step groups will never be executed concurrently, and can share the same hardware resources. Two I/O operations $w_1 = (P_{i_1}, P_{j_1})$, and $w_2 = (P_{i_2}, P_{j_2})$ are said to be *compatible* if and only if they can share a communication bus. The I/O operations are said to be in *conflict* if they are not compatible. The I/O operations which are scheduled in different control step groups can share a communication bus, and so are compatible. I/O operations which are scheduled in the same control step group can share a communication bus only if they transfer the same value in the same control step.

After scheduling is performed, each I/O operation is assigned to one of the control step groups. Then, the interchip connection can be determined with the goal of minimizing the number of I/O pins required. The problem can be modeled as a compatibility graph, in which each node represents an I/O operation. There is an edge between two nodes if and only if their corresponding I/O operations are compatible. On each edge is an associated weight, which reflects the benefit of having the I/O operations corresponding to these two end nodes assigned to a

communication bus. The weight, $weight(w_1, w_2)$, on the edge between w_1 and w_2 corresponding to two compatible I/O operations is given by

$$weight(w_1, w_2) = \begin{cases} (wf_i + wf_j)\min(B_{w1}, B_{w2}) & \text{if } i_1 = i_2 \wedge j_1 = j_2 \\ wf_i \min(B_{w1}, B_{w2}) & \text{if } i_1 = i_2 \wedge j_1 \neq j_2 \\ wf_j \min(B_{w1}, B_{w2}) & \text{if } i_1 \neq i_2 \wedge j_1 = j_2 \\ 0 & \text{otherwise} \end{cases}$$

where wf_i is a weighting factor used to specify the importance of sharing I/O pins of partition P_i, and B_{w1} and B_{w2} are the bit widths of I/O operations $w1$ and $w2$, respectively. The weight on an edge denotes the number of I/O pins which can be shared, and so can be saved if the I/O operations corresponding to these two end nodes are assigned to a shared communication bus when $wf_i = 1$ for all i. The minimum of two bit widths is taken because it indicates the number of pins which can be saved if the two I/O operations share a communication bus. The minimization of I/O pins for different partitions can compete with each other. The partition with a higher weighting factor will be given priority over the one with a lower value. Note that an edge can have a zero weight, and it is quite different from no edge at all since the I/O operations corresponding to the two nodes with an edge with a zero weight connected to them can share a communication bus even if they do not share I/O pins.

For the case of bidirectional I/O ports, in which an I/O port can be used as either a source or destination of an I/O transfer, both I/O operations $w = (P_i, P_j)$ and $w' = (P_j, P_i)$ require the same communication path on a communication bus if they are assigned to the same communication bus.

The problem of determining interchip connection with the goal of minimizing the total number of I/O pins required can be reduced to the problem of partitioning the compatibility graph into a number of disjoint cliques having the largest gain, where the gain is the summation of the weights on all edges that are contained in the cliques. The nodes in a clique represent the fact that the corresponding I/O operations are assigned to the same communication bus. The number of

communication buses is equal to the number of cliques partitioned. The structure of a communication bus (the I/O ports of which chips are connected to the bus) can be easily determined from the assignment of I/O operations to the bus since there must be a communication path with a sufficient bit width for each I/O operation assigned to the bus.

The problem is similar to the clique partitioning problem, which had initially been used by Tseng and Siewiorek [TS83] to model the problem of operator, register, and on-chip interconnection binding, and which is widely used for synthesis. The general clique partitioning problem has been known to be NP-hard. For *interval graphs*, which represent the intersection between a set of intervals along the real axis, the left-edge algorithm [HS71], which is a polynomial time algorithm, can be used to partition the graph into a minimum numbers of cliques. Springer [Spr91] has shown that the least-cost clique partitioning problem is NP-hard even for the interval graph. We believe that the problem of determining the interchip connection with the goal of minimizing the number of I/O pins required is also an NP-hard problem.

The compatibility graph of our problem has some special properties. The graph can be divided into L groups $G_0, G_1, \ldots, G_{L-1}$, as shown in Figure 5.1, where L is the initiation rate of the design. Group G_k contains the nodes which correspond to the I/O operations scheduled in control step group k. So, there is an edge between every node in G_i and every node in G_j (for $i \neq j$). Group G_k can be further divided into a number of subgroups $SG_{k,1}, SG_{k,2}, \ldots, SG_{k,n_k}$. The nodes in subgroup $SG_{k,j}$ correspond to the I/O operations which transfer the same value at the same control step and so can share a communication bus. Each subgroup usually contains only one node. There is an edge between any two nodes in a subgroup, while there is no edge between any two nodes in two different subgroups in the same group. Finding a set of cliques is equivalent to finding a set of disjoint subsets, S_k, in which there can be more than one node from the same group only if all of the nodes from the same group are in the same subgroup.

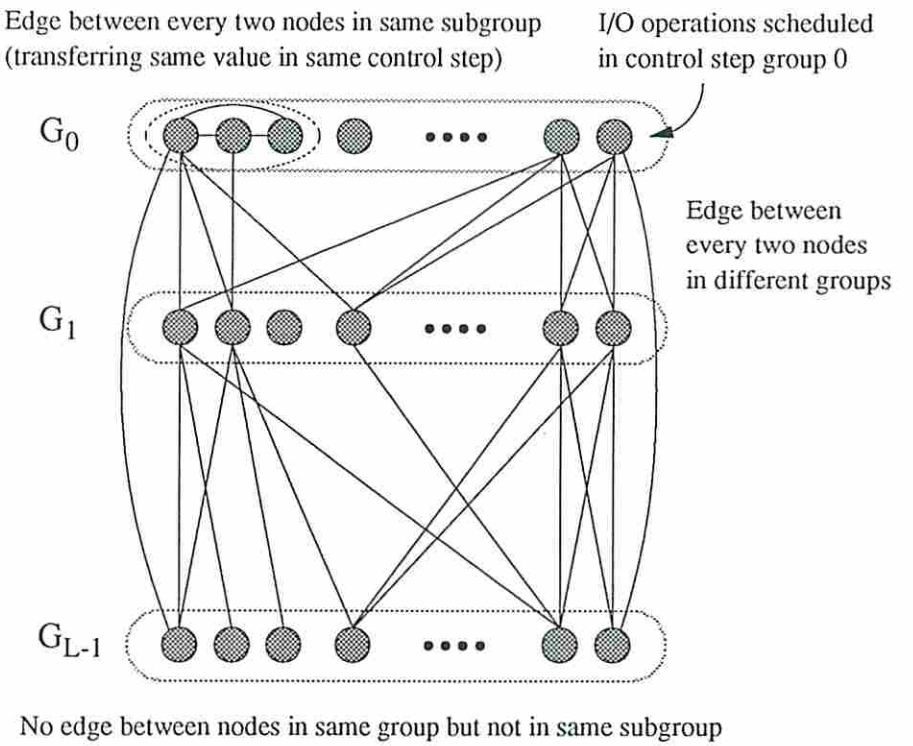


Figure 5.1: Compatible graph for interchip connection problem

By taking the advantage of the above properties, the heuristic procedure shown in Figure 5.2 has been developed. At first, the nodes corresponding to the I/O operations assigned in the same control step group are grouped. The nodes in each group are then divided into subgroups depending on whether the corresponding I/O operations transfer the same value in the same control step. The nodes in a subgroup are combined to form a supernode. A supernode represents a set of nodes which are in the same clique. Combining nodes $v_1, \ldots, v_k$ to form a supernode v means that nodes $v_1, \ldots, v_k$ are replaced by node v, all edges connected to any of $v_1, \ldots, v_k$ are removed, and an edge (v', v) is created for each node v' which is connected to all of $v_1, \ldots, v_k$. The weight, $weight(v', v)$, on the newly created edge is given by

$$weight(v', v) = \sum_{i=1}^{k} weight(v', v_i)$$

After these steps, there are only edges between nodes in different groups. Every clique will contain at most one node in each group since the nodes in a group conflict with each other. The graph induced from any two groups is bipartite. So, the cliques can be constructed by applying a series of bipartite weighted matchings to two groups giving priority to larger groups. After the matching between the two groups is found, a new group will be created by combining these two groups according the matching, and replaces them. Finally, only one group is left, and it will contain a number of supernodes, each of which represents a set of nodes in a clique. The I/O operations corresponding to the nodes in a clique are assigned to a communication bus. Then, the bus structure can be constructed according to the assignment of I/O operations to buses.

The most time-consuming step in the procedure is to solve the bipartite weighted matching problem by applying the Hungarian algorithm, which has a complexity of $O(n^3)$. So, the worse case complexity of the procedure is $O(Ln^3)$, where n is the total number of I/O operations. For the cases in which the I/O operations are evenly distributed among control step groups, the complexity will be $O(n^3/L^2)$.

Determine Interchip Connection
For each group G_k do
    Divide G_k into $SG_{k,1}, \ldots, SG_{k,n_k}$
Order G_i such that $n_i \geq n_j$ for $i < j$
For $i = 1$ to $L - 1$ do
    Apply Hungarian algorithm to G_0 and G_i
    For $j = 1$ to n_0 do
        $SG_{0,j} = SG_{0,j} \cup SG_{i,match(j)}$

Figure 5.2: A heuristic procedure for constructing interchip connection

5.3 Experimental Results

For comparison, the AR lattice filter is partitioned the same way as shown in Figure 4.7, and the same assumptions are made. Table 5.1 shows the numbers of I/O pins and operators required with variations of initiation rates and pipe lengths. For a given initiation rate, increasing the pipe length does not necessarily lead to a lower hardware requirement. For a given initiation rate, the design with least resource requirements is shown in bold font. The results produced by the previous approach described in Chapter 4 using different resource constraints are shown in Table 5.2. For all of the test cases, given the same interchip connections, better scheduling results than those shown can be obtained by postponing some of the operations as we have done here by constraining some of the operations and rerun the program. These results are shown in the parentheses at the output column.

The elliptic filter is also partitioned the same way as shown in Figure 4.20, and the same assumptions are made. Table 5.3 shows the numbers of I/O pins and operators required, and the input to output delay with variations of initiation rates and pipe lengths. The results produced by the previous approach are shown in Table 5.4. The previous approach can not produce any schedule for several designs with tight time and resource constraints even there exists a schedule. The reason is because of a small slack time in the critical loop and the greedy nature of the list scheduling technique implemented in our earlier approach. For most of

Inputs		Outputs									
Initiation Rate	Pipe Length	#I/O pins				#Adders			#Multipliers		
		P_0	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3
3	6	154	162	95	95	2	2	2	2	4	4
	7	170	90	130	114	2	2	2	2	3	2
	8	146	90	114	114	2	2	2	2	3	2
	9	**133**	**97**	**105**	**113**	**2**	**2**	**2**	**2**	**2**	**3**
	10	149	105	105	113	2	2	2	2	2	2
4	6	154	98	95	95	2	1	1	2	2	2
	7	154	106	114	98	2	2	2	2	2	2
	8	146	82	105	105	2	2	2	1	2	2
	9	146	98	95	95	2	2	1	1	2	2
	10	**138**	**98**	**95**	**95**	**2**	**1**	**2**	**1**	**2**	**2**
5	6	146	82	106	122	2	1	1	2	2	2
	7	**122**	**90**	**87**	**87**	**2**	**1**	**1**	**2**	**2**	**2**
	8	130	90	79	97	2	1	1	2	2	2
	9	154	90	97	113	2	2	1	2	2	2
	10	138	90	89	105	2	2	2	2	2	2

Table 5.1: Resources required for the AR filter with variations of initiation rates and pipe lengths using the technique described in this chapter

Inputs										Outputs
Initiation Rate	#I/O pins				#Adders			#Multipliers			Pipe Length
	P_0	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3	
3	109	97	87	87	2	2	2	2	2	2	11(9)
4	101	89	78	78	1	1	1	1	2	2	15(11)
	101	53	87	87	1	1	1	1	2	2	12(11)
5	77	81	70	70	1	1	1	1	2	2	10(9)
	85	53	79	79	1	1	1	1	2	2	13(10)

Table 5.2: Pipe length for the AR filter with variations of initiation rates and resource constraints using the technique described in Chapter 4

Inputs		Outputs											
Init.	Pipe	#I/O pins						(#Adders,#Multipliers)					In-Out
Rate	Length	P_0	P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5	Delay
	22	**32**	**48**	**48**	**32**	**32**	**32**	**(2,1)**	**(1,1)**	**(2,2)**	**(3,2)**	**(1,1)**	**21**
	23	32	48	48	32	32	32	(2,1)	(1,1)	(2,2)	(3,2)	(1,1)	23
5	24	32	64	48	32	32	32	(2,1)	(1,1)	(1,2)	(3,2)	(1,1)	23
	25	48	48	48	16	48	32	(2,1)	(1,1)	(1,2)	(3,2)	(1,2)	24
	26	48	64	32	32	48	16	(2,1)	(1,1)	(2,2)	(4,2)	(1,2)	23
	22	48	64	48	32	32	32	(2,1)	(1,1)	(1,1)	(2,2)	(1,1)	18
	23	16	48	32	32	16	48	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	18
6	**24**	**32**	**32**	**32**	**32**	**32**	**16**	**(2,1)**	**(1,1)**	**(1,1)**	**(2,1)**	**(1,1)**	**20**
	25	32	48	48	32	32	48	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	20
	26	32	48	32	16	48	32	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	22
	22	32	48	32	16	48	16	(2,1)	(2,1)	(1,1)	(2,2)	(1,1)	19
	23	32	48	32	16	48	32	(2,1)	(2,1)	(1,1)	(2,1)	(1,1)	20
7	24	16	48	32	16	32	48	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	20
	25	32	48	32	16	48	16	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	22
	26	**32**	**32**	**32**	**16**	**48**	**16**	**(2,1)**	**(1,1)**	**(1,1)**	**(2,1)**	**(1,1)**	**23**

Table 5.3: Resources required for the elliptic filter with variations of initiation rates and pipe lengths using the technique described in this chapter

Inputs												Outputs	
Init.	#I/O pins						(#Adders,#Multipliers)					Pipe	In-Out
Rate	P_0	P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5	Leng.	Delay
5	32	48	48	32	32	32	(2,1)	(1,1)	(2,2)	(3,2)	(1,1)	-(23)	-(22)
6	32	32	32	32	32	16	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	-(23)	-(19)
	32	32	32	32	32	32	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	24	18
7	32	32	32	16	48	16	(2,1)	(1,1)	(1,1)	(2,1)	(1,1)	28(24)	20(19)
	32	32	32	16	32	16	(1,1)	(1,1)	(1,1)	(2,1)	(1,1)	27(26)	20(19)

Table 5.4: Pipe length for the elliptic filter with variations of initiation rates and resource constraints using the technique described in Chapter 4

the test cases, given the same interchip connections, better scheduling results than those shown can be obtained by postponing some of the operations as we have done here by constraining some of the operations and rerun the program. These results are shown in the parentheses at the output column.

From the above results, the approach described in this chapter usually produces a design that requires more I/O pins because scheduling puts more constraints on the I/O pin optimization in the current approach. In addition, the distribution graph of I/O operations used in FDS, which is derived by combining the distribution graphs of the corresponding input and output operations, does not reflect the

usage of the communication bus accurately. On the other hand, due to interchip connection synthesis without considering the possible impact on scheduling and the greedy heuristics used in the list scheduling, the previous approach described in Chapter 4 usually produces a schedule with a longer input to output delay. For a design with tight time constraints, the previous approach may not be able to produce any schedule. The above conclusions are drawn based on the test cases, and may not be valid for some cases. However, we believe that they would be true in general.

In most of the test cases, better schedules can be obtained by postponing some of the operations. So, it is believed that a better scheduling result might have been obtained if a more advanced scheduling technique, such as PLS [HHL91], rather than list scheduling were used in the previous approach.

The I/O pin optimization described in this chapter tries to minimize the total number of I/O pins required. However, minimizing the total number of I/O pins used does not mean minimizing the manufacture cost because of the discrete number of I/O pins on a package. In conclusion, the approach described in the previous chapter with a more advanced scheduling technique will be more desirable than the approach described in this chapter.

Chapter 6

Sharing Buses in a Cycle

I/O pin resources can be utilized more effectively at the expense of more complicated control if we allow a communication bus to transfer more than one value at the same time by using different portions of a bus. In order to avoid the situation that each I/O pin would require an individual control signal, we assume that each communication bus is logically divided into a small number of sub-buses, each of which is composed of a number of bits. One or more consecutive sub-buses can be grouped and used to transfer a value. So, more than one value can be transferred on a communication bus at the same time by using different portions of the bus.

Like the problems described earlier, this problem is also approached in two steps: (1) interchip connection synthesis, and (2) scheduling. First, an ILP formulation for the subproblem of interchip connection synthesis is given. Due to the large amount of computation time required to solve the problem exactly, the heuristic search technique described in Chapter 4 is extended to cope with the extended problem. Then, a method of reassigning I/O operations to communication buses dynamically during scheduling is presented. Last, some experimental results are shown, and compared to the previous results for the restricted cases where only one value can be transferred on a communication bus in a single bus cycle.

Here, it is still assumed that each value must be transferred as a whole. That is, a value will not be divided into a number of sub-values, and then transferred in a number of control steps.

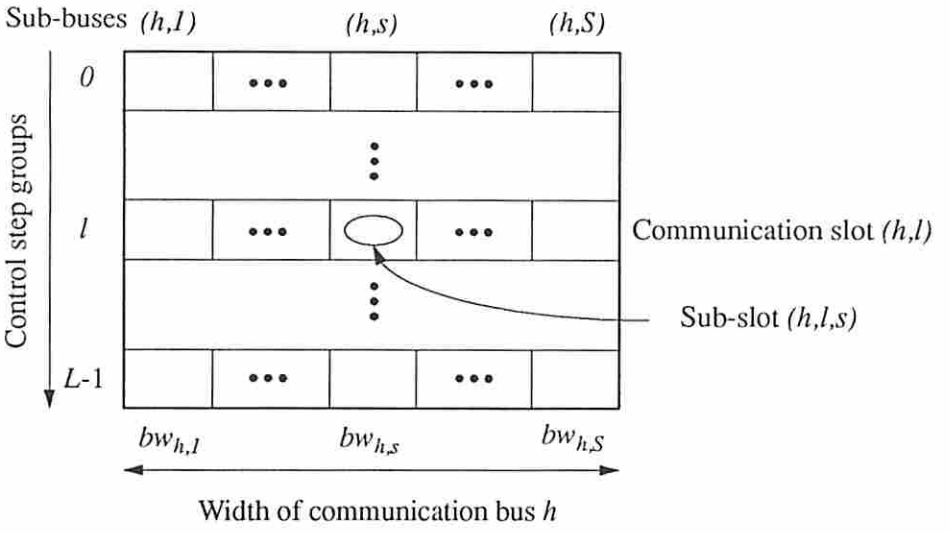


Figure 6.1: Illustration of communication slots, sub-buses, and sub-slots

6.1 Interchip Connection Synthesis

The same bus structure shown in Figure 4.6 will be used. For the sake of simpler control, a communication bus is logically divided into S sub-buses, each of which is composed of a number of bits, rather than into individual single-bit buses. Each communication bus consists of L communication slots, each of which corresponds to one communication bus usage in each control step group. Figure 6.1 illustrates the concept of communication slots, sub-buses, and sub-slots. The sub-bus (h, s) denotes the s'th sub-bus of the communication bus h. The communication slot (h, l) denotes the usage of the communication bus h in the control step group (bus cycle) l. The sub-slot (h, l, s) denotes the usage of the sub-bus (h, s) in the control step group l.

An I/O value transferred across chips using a partial or whole communication bus can be divided conceptually into S sub-components, each of which will be assigned to a sub-bus of a communication bus. Some of these sub-components can have zero bit widths, which means that the corresponding sub-buses are not assigned to the I/O transfer. Sub-components of a single value must be transferred

at the same time by using some contiguous sub-buses. That is, an I/O transfer must be assigned to one or more contiguous sub-slots.

6.1.1 ILP Formulation

The subproblem can be formulated as an ILP, which consists of three classes of constraints: (1) *assignment* constraints, (2) *data transfer* constraints, and (3) *resource* constraints. Each class of constraints will be described in Sections 6.1.1.1, 6.1.1.2, and 6.1.1.3. Some of constraints in the formulation are not in linear form. Linearization of these constraints will be discussed in Section 6.1.1.4.

The following notation will be used in the formulation.

- L : Initiation rate of the design.
- N : Number of partitions.
- R : Maximum number of communication buses.
- B_w : Bit width of I/O operation w.
- T_i : Total number of pins used for data transfers (pins used for power and control lines excluded) in partition P_i.
- $W_i = \{ w \mid$ every I/O operation w used to transfer a value to or from partition $P_i \}$.
- $W_v = \{ w \mid$ every I/O operation w used to transfer the same value $v \}$.
- $V = \{ v \mid$ every value v transferred across chips $\}$.
- $x_{w,h,l,s}$: A binary variable denoting a portion or the whole I/O operation w to be assigned to the sub-slot (h, l, s).
  $x_{w,h,l,s} = 1$ if a portion or the whole I/O operation w to be assigned to the sub-slot (h, l, s);
  $x_{w,h,l,s} = 0$ otherwise.

- $z_{w,h,l,s}$: An integer variable denoting the number of bits of I/O operation w to be assigned to the sub-slot (h, l, s).

- $bw_{h,s}$: An integer variable denoting the bit width of the sub-bus (h, s).

- $r_{i,h}$: An integer variable denoting the width of the I/O port of partition P_i connected to communication bus C_h.
  $r_{i,h} = 0$ if C_h is not connected to P_i.

6.1.1.1 Assignment Constraints

The class of assignment constraints states that every I/O operation must be assigned to contiguous sub-slots, and no two I/O operations can be assigned to the same sub-slot unless they transfer the same value.

$$\sum_{h=1}^{R} \sum_{l=0}^{L-1} \max_{s=1}^{S} x_{w,h,l,s} = 1 \qquad \forall w \in W \tag{6.1}$$

The above constraints state that every I/O operation must be assigned to some sub-slots of one and only one communication slot. $\max_s x_{w,h,l,s} = 1$ denotes that I/O operation w is assigned to some sub-slots of the communication slot (h, l).

An I/O operation can only be assigned to some contiguous sub-slots. In other words, the bit vector $< x_{w,h,l,1}, x_{w,h,l,2}, \ldots, x_{w,h,l,S} >$ can only contain at most one sequence of 1's. That is, there can not be more than two $0 \rightarrow 1$ or $1 \rightarrow 0$ transitions in the bit vector $< 0, x_{w,h,l,1}, x_{w,h,l,2}, \ldots, x_{w,h,l,S}, 0 >$. The beginning and ending zeroes are padded to exclude the illegal bit vectors with patterns of $< 1...10...01...1 >$, in which there are only two $0 \rightarrow 1$ or $1 \rightarrow 0$ transitions. A $0 \rightarrow 1$ or $1 \rightarrow 0$ transition between two consecutive bits can be detected by exclusive-ORing these two bits. The technique of using exclusive-OR to detect transitions in a bit stream has been widely used in the field of testing. So, the following

constraints are satisfied if and only if the bit vector $< x_{w,h,l,1}, x_{w,h,l,2}, \ldots, x_{w,h,l,S} >$ contains at most one sequence of 1's.

$$x_{w,h,l,1} + \sum_{s=2}^{S}(x_{w,h,l,s-1} \oplus x_{w,h,l,s}) + x_{w,h,l,S} \leq 2 \qquad \forall w, h, l \tag{6.2}$$

The following constraints state that no more than one I/O operation can be assigned to the same sub-slot.

$$\sum_{\forall w} x_{w,h,l,s} \leq 1 \qquad \forall h, l, s \tag{6.3}$$

However, I/O operations transferring the same value can be assigned to the same sub-slots. In this case, Constraint 6.3 must be replaced by the following constraints.

$$\sum_{\forall v} \max_{w \in W_v} x_{w,h,l,s} \leq 1 \qquad \forall h, l, s \tag{6.4}$$

However, if any sub-slot is assigned to two or more I/O operations transferring the same value, all of these I/O operations must be assigned to the same sub-slot(s). In other words, if the bitwise AND of the bit vectors $< x_{w,h,l,1}, \ldots, x_{w,h,l,S} >$ and $< x_{w',h,l,1}, \ldots, x_{w',h,l,S} >$, where w and w' transfer the same value, is not all zeroes, these two bit vectors must be identical. The integer variable $ov_{w,w',h,l}$ defined as follows can be used to check if any sub-slot of the communication slot (h, l) is shared by the I/O operations w, and w', transferring the same value.

$$ov_{w,w',h,l} = \max_{s}(x_{w,h,l,s} + x_{w',h,l,s}) \qquad \forall v, h, l \; \forall w, w' \in W_v,$$

where + indicates arithmetic add. $x_{w,h,l,s} + x_{w',h,l,s}$ can be 0, 1, or 2, since $x_{w,h,l,s}$ and $x_{w',h,l,s}$ can be either 0 or 1. So, $ov_{w,w',h,l}$ can have a value of 0, 1, or 2, and will have a value of 2 if the bitwise AND of these two bit vectors is not all zeroes.

The following constraints are used to force these two bit vectors to be identical if their bitwise AND is not all zeroes.

$$(ov_{w,w',h,l} \geq 2) \Rightarrow (\sum_{s}(x_{w,h,l,s} \oplus x_{w',h,l,s}) = 0) \qquad \forall v, h, l \; \forall w, w' \in W_v \qquad (6.5)$$

6.1.1.2 Data Transfer Constraints

The class of data transfer constraints states that the sub-buses to which an I/O operation is assigned must provide a communication path wide enough for the I/O operation.

There is a relation between $x_{w,h,l,s}$ and $z_{w,h,l,s}$. That is, the number of bits of a value to be transferred on a sub-slot can be greater than zero if and only if the sub-slot is assigned to the I/O operation.

$$(z_{w,h,l,s} > 0) \Leftrightarrow (x_{w,h,l,s} = 1) \qquad \forall w, h, l, s \qquad (6.6)$$

The number of bits to be transferred on a sub-bus can not exceed the bit width of the sub-bus.

$$bw_{h,s} \geq \max_{\forall w,l} z_{w,h,l,s} \qquad \forall h, s \qquad (6.7)$$

For every I/O operation, the total number of bits to be transferred must be equal to the bit width of the I/O operation.

$$\sum_{\forall h,l,s} z_{w,h,l,s} = B_w \qquad \forall w \qquad (6.8)$$

Define

$$a_{i,h,s} = \max_{\forall w \in W_i, \forall l} z_{w,h,l,s},$$

which is the maximum number of bits of I/O operations over all bus cycles in partition P_i assigned to the sub-bus (h, s). $a_{i,h,s}$ is also the minimum number of I/O pins of partition P_i connected to the sub-bus (h, s). The following constraints state that the minimum number of I/O pins connected to a communication bus

must provide a communication path for I/O operations assigned to the bus based on the idea that if a partition has I/O pins connected to some sub-bus all previous sub-buses must be connected to the partition.

$$(a_{i,h,s} > 0) \Rightarrow r_{i,h} \geq \sum_{t=1}^{s-1} bw_{h,t} + a_{i,h,s} \qquad \forall i, h, s \tag{6.9}$$

6.1.1.3 Resource Constraints

The resource constraints ensure that for each partition the total number of pins required cannot exceed the total number of pins available.

$$\sum_{h=1}^{R} r_{i,h} \leq T_i; \qquad \text{for } 0 \leq i \leq N \tag{6.10}$$

6.1.1.4 Linearization

In the above formulation, some of the constraints are not in linear form. Here, linearization of these constraints will be discussed. The following notation will be used in the discussion.

- B_x: a binary variable.
- I_x: an integer variable, or an expression having integer value.
- C: an variable having a value of 0, 1, or 2.
- M: a very large value.

A constraint with a maximum function of the form

$$B_x \geq \max_{i=1}^{n} B_i$$

can be linearized and replaced by the following constraints

$$B_x \geq B_i; \qquad \text{for } 1 \leq i \leq n.$$

A constraint with a maximum function of the form

$$B_x = \max_{i=1}^{n} B_i$$

can be replaced by the following constraints

$$\begin{aligned} B_x &\geq \max_{i=1}^{n} B_i, \text{ and} \\ B_x &\leq \sum_{i=1}^{n} B_i. \end{aligned}$$

The second inequality is used to force B_x to be zero if all of B_i are zeroes.

A constraint with a minimum function of the form

$$B_x \leq \min_{i=1}^{n} B_i$$

can be linearized and replaced by the following constraints

$$B_x \leq B_i; \qquad \text{for } 1 \leq i \leq n.$$

A constraint with a minimum function of the form

$$B_x = \min_{i=1}^{n} B_i$$

can be replaced by the following constraints

$$\begin{aligned} B_x &\leq \min_{i=1}^{n} B_i, \text{ and} \\ B_x &\geq \sum_{i=1}^{n} B_i - (n-1). \end{aligned}$$

The second inequality is used to force B_x to be one if all of B_i are ones.

A constraint with exclusive-OR function of the form

$$B_z = B_x \oplus B_y$$

can be replaced by

$$B_z = \max(B_x, B_y) - \min(B_x, B_y).$$

Constraint 6.5 has the form

$$(C \geq 2) \Rightarrow (I_x = 0),$$

which can be replaced by the following constraint

$$(2 - C)M \geq I_x.$$

For the cases where $C \leq 1$, the above constraint will be satisfied automatically since the left hand side will have a large value. For the cases that $C = 2$, the left hand side of the above constraint will become zero, and I_x will be forced to zeroes.

Constraint 6.6 has the form

$$(I_x > 0) \Leftrightarrow (B_x = 1),$$

which can be replaced by the following constraints.

$$\begin{aligned} I_x &\leq MB_x, \text{ and} \\ I_x &\geq B_x. \end{aligned}$$

The first inequality implies that B_x must be one if I_x is greater than zero. The second inequality implies that I_x must be greater than zero if B_x is equal to one.

Constraint 6.9 has the form

$$(I_z > 0) \Rightarrow (I_x \geq I_y),$$

which can be replaced by

$$(I_z > 0) \quad \Leftrightarrow \quad (B_z = 1), \text{ and}$$

$$(B_z = 1) \Rightarrow (I_x \geq I_y).$$

Then, the second inequality can be replaced by

$$I_x \geq I_y - (1 - B_z)M.$$

For the cases where $B_z = 0$, the above inequality will be satisfied automatically since the right hand side will have a negative value. For the cases that $B_z = 1$, the above inequality will be reduced to $I_x \geq I_y$.

6.1.2 Heuristic Search

Since the size of the formulation is even larger than that of the formulation of the restrictive cases described in Chapter 4, the heuristic search technique described in the previous chapter has been extended to handle this case.

To reduce the search space and also the computation time, for our prototype program we assume that a communication bus can be divided into at most two sub-buses. So, an I/O operation can be assigned to the first sub-bus, second sub-bus, or the whole bus. The same cost function described previously will be used. At each stage of selecting the communication buses with least costs for an I/O operation, each unsplit communication bus is tentatively split into two sub-buses if the second sub-bus is wide enough for the I/O operation while at least one of I/O operations previously assigned to the bus can be fit in the first sub-bus after the bus is split. That is, a communication bus will be split only if the width of the bus is greater than or equal to the sum of the bit width of the I/O operation under consideration and the bit width of one of the previously assigned I/O operations. The search space is also restricted by not allowing extension of the width of a communication bus to make two I/O operations able to share a communication slot. In other words, the width of a communication bus will be determined and set to the largest bit width of the I/O operations assigned to the bus.

6.2 Reassignment of I/O Operations to Buses

For the simplified cases where no communication buses are split, a successful reassignment of I/O operation to communication bus is composed of a sequence of preemption operations, since an I/O operation can preempt at most one I/O operation. For the cases where some of the communication buses are split into two sub-buses, an I/O operation may require one or both components of a communication slot. So, during the preemption procedure, one I/O operation could preempt two I/O operations, each of which could, in turn, preempt two I/O operations. In addition, the I/O operation preemption could be done only if both of the preempted I/O operations could successfully preempt other I/O operations. This would require backtracking. To reduce the search space and the computation time, we only allow one I/O operation to be preempted by another I/O operation, and prune the branches which require two I/O operations to be preempted at the same time. Then, the procedure is similar to the one of finding an augmentation path in the bipartite matching problem, the same as the simplified cases. What we face here it is that the answer may be "NO" even if the reassignment of an I/O operation is actually possible. However, this does not seem to prune better solutions in most cases.

6.3 Experimental Results

For comparison, the same partitioned AR lattice filter in Figure 4.7 is used, and the same assumptions are made. The interchip connections and schedules for the partitioned AR filter with different initiation rate are shown in Figures 6.2 – 6.7, assuming that all I/O ports can be bidirectional and two values can be transferred on a bus in a control step. The initial and final bus assignments are shown in Tables 6.1 – 6.3. For the designs with initiation rates of 3 and 4, the communication bus C_1 is split into two sub-buses, one with 8 bits and the other with 29 bits. For the design with an initiation rates of 5, in addition to the communication bus C_1

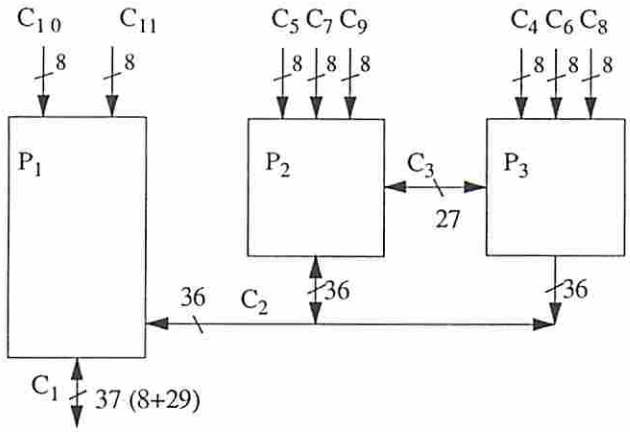


Figure 6.2: Interchip connection for the AR filter with an initiation rate of 3

being split into two sub-buses, the communication bus C_2 is also split into two sub-buses, each with 18 bits. The comparison among different bus assumptions is shown in Table 6.4. As we might expect, a smaller number of I/O pins are required if two values are allowed to be transferred on a communication bus at the same time. Because of the smaller numbers of I/O pins and communication buses used, the length of the pipeline may be longer as I/O operations compete for communication buses. One thing which needs to be mentioned is that the longer pipe length (15 stages) for the design in which two values are allowed to be transferred on a communication bus at the same time is not mainly caused by the smaller number of communication buses, but rather by the greedy heuristic used in list scheduling. A better scheduling result might have been obtained if a more advanced scheduling technique had been used. If +3 were postponed to control step 4, +11 and O1 would have been able to be scheduled in control steps 8 and 9, respectively. So, the length of the pipeline could be reduced to 10 stages, which is the same as the earlier design which did not share the bus cycles.

Designs with fewer I/O pins would be obtained if a communication bus could be split into more than two sub-buses. For the test case with an initiation rate of 4, 8 I/O pins of partition P_1 would have been saved by eliminating communication

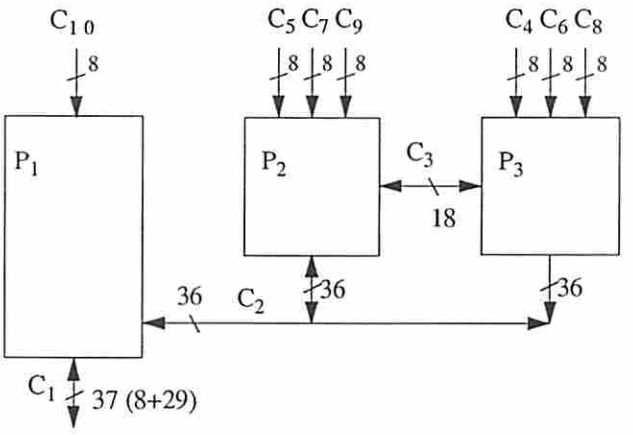


Figure 6.3: Interchip connection for the AR filter with an initiation rate of 4

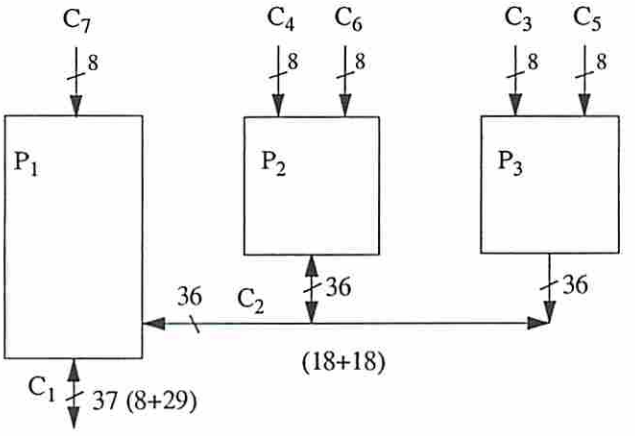


Figure 6.4: Interchip connection for the AR filter with an initiation rate of 5

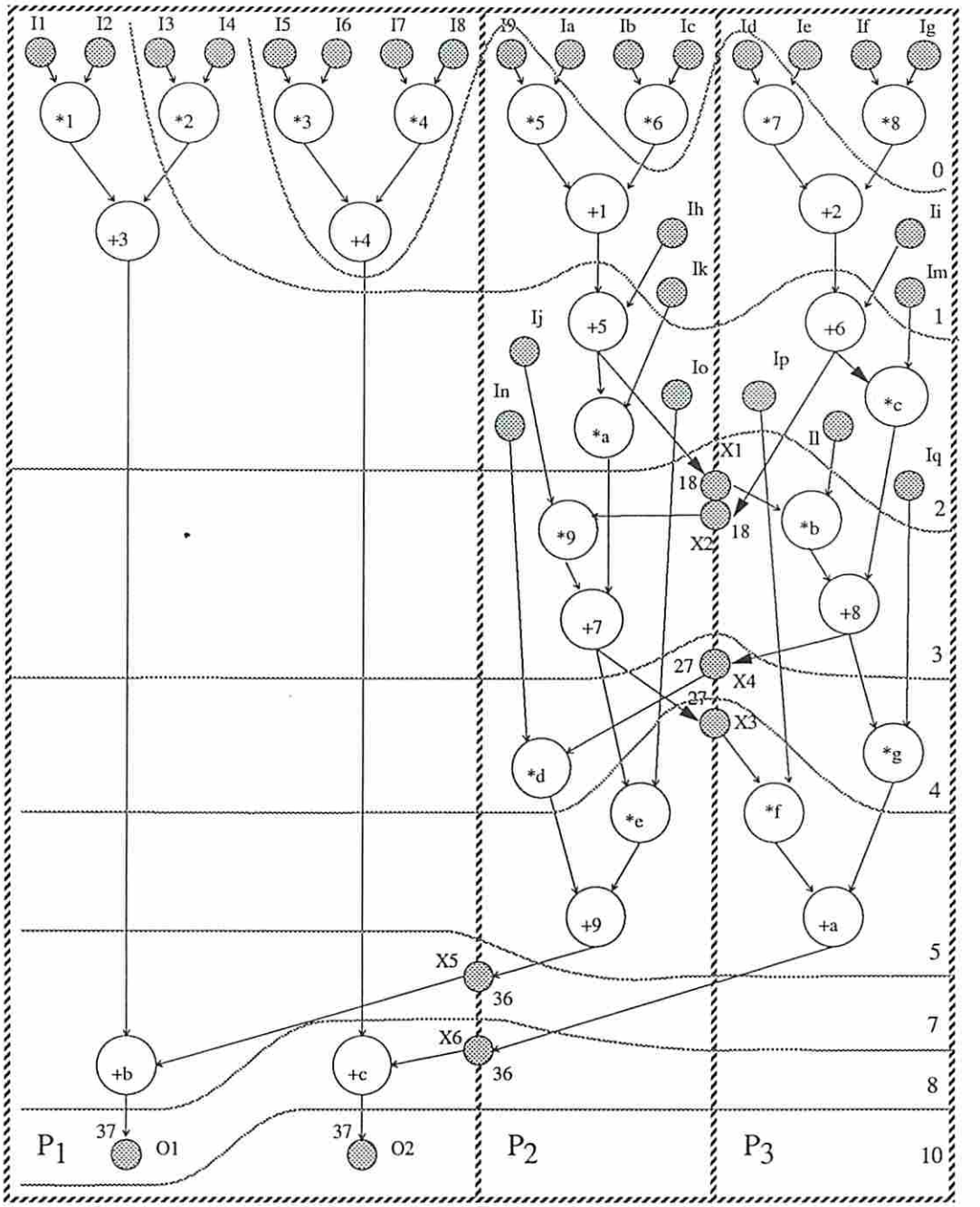


Figure 6.5: Schedule for the AR filter with an initiation rate of 3

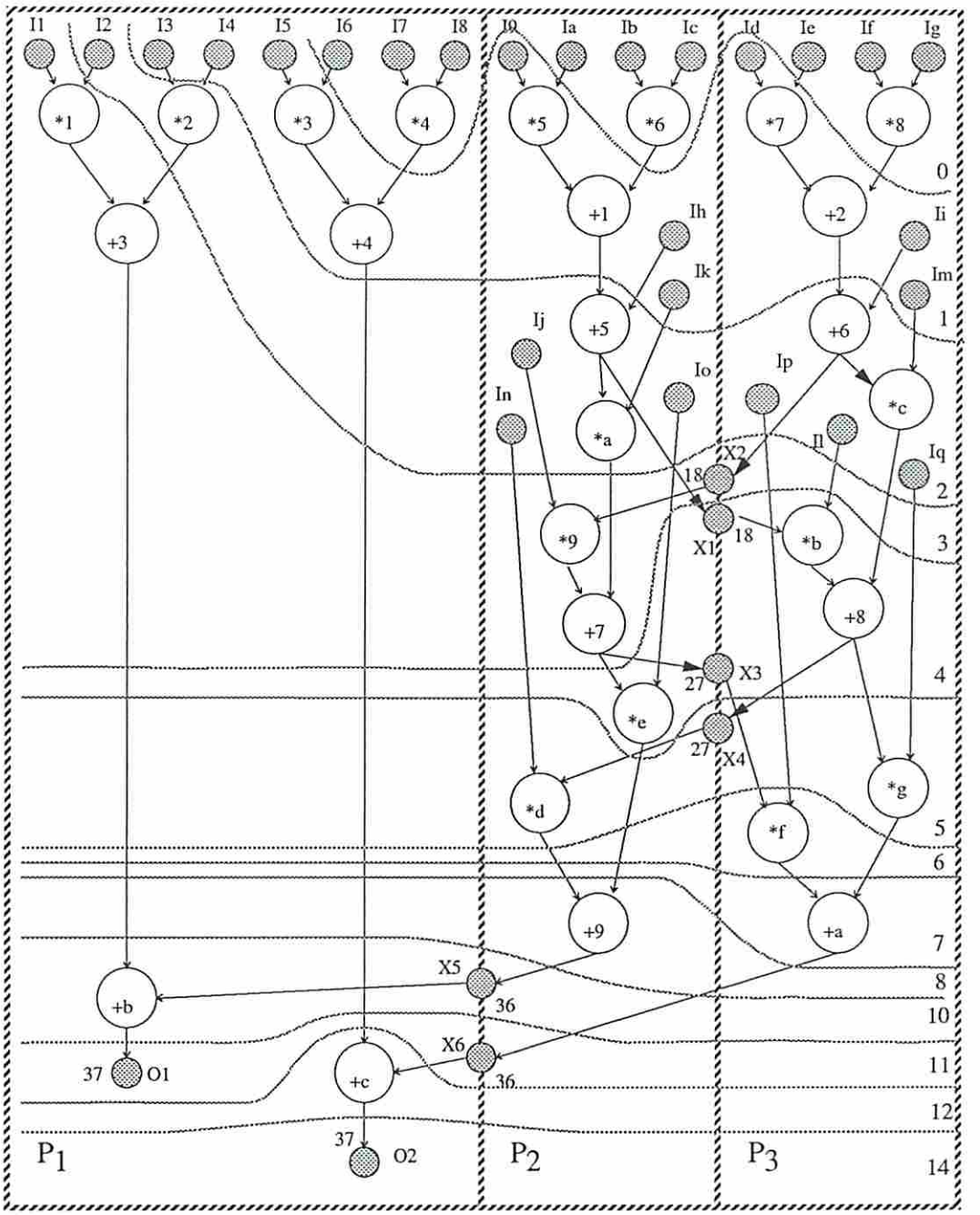


Figure 6.6: Schedule for the AR filter with an initiation rate of 4

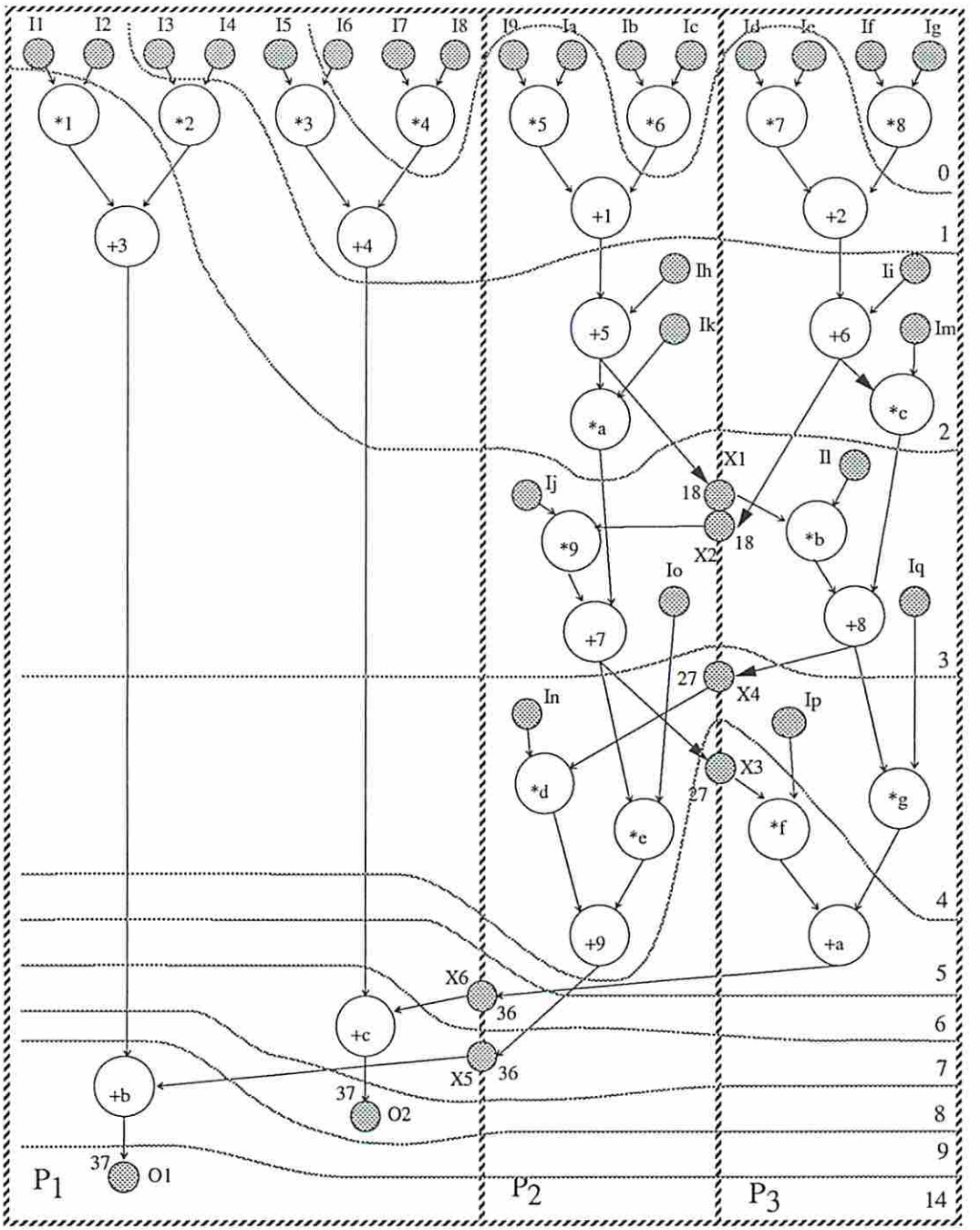


Figure 6.7: Schedule for the AR filter with an initiation rate of 5

Communication Bus	Initial Assignment			Final Assignment		
C_1'	O1	O2	I8	O1	O2	I8
C_1''			I7			I7
C_2	X3	X5	X6	X1	X5	X6
C_3	X1	X2	X4	X2	X3	X4
C_4	Im	Ip	Iq	Ie	Im	Ip
C_5	Ik	In	Io	Ia	Ik	In
C_6	Ig	Ii	Il	Ig	Ii	Il
C_7	Ic	Ih	Ij	Ic	Ih	Ij
C_8	Id	Ie	If	Id	If	Iq
C_9	I9	Ia	Ib	I9	Ib	Io
C_{10}	I4	I5	I6	I2	I4	I6
C_{11}	I1	I2	I3	I1	I3	I5

Table 6.1: I/O operation to bus assignment with an initiation of 3

Communication Bus	Initial Assignment				Final Assignment			
C_1'	O1	O2	I5	I8	O1	O2	I5	I8
C_1''			I6	I7			I4	I7
C_2	X3	X4	X5	X6	X3	X4	X5	X6
C_3	X1	X2			X1	X2		
C_4	Il	Im	Ip	Iq	Ie	Il	Im	
C_5	Ij	Ik	In	Io	Ia	Ij	Ik	
C_6	Ie	If	Ig	Ii	Ig	Ii	Ip	
C_7	Ia	Ib	Ic	Ih	Ic	Ih	In	
C_8	Id				Id	If	Iq	
C_9	I9				I9	Ib	Io	
C_{10}	I1	I2	I3	I4	I1	I2	I3	I6

Table 6.2: I/O operation to bus assignment with an initiation of 4

Communication Bus	Initial Assignment					Final Assignment				
C_1'	O1	O2	I3	I5	I8	O1	O2	I5	I8	
C_1''			I4	I6	I7			I1	I4	I7
C_2'	X3	X4	X5	X6	X1	X3	X4	X5	X6	X1
C_2''					X2					X2
C_3	Ii	Il	Im	Ip	Iq	Id	If	Ii	Il	
C_4	Ih	Ij	Ik	In	Io	I9	Ib	Ih	Ij	
C_5	Id	Ie	If	Ig		Ie	Ig	Im	Ip	Iq
C_6	I9	Ia	Ib	Ic		Ia	Ic	Ik	In	Io
C_7	I1	I2				I2	I3	I6		

Table 6.3: I/O operation to bus assignment with an initiation of 5

Initiation Rate	Bidirectional (No sharing)				Bidirectional (Sharing)			
	PinReq			PipeLen	PinReq			PipeLen
3	97	87	87	11	89	87	87	11
4	89	78	78	15	81	78	78	15
5	81	70	70	10	81	52	52	15

Table 6.4: Comparisons of number of pins required and pipe length for different assumptions

bus C_{10} (in Figure 6.3), if bus C_1 had been split into 4 sub-buses, each of which has at least 8 bits. Bus C_1 could be used to transfer $I1, I2, I3$, and $I4$ in a same cycle, $I5, I6, I7$, and $I8$ in another cycle, and $O1$ and $O2$ in the other two cycles.

Chapter 7

Other Extensions

In this chapter, some extensions to our research are discussed. First, modeling of data recursive edges and problems arising due to data recursive edges are presented. Some I/O operations will never be executed in the same execution instance since they are executed conditionally, if conditional branches exist across multiple partitions. Conditional sharing among these I/O operations is discussed in the next section. Then, modeling of time division I/O multiplexing is given. Last, handling a CDFG containing multiple-cycle operations is presented.

7.1 Data Recursive Edges

In general, an implicit outermost control loop is assumed on a CDFG. That is, the CDFG is performed repeatedly, each time for a new set of data. Each iteration of the CDFG is referred to as an *execution instance*. An operation in an execution instance may require a value generated by an operation in the current or a previous execution instance. This data dependency can be represented by an edge associated with a *degree d* in the CDFG, where d means that the value used by an operation is generated d iterations earlier. An edge with a degree 0 represents data dependency in the same execution instance. We refer to an edge associated with a degree greater than zero as a *data recursive edge*.[1] For the partial CDFG shown in Figure 7.1, an

[1]This is equivalent to subscripted values in the DDS [KP85], with x_{i-1} being of degree 1. Such information is explicit in signal flow graphs, where delays are shown.

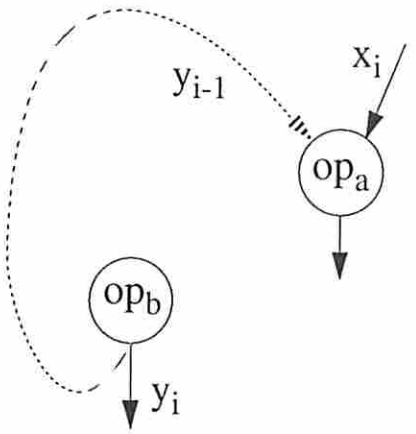


Figure 7.1: A partial CDFG with a data recursive edge

input value to operation op_a comes from the output of operation op_b of the previous execution instance. This data dependency is denoted by an edge (indicated by a dashed line) from op_b to op_a with degree 1. No input operation is required if an input value to an operation is produced by an operation of a previous execution instance in the same partition. The value produced in the previous execution instance can be stored for later use.

A maximum time constraint will be imposed on the two operations which have a data recursive edge connected between them, because a value must have been produced before it can be consumed. Assume that the initiation rate is L, and op_a and op_b of execution instance i are scheduled in time steps t_a and t_b, respectively. Then, op_b of execution instance $i-1$ is performed in time step $t_b - L$. Shown in Figure 7.2 are two execution instances of the CDFG. In a valid schedule, op_b of execution instance $i-1$ must be completed before op_a of execution instance i can be started. That is, $t_b - L < t_a$, or equivalently $t_b - t_a < L$, which can be viewed as a maximum time constraint between operations op_a and op_b. In general, for a data recursive edge with degree d, the maximum time constraint will be $t_b - t_a < dL - (c_b - 1)$ if op_b takes c_b cycles to execute, and t_b and t_a indicate the time steps the operations begin execution.

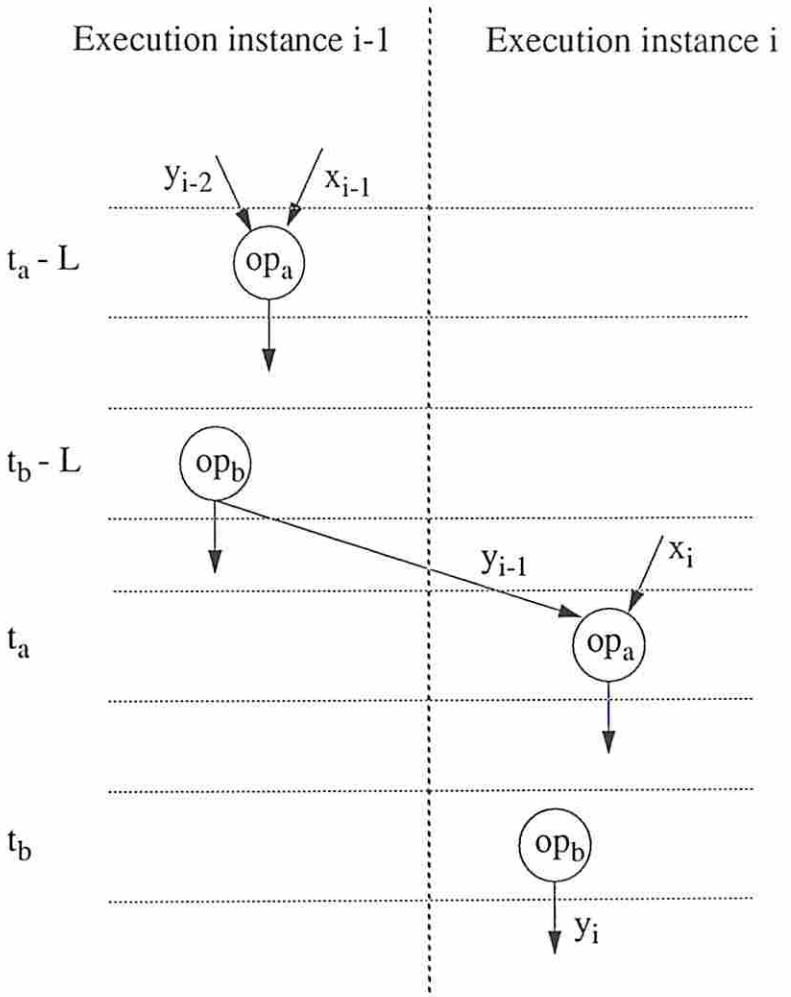


Figure 7.2: An example schedule of two execution instances

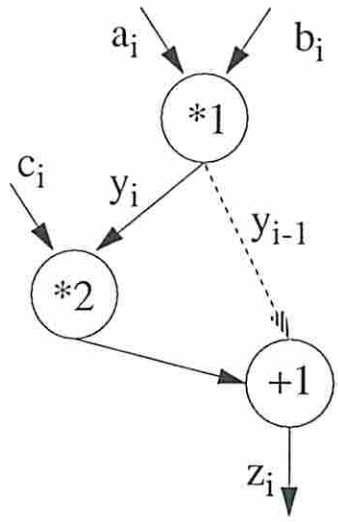
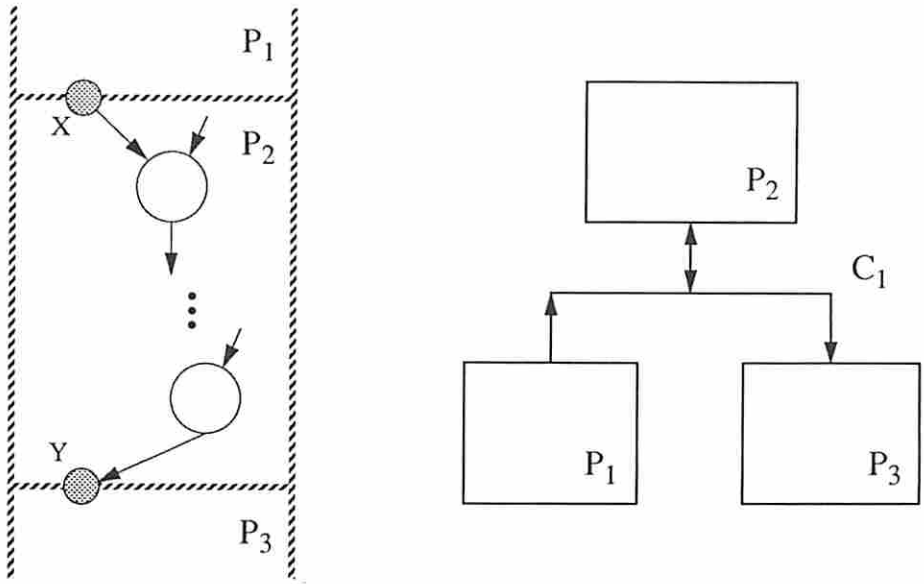


Figure 7.3: The CDFG representation for Expression (7.1)

Some researchers [GVM89, HHL91] implicitly assume that there will be d registers on the data transfer path corresponding to a data recursive edge with a degree d. So, a precedence edge from op_a to op_b must be added to prevent op_b from being scheduled before op_a to avoid an over-writing hazard where the new value y_i generated by op_b is written to the register before the old one y_{i-1} is used by op_a. Such a precedence edge would be inappropriate for some cases where the old value y_{i-1} is required after the new version y_i is used. This can be shown by the following simple example:

$$\begin{aligned} y_i &= a_i * b_i; \\ z_i &= c_i * y_i + y_{i-1}. \end{aligned} \tag{7.1}$$

The CDFG representation for the above expression is shown in Figure 7.3. If an edge from operation +1 to operation *1 is added, there will be a cyclic dependency among operations *1, *2, and +1. So, it will be impossible to schedule the CDFG unless some dataflow graph transformations are applied to the CDFG, such as splitting operation *1 into two operations, one for y_i, and the other for y_{i-1}. In



(a) A partial CDFG

(b) A possible interchip connection

Figure 7.4: An example showing no feasible scheduling

our approach, no precedence edge from op_a to op_b will be added for a data recursive edge from op_b to op_a. In this case, we may require more than d registers on the corresponding data transfer path.

For a CDFG containing maximum time constraints, there might exist no schedule if the interchip connections are synthesized without envisioning the effect of the maximum time constraints. This can be shown by the example CDFG in Figure 7.4(a). Figure 7.4(b) shows one possible interchip connection, where C_1 is the only communication bus on which I/O operations X and Y can transfer values. Suppose that I/O operations X and Y can only be scheduled in the same control step group due to the maximum time constraint and the minimum time constraint between them. The minimum time constraint is due to the precedence dependency among the operations between X and Y and resource constraints for these operations. There exists no schedule for the example given the interchip connection because I/O operations X and Y have to execute in the same control step group

and cannot share the same communication bus, while C_1 is the only bus capable of transferring values for X and Y.

One might attempt to develop a technique to predict if any two I/O operations can be assigned to the same communication bus without excluding all feasible pipelined schedules for a CDFG containing a data recursive edge during interchip connection synthesis. Unfortunately, the problem of determining whether assigning any two I/O operations to the same communication bus will result in no pipelined schedule for a CDFG containing a data recursive edge is NP-complete. This can be proven by polynomial-transforming a known NP-complete problem, precedent constrained scheduling, to the problem.

Definition 7.1 *Precedent Constrained Scheduling (PCS)* [GJ79]
Given a set of tasks $T = \{T_1, T_2, \ldots, T_n\}$, each taking 1 time unit, M processors, a partial order P on T, and a deadline D. Is there an M-processor schedule for T that meets the overall deadline D and obeys the precedence constraints?

Theorem 7.1 Given a partitioned CDFG which contains data recursive edges, resource constraints for each partition, a partial interchip connection, an assignment of any two I/O operations to the same communication bus.
The problem (ASG) of determining if there exists a multi-chip pipelined schedule with a bus assignment which meets the given partial bus assignment is NP-complete.

Proof: (a) $ASG \in NP$ since it can be checked in polynomial time whether a schedule satisfies all given constraints.

(b) We transform an instance $p \in PCS$ to an instance $q \in ASG$.
Let $T = \{T_1, T_2, \ldots, T_n\}$ be a set of tasks, P be a partial order, M be the number of processors, and D be a deadline in p.
We construct an instance $q \in ASG$, as shown in Figure 7.5, in which the numbers shown on the left are time steps for one possible schedule. The instance is pipelined between partitions but within partition P_2, execution is not pipelined, and contains

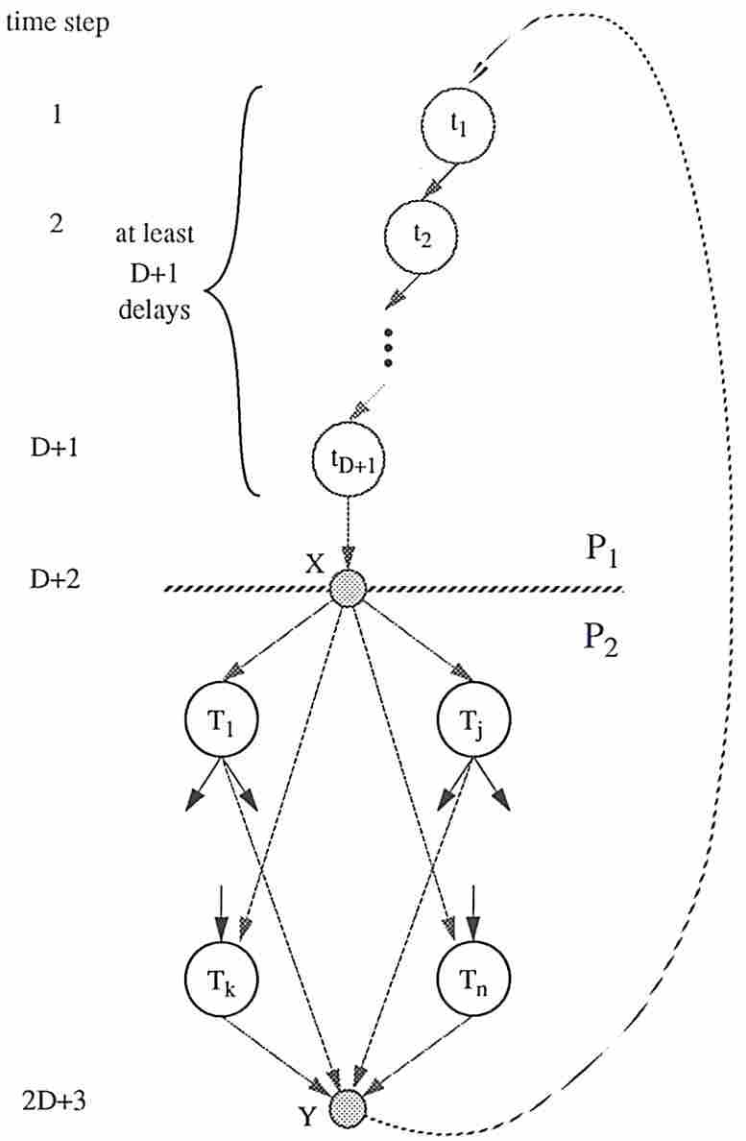


Figure 7.5: An instance in ASG transformed from an instance in PCS

- a set of operations

  $V = V_1 \cup V_2 \cup IO$,

  where $V_1 = \{t_1, \ldots, t_{D+1}\}$, a set of operations in partition P_1,

  $V_2 = T$, a set of operations in partition P_2, and

  $IO = \{X, Y\}$, a set of I/O operations, where X is an I/O operation feeding value forward P_2 and Y is an I/O operation for data recursive back to P_1;

  All operations in V take one control step to execute;

- a set of precedence edges

  $E = \{(Y, t_1), (t_1, t_2), \ldots, (t_D, t_{D+1}), (t_{D+1}, X)\} \cup E_X \cup P \cup E_Y$,

  where (Y, t_1) is a data recursive edge with degree 2,

  $E_X = \{(X, T_1), \ldots, (X, T_n)\}$, a set of edges from X to all nodes in V_2, and

  $E_Y = \{(T_1, Y), \ldots, (T_n, Y)\}$, a set of edges from all nodes in V_2 to Y;

- 1 operator for P_1;

- M operators for P_2;

- an initiation rate $L = D + 2$; and

- X and Y being assigned to the same communication bus.

Obviously, the above transformation can be done in polynomial time. Now, we want to prove that p is satisfied if and only if q is satisfied.

(i) Suppose that q is satisfied. Then, there exists a schedule for q, in which t_1, X, and Y are scheduled in control steps s_t, s_X, and s_Y, respectively, such that

$$s_Y - s_X \neq L, \tag{7.2}$$

$$s_X - s_t \geq D + 1, \text{ and} \tag{7.3}$$

$$s_Y - s_t \leq 2L - 1. \tag{7.4}$$

The inequality 7.2 holds because of the conflict-free constraint on the communication bus to which X and Y are assigned. The inequality 7.3 holds because of the precedence constraints through $t_1, t_2, \ldots, t_{D+1}, X$. The inequality 7.4 holds because of the maximum time constraints imposed by the data recursive edge. From the above inequalities, we have $s_Y - s_X \leq D + 1$. Hence, the maximum number of time steps allocated for T in the schedule is $(s_Y - 1) - (s_X + 1) + 1 \leq D$. So, p is satisfied.

(ii) Suppose that p is satisfied. Then, there is a schedule for p with a deadline D. Let S_k be the time step in which T_k is executed in the schedule, and $1 \leq S_k \leq D$. So, we can let V of q be scheduled as follows:

- $t_1, \ldots, t_{D+1}$ are scheduled in time steps $s_X - (D+1), \ldots, s_X - 1$;
- X is scheduled in time step s_X;
- $T_k \in V_2$ is scheduled in time step $s_X + S_k$; and
- Y is scheduled in the time step $s_X + D + 1$.

The above schedule for q satisfies the precedence constraints and resource constraints. The conflict-free constraint on the communication bus is also satisfied since I/O operations X and Y are not scheduled in the same control step group. $((s_X + D + 1) - s_X = D + 1$ is not a multiple of $L = D + 2.)$ The maximum time constraint imposed by the data recursive edge is also satisfied. $((s_X + D + 1) - (s_X - (D + 1)) = 2(D + 1) < 2L.)$ So, q is satisfied.

ASG is NP-complete since $ASG \in NP$, and PCS is polynomially transformable to ASG. □

From the above theorem, we know that synthesizing an interchip connection which guarantees a pipelined schedule exists for a partitioned CDFG containing data recursive edges is an intractable problem. However, a technique which can give a good approximate solution is still desirable.

7.2 Conditional I/O Operations

The behavioral description of a digital system usually contains conditional branches. For such a system, only the operations of selected branches will be executed in an execution instance. The operations on different conditional branches never executed at the same time in an execution instance can share hardware resources. In conditional resource sharing, a hardware module is shared among operations of mutually exclusive conditional branches executed in the same control step. Conditional resource sharing is not applied between different execution instances since execution of selected conditional branches can hardly be predicted in advance, as stated earlier by Park [PP88].

For a partitioned CDFG, in which a whole conditional block is put in a partition, all I/O operations will be executed in every execution instance. So, no conditional sharing among I/O operations is possible. In some cases, however, a conditional block can be too large to fit into a single partition because of area constraints. In these cases, a conditional block must be partitioned into several partitions, and only some of the I/O operations will be executed in an execution instance. In this case, conditional resource sharing should also be applied to I/O operations to achieve a better utilization of I/O pin resources. Conditional sharing on functional units have been reported by several researchers [PP88, HCDdA88, WY89]. Usually, mutually exclusive operations are identified before the task of conditional sharing begins. The coloring algorithm given by Park et al. [PP88] and the condition vector (CV) defined by Wakabayashi et al. [WY89] have been used to determine mutual-exclusiveness between functional operations. These techniques can also be used to determine mutual-exclusiveness between I/O operations. In some approaches [PP88, WY89], conditional sharing is performed during scheduling, while in Hwang's approach [HCDdA88], conditional sharing is predetermined before scheduling. In this section, the discussion will concentrate on conditional sharing among I/O operations, assuming that the mutual-exclusiveness between I/O operations has already been determined by one of these techniques.

Conditional resource sharing among I/O operations is performed before inter-chip connection synthesis. A compatibility graph is used in the process of conditional resource sharing. In the compatibility graph, a node represents a set of mutually exclusive I/O operations which are to share a communication slot. Two nodes are said to be compatible if and only if the I/O operations in the corresponding sets are all mutually exclusive and can be scheduled in the same control step in order to share a communication slot. Compatible nodes are connected by edges. The conditional sharing process consists of a series steps combining two compatible nodes. The pair of nodes with the most promising "benefit" is combined in each iteration. Combining two compatible nodes means that the I/O operations in the corresponding sets are to be scheduled in the same control step to share a communication slot.

Each node is associated with two attributes:

1. a time frame,[2] $frame = \{asap, asap + 1, \ldots, alap\}$,
   which is a range of control steps in which the I/O operations in the corresponding set can be scheduled, assuming that these I/O operations are scheduled in the same control step in order to share a communication slot; and

2. a bus connection structure, $\mathbf{r} = (r_0, r_1, \ldots, r_N)$,
   where r_i is the width of the I/O port of partition P_i connected to the bus, which is a communication bus to which the least number of I/O pins are connected such that the bus can still be used by these I/O operations, where $\max_{i=0}^{N}\{r_i\}$ is the bus width.

Each edge is also associated with a weight reflecting the *benefit* of having the I/O operations in the sets represented by the two end nodes share a communication slot. The benefit is computed in two steps. First, the *basic benefit* $w(e)$ on an edge $e = (v_1, v_2)$, which represents the benefit of combining the two end nodes

[2] the same as *freedom* used in MAHA [PPM86]

without taking into account possible exclusions of combinations of other nodes, is computed as follows:

$$w(e) = gain(e) - pf * penalty(e),$$

where

$$gain(e) = \sum_{i=0}^{N} \min(r_i(v_1), r_i(v_2))$$

is the total number of I/O pins which can be shared if nodes v_1 and v_2 are combined;

$$penalty(e) = \frac{\|frame(v_1) \cup frame(v_2)\|}{\|frame(v_1) \cap frame(v_2)\|} - 1$$

is the percentage of freedom of scheduling lost due to the combination of nodes v_1 and v_2; and pf is the weighting of the penalty factor specified by users.

In the above equation, $\|frame\|$ denotes the number of control steps in *frame*. Next, the *modified weight* $w'(e)$, which represents the benefit of combining the two end nodes, while considering the first-order effect of possibly excluding combinations of other nodes, is computed in the following way.

Let

$E_1 = \{e = (v_1, v) \mid v$ is neither v_2 nor connected to $v_2\}$,

$E_2 = \{e = (v_2, v) \mid v$ is neither v_1 nor connected to $v_1\}$,

$e_1 \in E_1 \wedge (\forall e \in E_1(w(e_1) \geq w(e)))$, and

$e_2 \in E_2 \wedge (\forall e \in E_2(w(e_2) \geq w(e)))$.

Then, the *modified benefit* $w'(e)$ is given by

$$w'(e) = w(e) - (\max(w(e_1), w(e_2)) + f * \min(w(e_1), w(e_2))).$$

The second term reflects the possible exclusion of other node combinations. Combining nodes v_1 and v_2 will not exclude the combinations $\{v, v_1\}$ and $\{v, v_2\}$ if v is connected to both v_1 and v_2. However, the combination $\{v_a, v_1\}$ will be excluded by combining v_1 and v_2 if v_a is not connected to v_2. Similarly, the combination

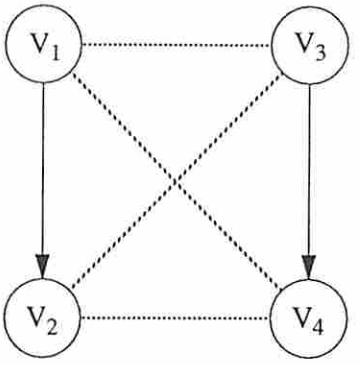


Figure 7.6: An example showing exclusion of conditional sharing

$\{v_b, v_2\}$ will be excluded by combining v_1 and v_2 if v_b is not connected to v_1. The combinations $\{v_a, v_1\}$ and $\{v_b, v_2\}$ may or may not be satisfied at the same time if v_1 and v_2 are not combined. This can be illustrated by the example shown in Figure 7.6, in which a dotted edge indicates the compatibility between the two end nodes, and an arrowed edge indicates the precedence relation between the two end nodes. The combination $\{v_2, v_3\}$ will exclude both combinations $\{v_1, v_3\}$ and $\{v_2, v_4\}$, while the combination $\{v_1, v_3\}$ will only exclude one of the combinations $\{v_2, v_3\}$ and $\{v_1, v_4\}$, and the exclusion will be due to the precedence constraints. So, f is used to partially reflect these variations, and will have a value between zero and one. It can be given by the user to explore alternative solutions.

After combining nodes v_1 and v_2, the compatibility graph is updated by replacing nodes v_1 and v_2 by node v', each pair of edges (v, v_1) and (v, v_2) by edge (v, v'), and deleting all edges (v, v_1) and (v, v_2). The attributes of the combined node $v' = \{v_1, v_2\}$ are determined by

$$\begin{aligned} frame(v') &= frame(v_1) \bigcap frame(v_2), \text{ and} \\ r_i(v') &= \max(r_i(v_1), r_i(v_2)), \qquad \text{for } 0 \leq i \leq N. \end{aligned}$$

A heuristic procedure used for conditional resource sharing among I/O operations is summarized in Figure 7.7. For a partitioned CDFG having n conditional

1: Perform ASAP/ALAP
2: Repeat until no more edge
3: Compute basic weight $w(e)$ for each edge e
4: Compute modified weight $w'(e)$ for each edge e
5: Select an edge $e = (v_1, v_2)$ having largest $w'(e)$
6: Update the compatibility graph by combining nodes v_1 and v_2
7: Compute attributes for the combined node $\{v_1, v_2\}$
8: Update ASAP/ALAP for predecessors and successors of v_1 and v_2

Figure 7.7: A heuristic procedure for conditional resource sharing among I/O operations

I/O operations, it takes at most $n - 1$ iterations since the number of nodes is decreased by one in each iteration. The most time consuming step in the loop is step 4, which has a time complexity of $O(n^3)$. So, the overall time complexity of the procedure is $O(n^4)$.

After the conditional sharing process is completed, we have a number of disjoint sets of I/O operations. The I/O operations in a set can be scheduled in the same control step to share a communication slot. However, these I/O operations are not restricted to share a communication slot. The objective of the conditional sharing process is to provide a good set-division on conditional I/O operations such that the sharing among the I/O operations in a set does not affect the sharing among the I/O operations in the other sets. The final decision on determining which I/O operations are to share a communication slot is left to the procedure of synthesizing interchip connections, because the interchip connection synthesis handle interchip connections in a more global way than the conditional sharing process, during which no unconditional I/O operations and I/O pin constraints are considered. In the procedure of interchip connection synthesis, the I/O operations in a set can be handled in the same way as I/O operations transferring the same value.

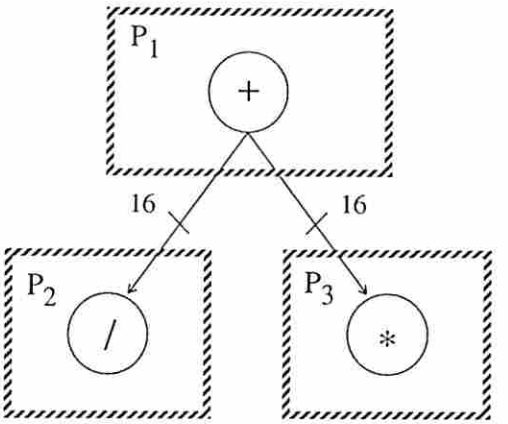
7.3 Time Division I/O Multiplexing

A value transferred across chips may have a large bit width. A large number of pins will be required to transfer the value as a whole. Sometimes, it would be desirable to split a value into several sub-values, each having a smaller number of bits, and to have these sub-values transferred in a number of cycles. In this case, I/O operations for such a value can be modeled with a *split* node, a *merge* node, and a number of I/O operation nodes, as shown in Figure 7.8(b). A *Split* node splits the bits of a value into several sub-components, each containing a smaller number of bits, while a *merge* node merges several values into a value having a larger number of bits. Note that only one *split* node is required for several I/O operations transferring a value to several partitions.

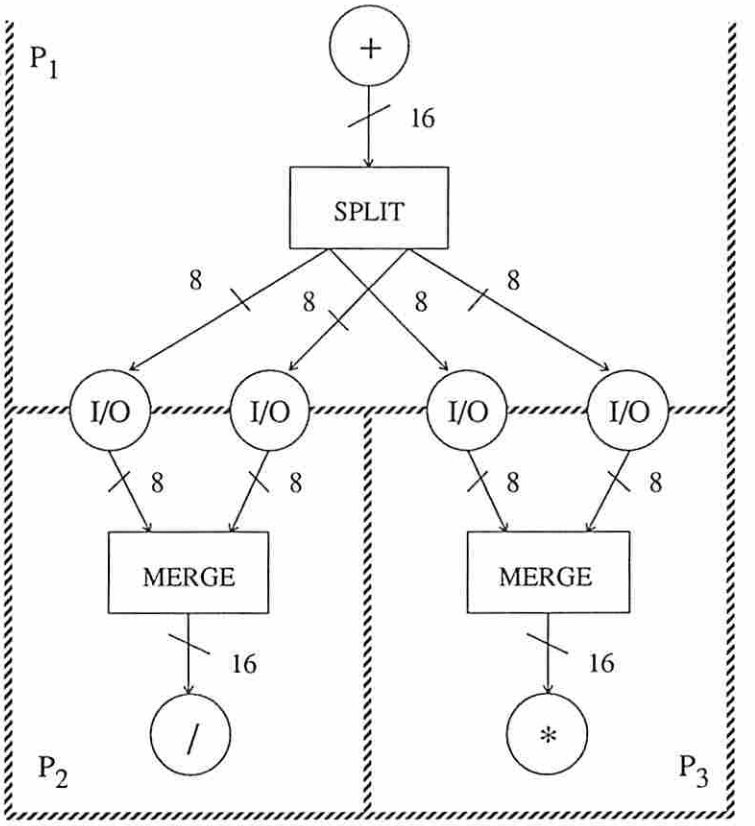
Although the number of pins required may be reduced with Time Division I/O multiplexing, larger chip area and degraded performance can result because more register control signals are required to latch the multiplexed values and/or more control steps are needed. We have assumed that the designer makes the decision which I/O operations are to be split and into how many components an I/O operations is split. Further study is required to develop a tool which could assist designers in making a time division I/O multiplexing decision or even to make the decision by itself.

7.4 Multiple-cycle Operations

An operation can take more than one cycle to execute depending on the hardware module selected. Such an operation is called a *multiple-cycle* operation. Multiple-cycle operations can be pipelined or non-pipelined depending on the implementation of the hardware modules. In this section, non-pipelined multiple-cycle operations are assumed. In our approach, a multiple-cycle operation will not be chained with other operations. The reason is that the chaining may degrade the utilization of functional units. Suppose that operations +1 and *1 are chained, as shown in



(a) Original Control Data Flow Graph



(b) Control Data Flow Graph after multiplexed I/O nodes inserted

Figure 7.8: Model for multiplexed I/O operations

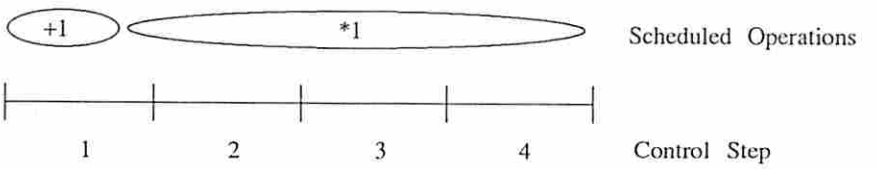


Figure 7.9: An Example of Operation Chaining

Figure 7.9, and that they are bound to functional units *add1*, and *mul1*, respectively. *Add1* cannot be used for other operations in control steps 2, 3, and 4, during which operation *1 is executing, because the result of +1 is not latched. (Values can only be latched at the boundary of cycles.)

The execution of an m-cycle operation is different from the sequential execution of m single cycle operations. The m-cycle operation must be bound to a functional unit whereas the m single cycle operations can be bound to different functional units. So, the lower bound for multiple-cycle operations, which is tighter than the bound given in BAD [KP90b, Kuc91] and MOSP [Jai90], is given by

$$o_i \geq \begin{cases} \lceil \frac{n_i}{\lfloor L/m_i \rfloor} \rceil & \text{if } L \geq m_i \\ \text{undefined} & \text{otherwise} \end{cases} \tag{7.5}$$

where

o_i: the number of functional units of type i,

n_i: the number of operations of type i,

L: the initiation rate,

m_i: the number of cycles to execute operation of type i.

Note that we cannot have a pipelined design with an initiation rate less than largest number of cycles taken by an operation.

Even for a given number of modules satisfying Equation 7.5, the schedule may not be completed if multiple-cycle operations are scheduled inappropriately. For example, there are three 2-cycle operations *op1*, *op2*, and *op3* in a CDFG, and an initiation rate of 6 is assumed. From Equation 7.5, one functional unit is sufficient.

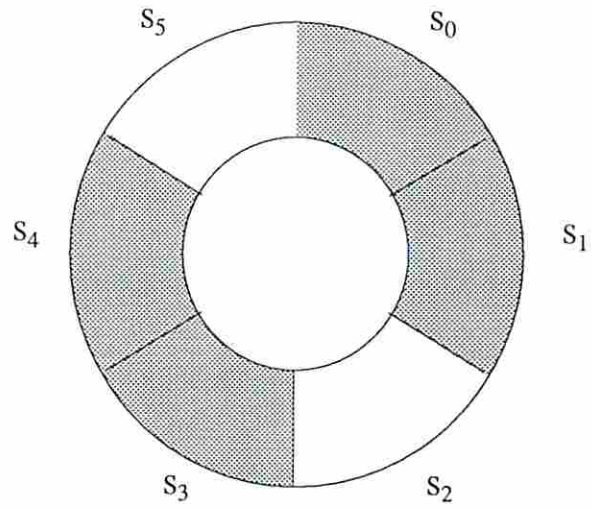


Figure 7.10: An example of an allocation wheel

However, if operations *op1* and *op2* are scheduled to start executing in control steps 0, and 3, respectively, operation *op3* is unable to be scheduled since the functional unit is free only in control steps s_2, and s_5, which are not contiguous. This can be shown with an allocation wheel in Figure 7.10, in which the shaded cells indicate a function unit has been allocated in those control steps.

Before scheduling a multiple-cycle operation, the program is required to check whether such an assignment will cause the number of resources to be insufficient for the rest of the operations. Initially, an allocation wheel of L cells is associated with each multiple-cycle module. When a multiple-cycle operation is scheduled, the corresponding cells in an allocation wheel are marked *used*. As scheduling proceeds, the allocation wheel may contain many fragments of *unused* cells. The fragmentation can decrease the maximum number of operations to which a functional unit can be allocated. If fragmentation caused by a tentative assignment causes the resources to be insufficient for the rest of the operations, the assignment has to be postponed until it is *safe*.

7.5 Conclusion

Of the topics discussed in this chapter, data recursive edges and multiple-cycle operations have been implemented in our prototype program. However, much more work needs to be done on the first three topics, namely data recursive edges, conditional I/O operations, and time division I/O multiplexing. This chapter has described the nature of the problems and spelled out some tentative approaches.

Chapter 8

Conclusions and Future Work

8.1 Contributions

In this thesis, we have addressed some problems which arise during the synthesis process of multiple-chip systems. We have also presented some techniques to synthesize multi-chip digital systems. These techniques have been implemented in our prototype software.

A class of partitioning, called simple partitioning, is identified. We have shown that the interchip communication problem of synthesizing multi-chip pipelined systems with a simple partitioning can be reduced to a pin allocation problem, which can be solved simultaneously during the scheduling process. The pin allocation problem has been formulated as an ILP. The size of the formulation is small enough to be able to solve the pin allocation problem during scheduling process.

Due to the complexity of the problem, for systems with a more general partitioning, the problem is divided into two subproblems: (1) interchip connection synthesis, and (2) scheduling. The techniques to solve these subproblems in either order have been discussed in Chapters 4 and 5. For the approach in which interchip connections are synthesized before scheduling, the problem of interchip connection synthesis has been formulated as an ILP, and solved by an heuristic search technique. We have also proven that synthesizing an interchip connection

which guarantees a pipelined schedule exists for a partitioned CDFG containing data recursive edges is an intractable problem.

For the other approach in which interchip connections are synthesized after scheduling, interchip connection synthesis can be modeled as a maximum-gain clique partitioning problem. An heuristic clique partitioning technique has been developed by taking the advantage of the special properties of the compatibility graph for the problem.

I/O pin resources can be utilized more effectively if a communication bus is allowed to transfer more than one value at the same time by using different portions of the bus. The interchip connection synthesis problem for these cases has also been formulated as an ILP. The techniques described in Chapter 4 have also been extended to deal with these cases.

We also point out some future research directions, and have presented a tentative technique for sharing communication buses among conditional I/O operations.

8.2 Future Work

The work discussed in this thesis is just the beginning. There is still more work to be done. As the experimental results show in Chapter 5, the list scheduling technique implemented in our prototype software does not work well for pipelined scheduling, and usually produces inferior schedules. Better scheduling results could be obtained by simply replacing the list scheduling by a more advanced scheduling technique, such as PLS [HHL91], which can be easily adapted to the prototype program.

The number of pins required may be reduced by using time division I/O multiplexing. However, larger chip area and degraded performance can result because more register control signals are required to latch the multiplexed values and/or more control steps are needed. Tradeoff among the number of I/O pins required, chip area and system performance requires further study.

For the approach in which interchip connection is synthesized before scheduling, the decisions made in the process of interchip connection synthesis will impose constraints on scheduling, and so affect the quality of scheduling results. Furthermore, the constraints imposed by interchip connection synthesis can exclude all feasible pipelined schedules for a design with maximum time constraints. It is desirable to develop some techniques which can give a good prediction of the impacts on scheduling during interchip connection synthesis.

An alternative approach to obtain better designs, interchip connection synthesis and scheduling should be performed simultaneously. Some scheduling techniques [NK90, PK91] are good candidates to be integrated with interchip connection synthesis, since these techniques are iterative improvement methods, in which tentative interchip connections can be determined and improved during the refinement process.

Since very little structural information is available during behavioral-level partitioning, partitioners make decisions based on some sort of predictions, which may not be accurate enough. So, the tasks of partitioning and synthesizing would likely be repeated a number of times. It would be desirable if useful information from the synthesis tools could be fedback to guide the behavioral-level partitioner. Further study is required to define some measurements, which are useful for partitioner to produce a better partitioning.

Reference List

[Bar81] M. Barbacci, "Instruction Set Processor Specification (ISPS): The Notation and its Applications," *IEEE Transaction on Computers*, C-30(1), pp. 24-40, Jan. 1981.

[CK86] R. Camposano and A. Kunzmann, "Considering Timing Constraints in Synthesis from a Behavioral Description," *Proceedings IEEE International Conference Computer Design*, pp. 6-9, Oct. 1986.

[GM90] R. Gupta and G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems," *Proceedings of International Conference on Computer-Aided-Design*, pp. 216-219, Nov. 1990.

[GE91] C. H. Gebotys and M. I. Elmasry, "Simultaneous Scheduling and Allocation for Cost Constrained Optimal Architectural Synthesis," *Proceedings of the 28th Design Automation Conference*, pp. 2-7, June 1991.

[Gebo92] C. H. Gebotys, "Optimal Synthesis of Multichip Architectures," *Proceedings of International Conference on Computer-Aided-Design*, pp. 238-241, Nov. 1992.

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

[Girc84] E. Girczyc, "Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions," *PhD thesis, Carleton University*, Jul. 1984.

[Gomo60] R. Gomory, "All-Integer Integer Programming Algorithm," *IBM Research Center Report RC-189*, January 1960; also in *Industrial Scheduling* (eds.: Muth and Thompson), Englewood Cliffs, NJ., Prentice-Hall, 1963.

[GVM89] G. Goossens, J. Vandewalle, and H. DeMan, "Loop Optimization in Register-Transfer Scheduling for DSP-systems," *Proceedings of the 26th Design Automation Conference*, pp. 826-831, June 1989.

[HCDdA88] K. S. Hwang, A. E. Casavant, M. Dragomirecky, and M. A. d'Abreu, "Constrained Conditional Resource Sharing in Pipeline Synthesis," *Proceedings of International Conference on Computer-Aided-Design*, pp. 52-55, Nov. 1988.

[HCLH90] C. Huang, Y. Chen, Y. Lin, and Y. Hsu, "Datapath Allocation Based on Bipartite Weighted Matching," *Proceedings of the 27th Design Automation Conference*, pp. 499-504, June 1990.

[HH90] L. Hafer and E. Hutchings, "Bringing up Bozo," *Tech Report CMPT TR 90-2, School of Computer Science, Simon Fraser University, Burnaby, B.C. V5A 1S6*, March 1990.

[HHL90] C. Hwang, Y. Hsu, and Y. Lin, "Optimum and Heuristic Data Path Scheduling Under Resource Constraints," *Proceedings of the 27th Design Automation Conference*, pp. 65-70, June 1990.

[HHL91] C. Hwang, Y. Hsu, and Y. Lin, "Scheduling for Functional Pipelining and Loop Winding," *Proceedings of the 28th Design Automation Conference*, pp. 764-769, June 1991.

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

[HS71] A. Hashimoto and J. Stevens, "Wire Routing By Optimizing Channel Assignment Within Large Apertures," *Proceedings of the 8th Design Automation Workshop*, pp. 155-169, June 1971.

[Jai90] R. Jain, "MOSP: Module Selection for Pipelined Designs with Multiple-Cycle Operations," *Proceedings of International Conference on Computer-Aided-Design*, pp. 212-215, Nov. 1990.

[KM90] D. Ku and G. De Micheli, "High-level Synthesis and Optimization Strategies in Hercules and Hebe," *EURASIC, Proceedings of the European Conference on ASIC design*, May 1990.

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

[KP90a] K. Küçükçakar and A. C. Parker, "MABAL - A Software Package for Module And Bus Allocation," *International Journal of Computer Aided VLSI Design*, pp. 419-436, Nov. 1990.

[KP90b] K. Küçükçakar and A. C. Parker, "BAD : Behavioral Area-Delay Predictor," *Tech Report 90-31, University of Southern California*, Nov. 1990.

[KP91] K. Küçükçakar and A. C. Parker, "CHOP: A Constraint-Driven System Level Partitioner," *Proceedings of the 28th Design Automation Conference*, pp. 514-519, June 1991.

[Kuc91] K. Küçükçakar, "System-Level Synthesis Techniques With Emphasis On Partitioning And Design Planning," *PhD Dissertation, University of Southern California*, Oct. 1991.

[Kun84] S. Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processor," *Proceedings of the IEEE*, pp. 867-884, July 1984.

[KWK85] S. Y. Kung, H. J. Whitehouse, and T. Kaliath, "VLSI and Modern Signal Processing," Prentice-Hall, pp. 257-276, 1985.

[LHL89] J. Lee, Y. Hsu, and Y. Lin, "A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis," *Proceedings of International Conference on Computer-Aided-Design*, pp. 20-23, Nov. 1989.

[LIN87] "LINDO - Users Manual for Linear, Integer and Quadratic Programming with LINDO," *LINDO Systems, Inc.*, 1987

[LP91] D. A. Lobo and B. M. Pangrle, "Redundant Operator Creation: A Scheduling Optimization Technique," *Proceedings of the 28th Design Automation Conference*, pp. 775-778, June 1991.

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

[MK88] G. De Micheli and D. C. Ku, "HERCULES: A System for High-Level Synthesis," *Proceedings of the 25th Design Automation Conference*, pp. 483-488, June 1988.

[MPC88] M. C. McFarland, A. C. Parker and R. Camposano, "Tutorial on High-Level Synthesis," *Proceedings of the 25th Design Automation Conference*, pp. 330-336, June 1988.

[NK90] J. A. Nestor and G. Krishnamoorthy, "SALSA: A New Approach to Scheduling with Timing Constraints," *Proceedings of International Conference on Computer-Aided-Design*, pp. 262-265, Nov. 1990

[NT86] J. A. Nestor and D. E. Thomas, "Behavioral Synthesis with Interfaces," *Proceedings of International Conference on Computer-Aided-Design*, pp. 112-115, Nov. 1986

[PG86] B. M. Pangrle and D. D. Gajski, "State Synthesis and Connectivity Binding for Microarchitecture Compilation," *Proceedings of International Conference on Computer-Aided-Design*, pp. 210-213, Nov. 1986.

[PK89] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 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," *Proceedings of the 27th Design Automation Conference*, pp. 77-83, June 1990.

[PK91] I. Park and C. Kyung, "Fast and Near Optimal Scheduling in Automatic Data Path Synthesis," *Proceedings of the 28th Design Automation Conference*, pp. 680-685, June 1991.

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

[PPM86] A. C. Parker, J. Pizarro and M. J. Mlinar, "MAHA: A Program for Datapath Synthesis," *Proceedings of the 23rd Design Automation Conference*, pp. 461-466, June 1986.

[PR91] M. Potkonjak and J. Rabaey, "Optimizing Resource Utilization Using Transformations," *Proceedings of International Conference on Computer-Aided-Design*, pp. 88-91, Nov. 1991.

[PS82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, pp. 326. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1982.

[Spr91] D. L. Springer, "Coloring and Clique Partitioning for Data Path Allocation," *PhD thesis, Carnegie Mellon University*, May 1991.

[Tri87] H. Trickey, "Flamel: A High-Level Hardware Compiler," *IEEE Transactions on Computer-Aided Design*, vol. 6, pp. 259-269, March 1987.

[TS83] C. Tseng and D. P. Siewiorek, "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proceedings of the 20th Design Automation Conference*, pp. 490-496, June 1983.

[VHDL88] IEEE Standard VHDL Language Reference Manual. The Institute of Electrical and Electronics Engineers Inc., March 1988.

[WS89] N. Woo and H. Shin, "A Technology-Adaptive Allocation of Functional Units and Connections," *Proceedings of the 26th Design Automation Conference*, pp. 602-605, June 1989.

[WY89] K. Wakabayashi and T. Yoshimura, "A Resource Sharing and Control Synthesis Method for Conditional Branches," *Proceedings of International Conference on Computer-Aided-Design*, pp. 62-65, Nov. 1989.