Architectural Synthesis for FPGA Based Signal Processing Systems

Behzad Sajjadi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented as Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science
Concordia University
Montreal, Quebec, Canada

April 1996

ISBN 0-612-18436-6

Abstract

Architectural Synthesis for FPGA Based Signal Processing Systems

Behzad Sajjadi

In this thesis, we present architectural models for data paths used in signal processing applications suitable for FPGA implementation technologies. First, we target the multiplexor based data paths that are suitable for a number of FPGAs which do not support buses or SRAMs. We also address the modeling for partitioned data paths which is essential because of the limited gate capacities of some FPGAs. We then represent a data path model for FPGAs that support busses and RAMs. Our model allows maximum flexibility in scheduling bus transfers independent of operation scheduling whereas conventional approaches assume coincidence of bus transfers and operations. This technique reduces the number of buses in the data path and achieves high throughputs by efficiently storing data variables using RAMs. We also introduce three novel integer linear programming formulations for the synthesis of multiplexer based, multiple partitioned and bus based data paths. By using much tighter constraints than previous approaches and effective cost measures (*structural complexity*) in our formulations, we can produce optimal data paths that are favorable compared to other synthesis methodologies. Finally, we demonstrate the qualities of the data paths resulting from our approach for typical signal processing synthesis benchmarks.

To my parents:
Mostafa and Fakhri Sajjadi
and my wife:
Mahshid Izady

Acknowledgement

I would like to thank my supervisor Dr. Baher Haroun for all of his guidance and support. His great enthusiasm during teaching my undergraduate studies encouraged me to pursue my Masters and his vision and great ideas throughout my graduate studies made this work possible.

Special thanks to Dr. Asim Khalili as he was a true teacher in life for me and for guiding this thesis to the end.

Also, great appreciation should go to my wonderful wife and my parents for their support and encouragement throughout my studies.

And finally, I would like to thank all my friends in VLSI lab and the VLSI staff as they made my stay at Concordia University more enjoyable.

Table of Contents

List of Figures and Tables

Abbreviations

ALAP	As Late As Possible
ASAP	As Soon As Possible
CDFG	Control Data Flow Graph
CLB	Configurable Logic Block
DCT	Discrete Cosine Transform
EWF	Elliptic Wave Filter
FDCT	Fast Discrete Cosine Transform
FPGA	Field Programmable Gate Array
FU	Function Unit
HLS	High Level Synthesis
IDCT	Inverse Discrete Cosine Transform
ILP	Integer Linear Programming
LE	Logic Elements
LP	Linear Programming
MPGA	Mask Programmable Gate Array
MIP	Mix Integer Programming
OSTA	Optimal Synthesis Tool for Architectures
LUT	Look Up Table

Chapter 1: Introduction and Thesis Overview

1.1 Introduction

1.1.1 Design Automation

This thesis presents an automated synthesis methodology for digital signal processing data paths in VLSI ASICs and FPGAs. Design automation of complex digital systems have been an active area of research in recent years. Generally, synthesis is a task that takes a specification of the behavior required of a system and a set of constraints and goals to be satisfied and then finds (constructs/designs) a structure that implements the behavior based on satisfying the goals and constraints [6].

High Level Synthesis was initially concentrated on logic circuits than sequential systems. In recent years there has been a trend toward automating synthesis at even higher levels of the design hierarchy due to the following reasons:

a) Since much of the cost of the chip is due to design development, automating this process can significantly reduce the cost and also produce a shorter design cycle.

b) Another advantage in automation of synthesis is to reduce errors in the final design. Therefore, less debugging is needed as the final design is much closer to the initial specifications.

c) Having the ability to perform fast searches within the design space can produce different designs with various trade offs. Hence, a wider design space exploration is possible. Designer concentrates on high level specification rather than the lower level details. Therefore more complex designs can be handled by fewer designers in shorter cycles [7].

d) Another reason for the emphasis on high-level design methodologies is that high-level architectures are self-documenting which facilitates the redesign. As a result of using

standard specification languages it is much easier for a non-expert to produce a design which meets a given set of specification without being knowledgable in the underlying implementation technology.

As we focus in this thesis on data paths architecture synthesis, in the next section high level synthesis is defined and the subtasks of architectural synthesis are reviewed. These include scheduling, and allocation of function units, registers, and interconnects.

1.1.2 High Level Synthesis (HLS)

We define synthesis as translation from a behavioral description into a structural description as shown in Figure 1.1 [8]. We can represent design tools as arcs along a domain's axis or between the arcs. Basically, High Level Synthesis tools generate from an algorithm, a Register-Transfer Level structure which includes a data path, that is network of registers, function units, multiplexers and buses.

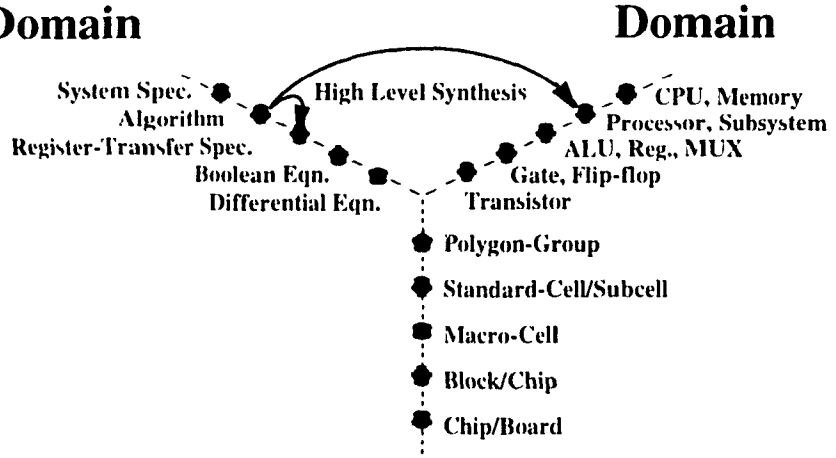


Figure 1.1: Transition in the Y-chart defines the High-level synthesis within the domains and levels of description [8].

Recently, HLS has been applied to the design of Digital Signal Processing applications. Different implementation technologies have been targeted: (1) VLSI using full custom,

Standard Cell and Mask Gate Arrays, and (2) Field Programmable Gate Arrays (FPGA). FPGAs have been among the fastest growing segments of the semiconductor industry, with applications ranging from the integration of a few thousand gates to microprocessors (10's of thousands of gates).

The restricted structure of FPGAs as compared to the flexibility of full custom VLSI, makes efficient mapping of designs on FPGAs more complex and time consuming. As a result, design automation tools starting at higher levels of abstraction are essential to the effective use of FPGAs.

In this thesis, as highlighted in Figure 1.1, we address the synthesis task of mapping an algorithm to a final architecture by performing resource allocation, scheduling, and resource assignment. First, we review the basic techniques and various approaches in High Level synthesis along with the existing problems for mapping architectures into M/FPGAs using these approaches. Then we represent architectural models that are used in our synthesis technique.

1.2 Basic Subtasks in Synthesis Task

As we explained earlier, the synthesis task starts with a behavioral description of a digital system and a set of constraints to produce a structure of the digital system that satisfies the constraints and minimizes an area-delay cost function. This approach consists of the following subtasks. [9]

1.2.1 Behaviour Specification

The first subtask is to describe the behavior of the digital system using Hardware Description Language (HDL) and translation of this language into a Control Data Flow Graph (CDFG).

1.2.2 Scheduling

The next subtask, operational scheduling assigns each operation in the CDFG to a specific control step. Roughly speaking, operation scheduling determines the cost-speed trade offs of the design. Therefore, the scheduling problem can be divided into *Time-Constraint Scheduling* and *Resource-Constraint Scheduling*, each of them with different requirements. Time-Constraint scheduler finds the cheapest possible schedule for a given maximum number of time steps. In this case, the design is subject to a speed constraint and the objective is to find the minimum number of resources to meet this time constraint. On the other hand, resource-constraint scheduler has limitation on the area (number of resources) and therefore, it tries to find the fastest schedule which satisfies the constraints.

There are number of scheduling techniques which can be categorized as follows: *Iterative/Constructive algorithms and Transformal algorithms*. As Soon As Possible (ASAP) scheduler which simply assigns every operation in CDFG from the first control step to the last control step as soon as all its precedecessors are scheduled, is one example of Iterative algorithm. Another example is As Late As Possible (ALAP) which performs the same task as ASAP, only the operations are assigned from the last control step to the first with precedence relation still applied. The above two schedulers are not resource constraint as they do not put any restriction on the maximum number of function units required, therefore, they are not practical. A more practical type of Iterative approach is Force-Directed scheduling [27],[34]. In this algorithm "force" values are calculated for all the operations at all feasible control steps. The pairing of operation and control step that has the most attractive force is selected and assigned. After the assignment, the forces of the remaining operations are reevaluated. This evaluation and assignment is iterated until all the operations are assigned. Since the maximum number of control steps needs to be defined, force-directed corresponds to time-constraint scheduling. List-scheduler [35],[36] is another example of Iterative/Constructive algorithm. In this scheduler, there is limitation on the maximum number of function units within every control steps. The

scheduler assigns the operations according to their precedence relationship and a priority given to them according to heuristic rules. The list-scheduler corresponds to resource-constraint scheduling as it requires that the number of function units be specified. Another type of Iterative/Constructive algorithm is Integer Linear Programming (ILP) approach [11]. In this technique, the scheduling problem is formulated as an optimization problem meeting certain constraints. The objective is to minimize the cost of the resources and the total number of control steps under these constraints. One disadvantage of ILP is the difficulty of formulating the problem. It also requires excessive CPU time to solve large problems. However, it can be a viable approach for small to medium problems as it produces optimal solutions.

Simulated Annealing algorithm which is based on stochastic search is an example of Transformal approach [11]. Beginning with an initial schedule, the annealing procedure improves upon the schedule by iteratively modifying it and evaluating the modification. A modification is accepted for the next iteration if it results in a better schedule. If the modification does not result in a better schedule, its acceptance is based on a randomized function of the quality improvement and the annealing temperature. This approach has a major advantage as it helps the optimization problem not to get stuck in local minima. It also produces near optimal solutions. However, it suffers from long computation time and requires complicated tuning of the annealing parameters.

There are number of other scheduling algorithms with the goal of minimizing both the delay and the area of the design. The major task is to assign each operation to a control step where it will be executed.

1.2.3 Allocation and Binding

The next subtask performs the assignment (binding) of function units to operations, and storage units to the data variables based on minimally allocating resources such as

multiplexers, registers, wires and interconnects. Scheduling and allocation are interdependent, since scheduling has an important effect on the hardware to be allocated and allocation fixes a limit on the parallelization of operations when scheduling.

1.2.4 Control

The final subtask produces a control unit based on the schedule graph and data path assignment produced by previous subtasks.

Previously, it was argued that by decomposing synthesis into these sub-problems, one can approach each part separately and optimally solve each part. However, this decomposition, does not necessarily produce a global optimal solution and it is only intended to reduce the time complexity and search iteration loops. It was observed from the decomposition approach that the complexity of the structure of the resulting data path was high. This produced a very inefficient design specially in interconnect costly technologies such as MPGA and more importantly FPGA's. This has led to our conclusion and others [10], that only by combining these tools can an efficient structure be produced. In order to achieve an efficient data path, optimal approaches such as Stochastic Search or Integer Linear Programming with branch and bound should be used. There are three main issues in the optimal approaches which needs to be considered. These are: (1) design space construction which is easier to construct for stochastic approaches than ILP solutions. This is due to the fact that a full formulation for ILP is required and generally formulating the problem is a difficult task. (2) design space search which needs careful consideration of moves in the space for stochastic approaches otherwise search time will be large, while for ILP there are standard approaches built around branch and bound methods. There are also, commercial solvers such as GAMS/CPLEX or LINDO which are based on the advance search techniques. (3) objective function to be minimized is also an important factor which needs to be considered as a good objective function can select the optimal architecture.

ILP approaches have an advantage as they can guarantee an optimal solution which allows us to accurately qualify the objective function. Our tool is based on both these approaches. There are three ILP formulations which are the major contribution of this thesis. These formulations perform the scheduling, allocation and binding of operations (for both partitioned and non-partitioned architectures), bus transfer scheduling, bus allocation and binding, bus loading and storage minimizations. For the data storage assignment, interconnection and clock cycle minimizations, and actual floorplanning and routing we use the stochastic approach presented in [13]. These approaches will be explained later.

1.3 Architectural Models for Data Paths

It has become apparent that application specific signal processing systems can be implemented or prototyped using reconfigurable boards of multiple field programmable gate arrays (FPGA). In designing signal processing architecture using multiple FPGAs, one is always faced by the internal FPGA structural characteristics [14], and the restriction imposed on the connections between the multiple FPGAs on the board [15], [16]. These characteristics influencing an architectural model are:

(a) The *limited interconnection resources* for FPGAs makes interconnection a primary resource for optimization especially for long bit-width data paths. Otherwise, larger (more costly) FPGAs may be required. Also, the interconnections delay is large in FPGAs due to the configurable series switch resistance and capacitive effects, limiting interconnection loading is critical in determining the cycle time of the architecture.

(b) The abundance of registers (almost as abundant as the logic blocks in most FPGAs). This makes the resources such as adders have the same value as registers and this can influence the final architecture style.

(c) Configurable Logic Block (CLBs)are composed of programmable logic, Look Up Table (LUT) followed by a register. Hence, data paths which utilize this property of having

registers in front of logic, stand to maximize the usage of the FPGA resources.

(d) The most general logic modules have a limited fan in (e.g. 4 i/p LUT or 8 i/p multiplexers, etc.). Hence, architecture design that minimizes (and/or can constrain) the fan in at any module is very efficient. Otherwise an extra level of logic gets added with a detrimental effect on the architecture clock cycle.

(e) In case of a multiple FPGA system (e.g. [15], [16]), the application is partitioned between the different FPGAs in the system. Due to the interconnection structure of the board with multiple FPGA, the input/output pin limitation of each FPGAs on the board and the large pin to pin (external to the FPGAs) delay associated with such connections, the architectures implementing the partitions on each FPGA have to be concurrently synthesized for optimal performance under resource (pin and gate count) constraints.

(f) Some families (e.g. Actel for FPGA) do not allow for buses (i.e. tri-state drivers) internal to the FPGA, hence multiplexers are the only method for logic-block sharing. However, some other types of FPGAs (e.g. Xilinx XC4000 series and the ORCA ATT2C series) have global wires that can be driven by tri-state drivers, to implement either wide muxes or chip wide busses. Therefore, synthesis approaches which can easily be adapted to both families tend to be more efficient.

These and other details of internal structural characteristics of FPGA [14] influence the determination of an architectural model and synthesis approach of a data path. The following sections represent the architectural models used (in this thesis) for multiplexer based and bus based data paths.

1.3.1 Multiplexer Based Data Paths

For multiplexer based data paths, we used the architectural model represented in Figure 1.2 [1]. Different types of modules used, are: register module, function unit module, and functional unit multiplexer module. If the number of mux inputs is small, the functional

unit multiplexer module can be part of the function of CLBs implementing the function unit module. On the other hand, when the number of mux inputs is large, extra mux submodules are implemented using extra CLBs.

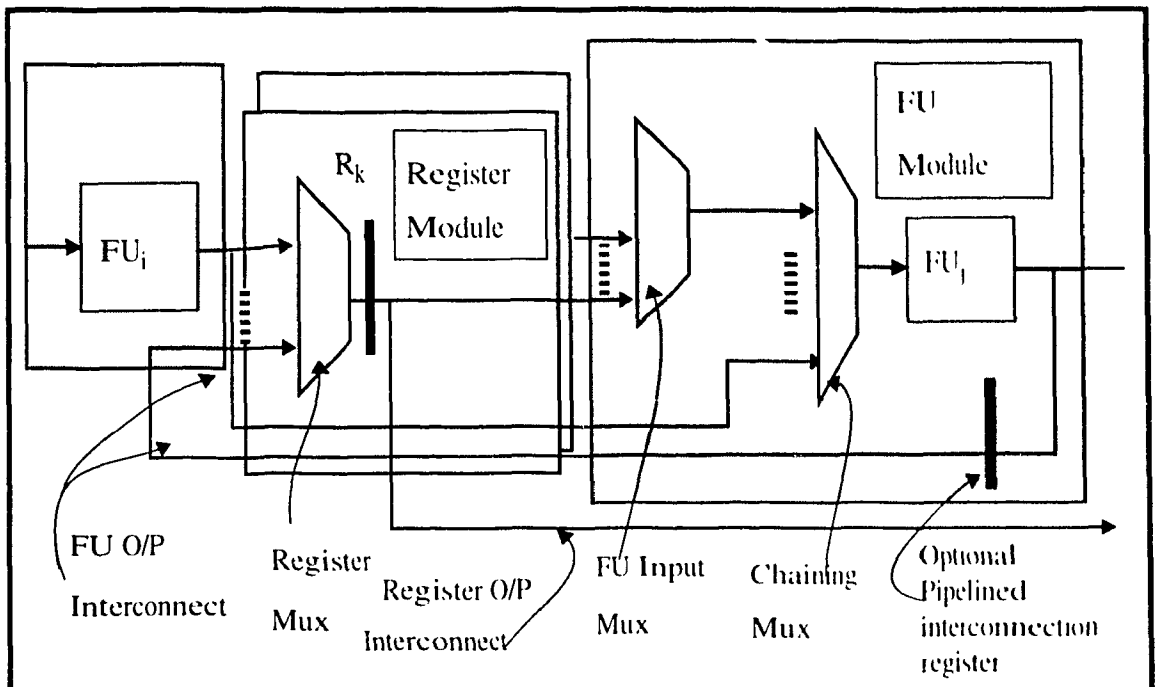


Figure 1.2: A multiplexer based data path model. Most FPGAs are composed of *logic elements* (LE) or *combinational logic blocks* (CLB) that have a register or a latch at the output of the logic. Note that a register module maps (in a bit slice manner) effectively to a LE or CLB in an FPGA. The FU module also maps to a CLB or LE in a bit-slice manner. The multiplexers at the input of a FU can be supported either by the same CLB/LE or by another preceding CLB/LE. Typically the number of mux inputs that are supported by a single CLB/LE is limited for a given technology, therefore an architecture that can limit the number of inputs per Module is desirable to reduce the depth of the logic used to implement a FU or Register Module. A pipeline register can optionally be used at the output of a FU (at no extra LE/CLB cost) to pipeline a very long interconnection, as long as the data is not required by the destination FU in the next clock cycle.

Input/output ports to the system are considered as either function units or registers. A module is typically mapped on an M/FPGA to neighboring logic blocks with local connections between the bit-slices. When mux fan in is higher than the maximum fan in of a logic block an extra level of logic blocks are added to account for the extra mux inputs. The register at the output of a function unit can be used to pipeline a high fan out or long interconnect. This requires an extra clock cycle for the data transfer. The next chapter deals with this issue in more details. Both single and two phase clocking schemes can be used in our architectural model.

1.3.2 Bus Based Data Paths

The model of the architecture that is used for bus based data paths follows the general structure of Figure 1.3 [3].

An example of a full data path is shown in Chapter 5, Figure 5.6 [5]. Different types of modules used, are: register module, register-file module (implemented as a RAM), functional unit and FU multiplexer module. If the number of mux inputs is small, the FU multiplexer module can be part of the function of the CLBs implementing the FU module. On the other hand, when the number of mux inputs is large, extra mux sub-modules are implemented using extra CLBs.

A two phase clock ($\Phi 1$ *and* $\Phi 2$*) or a pseudo two phase* ($\Phi 1$ *and* $\overline{\Phi 1}$ *) can be used* to define the data transfers between the register-file (RAM) and the registers of the data path. For data transfers between the registers of the data path only one clock phase is used. The data is transferred from a data path register to the FU input, and then through a FU output to one or more of the data path registers.

For the transfers between the registers and the register file, $\Phi 1$ *and* $\Phi 2$ (or $\overline{\Phi 1}$) are used. The data path registers are considered as "master" registers and the slaves are the storage locations inside the register-file (RAM).

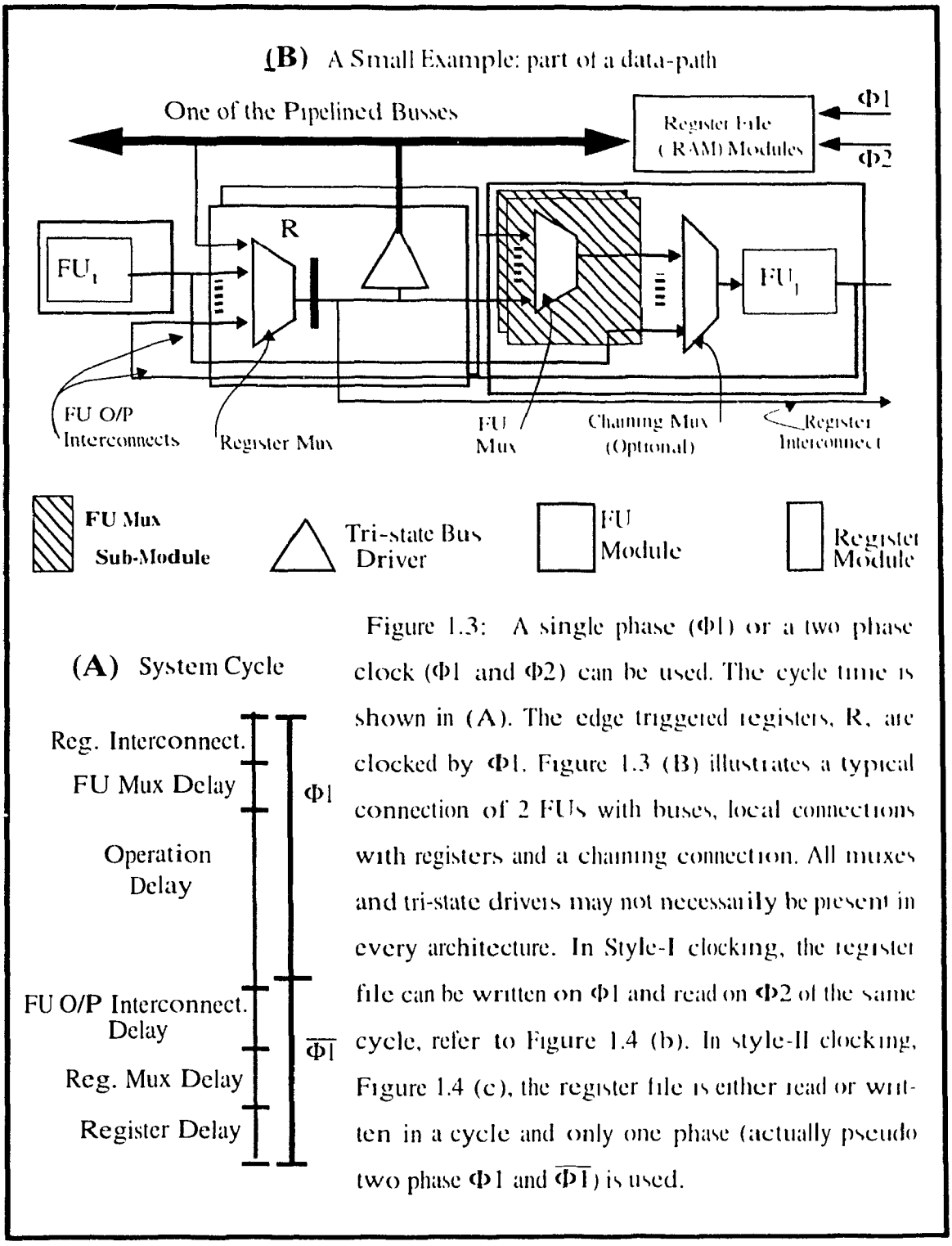


Figure 1.3: A single phase (Φ1) or a two phase clock (Φ1 and Φ2) can be used. The cycle time is shown in (A). The edge triggered registers, R, are clocked by Φ1. Figure 1.3 (B) illustrates a typical connection of 2 FUs with buses, local connections with registers and a chaining connection. All muxes and tri-state drivers may not necessarily be present in every architecture. In Style-I clocking, the register file can be written on Φ1 and read on Φ2 of the same cycle, refer to Figure 1.4 (b). In style-II clocking, Figure 1.4 (c), the register file is either read or written in a cycle and only one phase (actually pseudo two phase Φ1 and $\overline{\Phi 1}$) is used.

The clocking is explained in Figure 1.4 (b). Note that the RAM is written in the beginning of the cycle (Φ1) and read at the end (Φ2 (or $\overline{\Phi 1}$)). Hence, the cycle time is determined by either the critical path between any of the data path registers or the read and

write time plus the bus delays of a register file

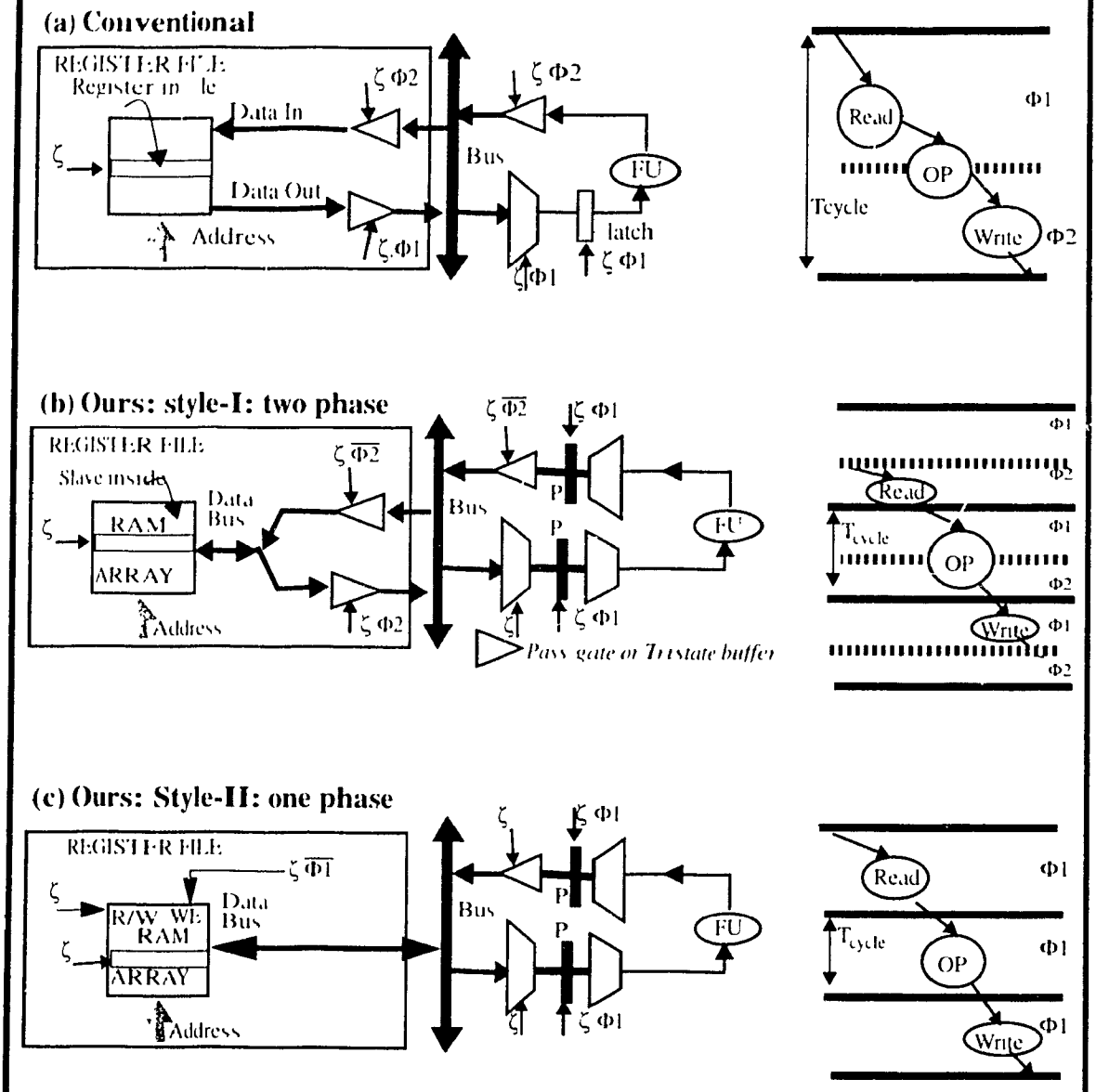


Figure 1.4: (a) A non-pipelined bus with a register file where read and write access delays are chained in one cycle. (b) Style I: A pipelined bus and register file (note the different clocking) where the read or write times take their own part of different cycles. The bits ζ's generated from the controller are "AND"ed with the clocks (i.e. $\zeta.\Phi1$, $\zeta.\Phi2$) to indicate the latching time. In Style-II, (c), either a read or a write can be performed in one cycle where only a single phase is required for the timing as shown. Style-I and II require that data stored in a register file to have a life time greater than or equal to 2 and 3 cycles respectively.

Input and output ports to the system can be considered as data path registers (R).

Our architecture model allows both individual registers and grouped storage in register files which are implemented as RAMs, hence reducing storage area. For example, in XC4000 FPGAs one can store 32 bits in the form of a RAM in a CLB instead of only 2 bits as registers. Support of RAMs is essential, especially in the case of handling signal processing algorithms which require large storage (e.g. multi-channel filters).

We propose two styles for clocking of the data path that allow for an efficient implementation. Figure 1.4 (a) shows the conventional register file model (a) where the register file is accessed by a non-pipelined bus (e.g in [17][18][19] and [20]). Figure 1.4 (b & c) show our proposed style-I and II of the data path where a register file based on RAM implementation is accessed by a pipelined bus. In case of the non-pipelined bus access (conventional case), the read and write time of the register file array are part of the system cycle as shown. When large data storage is required, chaining the large read and write time in one cycle increases the system cycle time which reduces the clock rate and hence may result in large performance reduction.

By pipelining the bus, the read and writes are scheduled in separate cycles preceding and following the operation respectively. The decision then to store the data in a large RAM can only be done if the life time of a variable is at least two cycles for a style-I data path and three cycles for a style II data path.

By allowing any variable to exist in more than one storage location, that is in a register and in a register file location, and by supporting flexible bus transfers, our approach has removed the bus delay from the critical path. The buses and their associated register files are hence treated as functional units and their influence on the clock cycle duration is independent of other functional unit delays. Therefore, the clock cycle of the data path is controlled by either a critical path through a FU or through a register file and a bus but not by adding both as is the case in conventional bus based data paths. Hence, style-I and II

have faster clock rates than conventional approaches. They also require less buses and resources which we will later demonstrate.

Based on these data paths, in following chapters, we will introduce an Integer Linear Programming (ILP) approach which concurrently performs scheduling, allocation and binding of operations and can produce architectures with minimum interconnects, multiplexers and storage usage within a reasonable running time. We will also show the advantages of our approach over previous ILP techniques and we demonstrate how this approach can be used for implementing Field Programmable Gate Arrays (FPGAs).

Chapter 2: Overview of the Synthesis Approach for Optimal Mapping of Signal Processing Architectures into M/FPGAs

In this chapter, we will review previous work done on synthesis and present an overview of the design methodology for synthesis for which the thesis has contributed the optimization ILP formulation.

2.1 Literature Review of Synthesis of Multiplexer Based Data Paths

Our target is to synthesize architectures for the boards with multiple FPGA's. Hence, the system can be implemented on one FPGA or partitioned on more than one FPGA depending on its complexity. Existing data path synthesis approaches targeting a *random topology* architecture, (i.e., a point to point connectivity model using muxes, registers and functional units) target VLSI implementations. These approaches either use heuristics, such as Cathedral-III [21], Sehwa [28], using stochastic search [29],or use optimal approaches (ILP) (e.g. Oasic[12], [10], [17], and [22]), and generate *multiplexer based* FPGA architectures but do not explicitly address the concerns for FPGAs (refer to chapter 1, section 1.3). *Specifically*, they do not have the capability of *accurately* accounting for interconnections *concurrently* in their operation scheduling and binding inside each FPGA or partition **(intra-partition interconnection)** as well as accurately accounting for pin and interconnection limitations between different partitions **(inter-partition interconnections)**.

2.2 Literature Review of Synthesis of Bus Based Data Paths

Other approaches targeting bus based architectures (e.g. Cathedral-II [23],

HYPER[25], SPAID-X[19], STAR[26], and [20]) assume that a bus transfer occurs in the same cycle of an operation, hence the cycle time becomes larger than the bus delay, RAM read and write delay in addition to a FU and other multiplexer delays. For such architectures, cycle times are long and are more suited for ASIC implementations where special techniques can be used to reduce the effects of bus and RAM delay[18].

Previous approaches using ILP synthesis of bus based architectures [17][19], have assumed that for variables generated by an operation and destined to a register file through a bus, such a bus transfer occurs at the same clock cycle as the source operation is executing. Similarly, variables that are transferred by a bus to a FU, are transferred directly to the FU input through the bus from the register files. This scheduling restriction is due to the hardware data path and synthesis model which assumes that a variable can only be stored in only one register file location during its life time. *Such a restriction is not necessary.* As it is shown in our model, by allowing one variable to exist in more than one register and register file location and by supporting flexible bus transfers, the bus delay is removed from the critical path which determines the architecture cycle duration. The buses and their associated register files are hence treated as functional units (may also become on the critical path of cycle calculations, on their own, and not in series with other FUs). Details of this model was represented in chapter 1, section 1.3 and an example of final architecture is shown in chapter 5.

2.3 Importance of Interconnection Minimization Concurrently with Scheduling

In previous approaches the binding of *data transfers* to interconnections is performed after the operation scheduling, which may result in architectures with a large use of non-local interconnects, especially those with fan outs greater than 1. In addition, there is a trade-off between decreasing register storage and the number of multiplexers and

interconnections used within a single or multiple FPGAs. Hence, trade-offs in scheduling that may result in more registers but less interconnections have not been investigated previously. We demonstrate the above argument with a small example [1] [4].

Case-1 of Figure 2.1 represents the scheduling and binding without accounting for

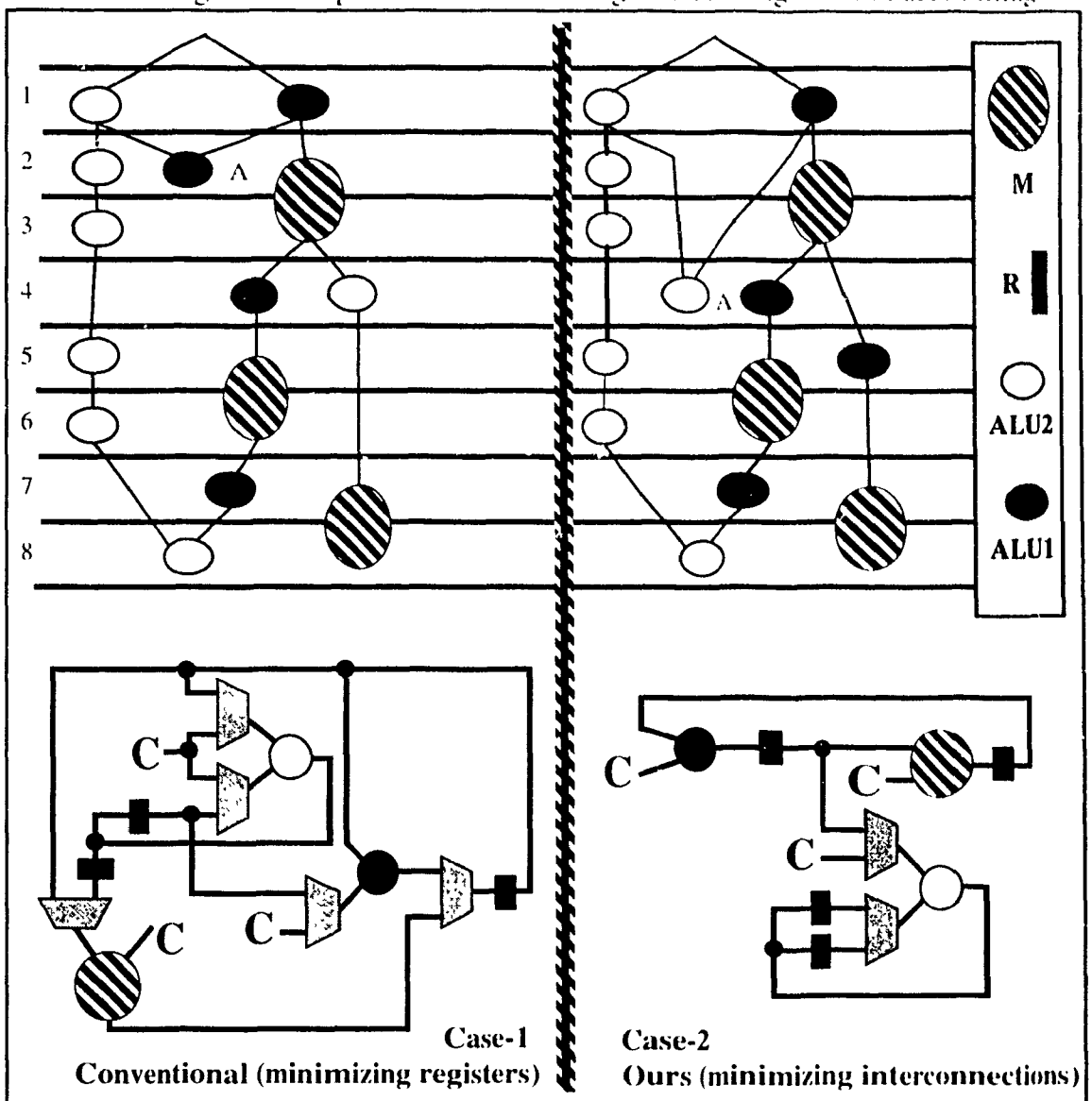


Figure 2.1: Importance of minimizing interconnections together with scheduling Case-1 uses 10 mux inputs, 3 registers and has 4 interconnects with fan outs greater than 1, while case-2 uses only 4 mux inputs, 4 registers and has 2 interconnects with fan outs greater than 1.

interconnections (but with register minimization) as done in conventional approaches. Case-2, our approach represents the scheduling and binding that accounts for interconnections (part of a structural complexity cost).

In case-1, 7 mux inputs and 3 registers are used, while in case-2 only 2 mux inputs and 4 registers are used. Case-2 has an obvious simpler structure than case-1. Case-1 has 4 interconnects with fan out greater than 1 while case-2 has only 2. Notice that for case-1, the maximum number of overlapping variable life-times are 3. These were obtained because register minimization forces operation A not to schedule in cycle 4 or higher. If operation A was scheduled in cycle 4, as in case-2, this would increase the number of overlapping variable life-times to 4. However, in case-2, a structural complexity measure is used and the scheduling of operation A in cycle 4 minimizes that measure. It is clear that Case-2 has a better architecture. This structural complexity is later explained in more details.

2.4 Synthesis Overview

In this section, we present a three step synthesis process which was developed in [3]. This will clarify the contribution of this thesis to the synthesis process as will be explained in the following discussions.

In our approach, as shown in Figure 2.2, the splitting of a synthesis task into three steps is facilitated by using a flexible architectural model that allows independent scheduling and binding of operations, allocating buses and scheduling bus transfers and finally interconnection and storage binding in conjunction with floor planning. Our tool, *OSTA (Optimal Synthesis Tool for Architectures)*, which is based on an ILP approach for scheduling and bus assignment followed by a stochastic storage and interconnect binding, is capable of performing all the synthesis tasks presented in Figure 2.2.

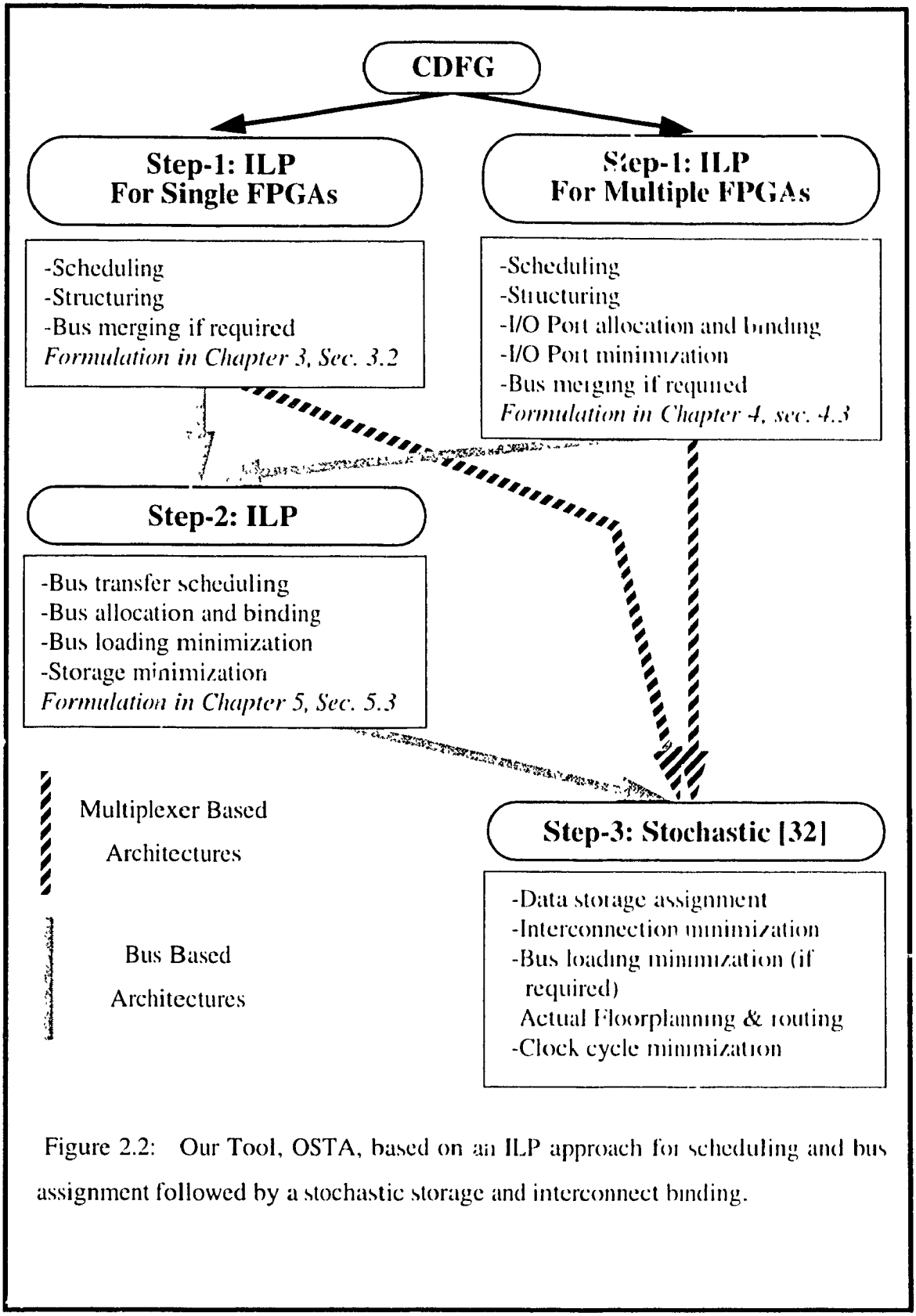
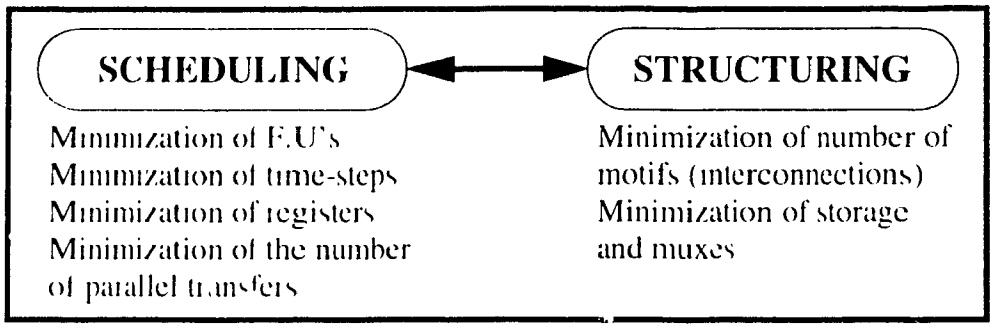
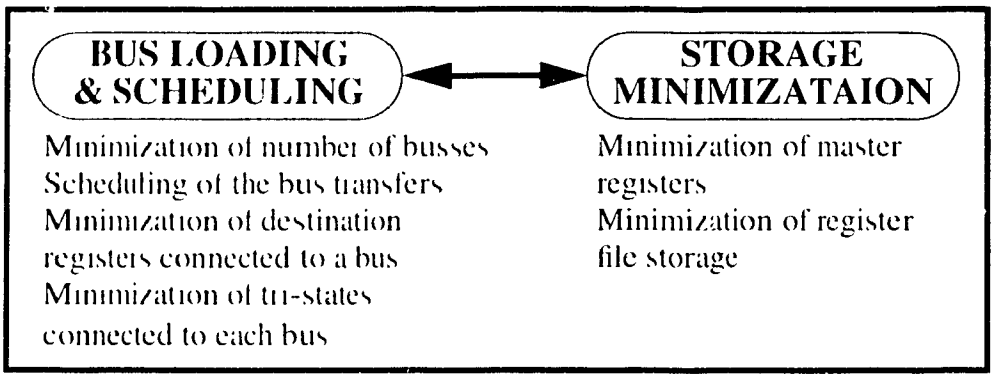


Figure 2.2: Our Tool, OSTA, based on an ILP approach for scheduling and bus assignment followed by a stochastic storage and interconnect binding.

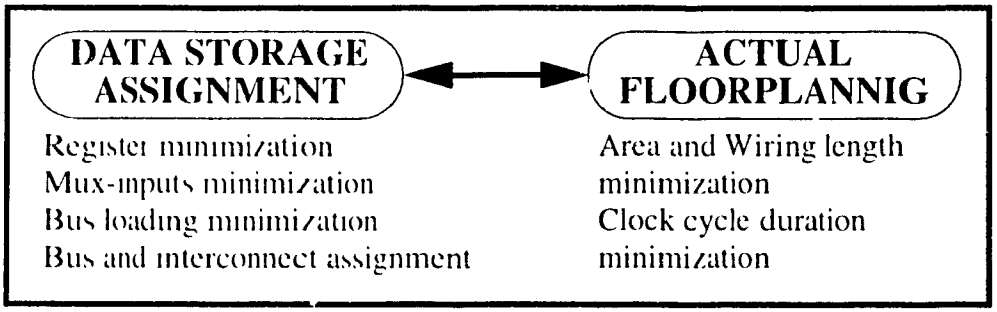
This approach is ideal for ILP based solutions as it reduces the number of zero-one variables used to represent the design space for achieving reasonable synthesis CPU seconds. Figure 2.3 [5] summarizes the tasks performed in all three steps.



(A) Step 1



(B) Step 2



(C) Step 3

Figure 2.3: (A) Step-1: Scheduling and Structuring are strongly interdependent and are, therefore, performed together (Chapter 3 and 4). (B) Bus loading and scheduling along with storage minimization are discussed in Chapter 5. (C) Step-3: The decision to assign a variable to a specific register, or a data transfer to a specific bus is strongly dependent on the floorplan. This is detailed in [32].

In general any operation scheduling and binding solution can be used in step-1. Since scheduling and structuring are strongly interdependent, they should be performed together. Therefore, minimization of function units, time-steps, registers, number of parallel transfers, number of motif instances (interconnections), and finally storage and muxes are all performed in step-1. This step can be used for both mux and bus based architectures with few minor modifications. Chapter 3 and 4 describe novel ILP formulations for non-partitioned and partitioned signal processing algorithms respectively. These formulations are suitable for the synthesis tasks of step-1. The main reasons for recommending the minimization of a structural complexity cost during scheduling and operation to function unit binding are: 1)This reduces local interconnections by optimally binding operations to function units. 2)This can increase the existence of data transfers with life time greater than two that are specifically suitable to be assigned to buses which can reduce the final possible number of local interconnections by merging them with buses in step-2 of the synthesis.

Transforming multiplexor based structural data paths to FPGAs supporting RAMs and buses is done in the second step of the synthesis process. Chapter 5 introduces an ILP formulation that optimally selects and assigns data transfers to buses while scheduling the bus transfers to minimize the use of the buses, bus loading and data storage for both registers and register files.

Finally, the decision to assign a variable to a specific register, or a data transfer to a specific bus is performed in the third stage of our synthesis process. Since these factors are strongly dependent on the floorplan, the stochastic approach presented in [32] can be used to perform data storage assignment, interconnect and bus loading minimization along with actual floorplanning and routing with clock cycle reduction

The following chapters explains each of these steps and outlines the ILP formulations that perform these tasks and produces optimal architectures which are suitable for FPGAs.

The preceding section presents a structural complexity criteria (first developed in [1]) as a cost measure for our ILP formulations.

2.5 Structural Complexity Criteria

As we explained earlier, in our synthesis methodology, we assume that register binding is performed after combined operation scheduling and binding to FUs. The register binding is delayed to a phase where detailed placement and routing of the final architecture is performed concurrently with register binding as explained in [13]. This delayed binding of storage, makes it essential to account for interconnection and storage cost while doing scheduling and binding.

Without loss of generality, we will restrict our discussions to the class of behavior that can be executed with two types of functional units (FU), the Adder (⊕) and Multiplier (⊗). The total number of ⊕ and ⊗ allocated are $N_{\oplus}$ and $N_{\otimes}$, respectively. Each FU is indexed with indices i and j from 1 to $N_{\oplus}$ and $N_{\otimes}$, respectively ($\oplus_i$ or $\otimes_j$).

A motif in the graph is an item representing all edges that have the same FU bound to its source operation and the same FU bound to its destination operation. There are four types of motifs in our case as shown in Figure 2.4. Hence, the binding of operations to FU

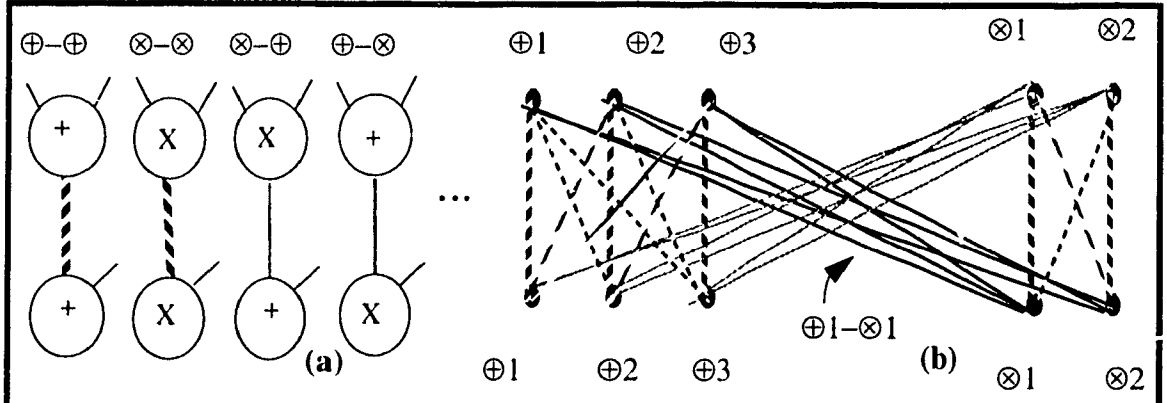


Figure 2.4: (a) The 4 types of motifs indicated next to the data transfers, (b) all possible instances for all the 4 types of motifs for $N_{\oplus}$= 3 and $N_{\otimes}$ =2. One instar ce (⊕1–⊗1) out of the $(N_{\oplus}+N_{\otimes})^2$ is shown.

can rather be posed as a binding of data transfer edges (or variables) to motifs. One important property of a motif is that: if two data transfer edges are assigned to that motif and their life-times are non-overlapping, then they can be assigned the same interconnections and the same register. If their life-time is overlapping then they use different registers. A motif (e.g.$\oplus_i$–$\otimes_j$) represents a direct interconnection (refer to Figure 1.2), from the output of $\oplus_i$ to one or more of the register modules connected to the input of $\otimes_j$. Note that we do not differentiate between which input of the FU is used. The reason is two fold. First, the cost of extending the interconnect locally from one FU input to another is small. Second, operation alignment may possibly take care of minimizing this local interconnection. If no motif is used between two different FU then these FUs are not directly connected in hardware. *By minimizing the number of motifs used, we are minimizing the interconnects, therefore maximizing the use of these connections by the data transfers* (refer to Figure 2.1).

In order to minimize storage, we *minimize the maximum overlap* between all data transfer (edge) life-times with the same destination functional unit type. This measure is suitable if our goal is to maximally re-use registers to minimize their count in the data path. Such a blind minimization can result in significant cost in terms of number of high fan out interconnects and also large multiplexers at the input of the registers. Hence, other measures are required, with which, the different structural differences can be balanced. These measures are explained below.

Minimizing the number of incompatible motifs to reduce multiplexer cost: Two different motif instances will result in at least an extra mux input if their destination nodes (FU) are the same(e.g. $\oplus_k$–$\otimes_j$, $\oplus_l$–$\otimes_l$, i/=k, l=j). The extra mux inputs are at the input of a register or at the destination of FU, or both. These two motif instances are said to be *incompatible*.

Any motif that gets assigned to a data transfer which has a life-time of one cycle is said

to be *nonpipelinable*. The interconnection corresponding to that motif cannot be pipelined since the data transfer requires one extra cycle to forward the data from the function unit pipeline register to the destination register. Hence, this measure maximizes the number of pipelinable interconnections. Pipelining of long/high fan out interconnects is done upon mapping the data path to the FPGA.

Therefore, a structured data path architecture will result from a binding of the data transfer edges to motifs that minimizes a weighted sum of: the number of motif instances, overlaps in data transfers, the number of incompatible motifs and the number of nonpipelinable motifs. This sum is the "structural complexity" measure.

In this thesis we use the above structural complexity criteria as a cost measure in our ILP formulations to account for all the issues pointing to FPGA architecture. The following chapters demonstrate the effectiveness of using this criterion in producing structured architectures.

Chapter 3: ILP Synthesis of Non-Partitioned Signal Processing Architectures for Multiplexer Based FPGAs

3.1 Introduction

In this chapter, we introduce an integer linear programming formulation for combined scheduling and operation bindings. This formulation which is intended for multiplexer based data paths, produces architectural results having considerably lower interconnections and mux inputs than previous ILP solutions.

Previous approaches do not have the capability of accurately accounting for interconnections concurrently in their operation scheduling and binding. The binding of data transfers to interconnections is performed after the operation scheduling assignment to function units. This can result in architectures with large use of interconnects, specifically those with fan outs greater than one. In addition, there is a trade off between decreasing register storage and the number of multiplexers and interconnections used. These trade offs are investigated here.

This chapter is organized as follows: Section 3.2, outlines the approaches and the ILP formulation that performs combined scheduling and binding. The results of using our formulation for typical signal processing synthesis benchmarks are presented in section 3.3. Finally concluding remarks are made in section 3.4.

3.2 ILP Formulation for an Optimal Architecture

The objective is to minimize a weighted combination of the total execution time and the "**structural complexity**" measure. We have used a set of variables as the basis for the ILP formulation. The notations used along with the list of variables are summarized in Figure 3.1 and Figure 3.2 respectively [4].

Fu_t: Different type function units used in CDFG.

N_t: Max. number of FU type t allowed at each cycle.

Op: Set of all operations. **W:** Set of all operations without successors.

Range(op): the cycles in which the operation can be scheduled.

D(op): Delay of each operation.

L(op): Latency of operation on a FU.

Op(i)<Op(j): precedence relation; Op(i) should occur before Op(j).

$Op(i) \rightarrow Op(j)$ **:** edge with Op(i) as source and Op(j) as destination.

Wrap: set of all wrap around edges (recursive z^{-1} edges).

Figure 3.1: Notations used in our formulation.

$X_{(op,n,s)}$: =1 only if operation "op" is assigned to function unit "n" and is scheduled into cycle "s" otherwise = 0.

$Motif_{(Fut, n, Fut', n')}$: =1, if there is a direct path for data from function unit "n" to function unit "n' " otherwise = 0.

CStep: Integer variable representing the total number of control steps.

$Incomp_{(Fut)}$: Integer variables representing the maximum number of incompatible motifs for each type function unit

$Nonpipe_{(Fut, n, Fut', n')}$: = 1, if motif(Fut, n, Fut', n') is non-pipelinable.

$Maxovlap_{(Fut)}$: Real variables representing the maximum overlap of edges of same destination function unit at all cycles.

Figure 3.2: Integer variables used in our formulation.

Our formulation consists of number of constraints and an objective function. Equation (3.1) guarantees that each operation is assigned to only one function unit and one cycle.

$$\sum_{s \in Range(op)} \sum_{n=1}^{Nt} X_{op,n,s} = 1 \qquad \forall op \qquad (3.1)$$

Constraint (3.2) ensures that every function unit can be assigned to maximum one operation within the same control step.

$$\sum_{p=s-L(op)+1}^{s} \sum_{op \in Fu_t} X_{op,n,p} \leq 1 \qquad \forall n, \forall s \qquad (3.2)$$

Inequality (3.3) ensures that the precedence relations of the data flow graph will be preserved. This prevents an operation from being assigned after another operation if an actual or virtual edge exists between these operations.

$$\sum_{p=s-D(op_i)+1}^{ALAP(op_i)} \sum_{n=1}^{Nti} X_{op_i,n,p} + \sum_{p=ASAP(op_j)}^{s} \sum_{n=1}^{Ntj} X_{op_j,n,p} \leq 1 \qquad (3.3)$$

$$\forall (op(i) < op(j))$$
$$ALAP(op_i) + D(op_i) - 1 \geq \forall s \geq ASAP(op_j)$$

No operation should be scheduled after CStep as described in constraint (3.4).

$$\sum_{s \in Range(op)} \sum_{n=1}^{Nt} s \times X_{op,n,s} - CStep \leq (-D(op) + 1) \qquad \forall op \in W \qquad (3.4)$$

These first 4 constraints are similar to the ones in [10]. Using only these four constraints in conjunction *with only* overlap minimization (similar to [10]), results in a number of equivalent optimal bindings that may eventually produce data paths with a high number of interconnects. Case in point, refer to Figure 2.1, to see the effect of taking motif

minimization into account. These 4 constraints ensure scheduling precedence and binding conflict resolutions.

The Extended Formulation: In order to include all terms of the structural complexity, we used another four constraints.

We add inequality (3.5) to count motifs. Figure 3.3 explains how this inequality is used

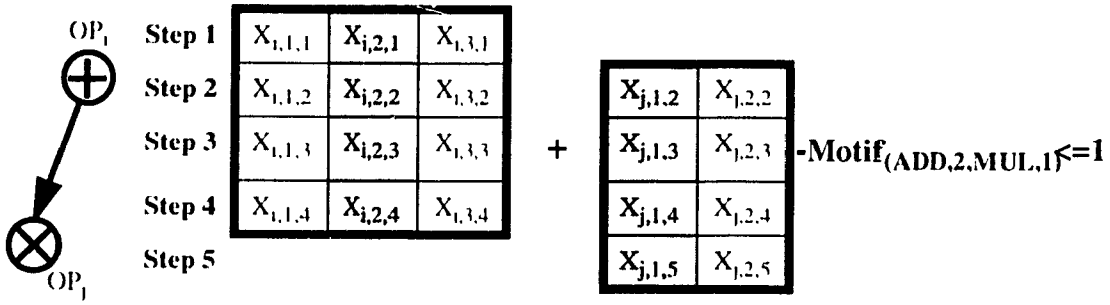


Figure 3.3: For every edge in the signal flow graphs, including recursive edges (with z-1 delay), we assign sum the shaded Xijk variables as shown. Other constraints ensure that only one of the Xijk variables per highlighted rectangle is set to one, while the others are zero. If one X is set in each shaded column, then Motif(add,2,mul,1) is set to 'one' to satisfy the constraint.

to set the zero-one variable $\mathbf{Motif_{(ADD,2,MUL,1)}}$. This variable is used in our cost function, eqn. 3.9 as explained later.

$$\sum_{\substack{s \in Range(op_i) \\ op_i \in Fu_i}} X_{op_i,n,s} + \sum_{\substack{s \in Range(op_j) \\ op_j \in Fu_{i'}}} X_{op_j,n,s} - Motif_{(Fu_i,n,Fu_{i'},n')} \leq 1 \qquad (3.5)$$

$$\forall (op_i \rightarrow op_j) \qquad \forall n\,(n = 1 \ldots Nt) \qquad \forall n'\,(n' = 1 \ldots Nt')$$

The number of incompatible motifs are minimized by introducing an integer variable for each type of functional unit. These variables are set to the maximum of the total

number of edges with the same destination type in inequality (3.6).

$$\sum_{Fu_t n=1}^{Nt} Motif_{(Fu_t,n,Fu_{t'},n')} - Incomp_{(Fu_t)} \le 0 \qquad \begin{matrix} \forall n' \\ \forall Fu_{t'} \end{matrix} \qquad (3.6)$$

Constraint (3.7) is used to minimize the number of non-pipelinable motifs. It sets the variables "Nonpipe" to 1, only if the length of the motif is one cycle.

$$X_{op_i,n,s} + X_{op_j,n',(s+D(op_i))} - Nonepipe_{(Fu_t,n,Fu_{t'},n')} \le 1 \qquad (3.7)$$

$$\begin{matrix} \forall (op_i \rightarrow op_j) & \forall n\,(n = 1 \ldots Nt) & op_i \in Fu_t \\ \forall (s \in Range(op_i)) & \forall n'\,(n' = 1 \ldots Nt') & op_j \in Fu_{t'} \end{matrix}$$

Using inequality (3.8), the number of edges in a cycle that have the same destination type is calculated. The "Maxovlap" variable for each FU type is set to the maximum of these numbers over all cycles. For multiple output operations, since we have no knowledge at this step if these output edges could be shared by the same register, the best assumption is to consider them separately but with less than unity contribution (K < 1).

$$K \times \left(\sum_{\substack{\forall (op_i \rightarrow op_j) \\ op_j \in Fu_t \\ edge \notin wrap}} \left(\sum_{p=ASAP(op_i)}^{s} \sum_{n=1}^{Nti} X_{op_i,n,p} - \sum_{p=ASAP(op_j)}^{s} \sum_{n=1}^{Ntj} X_{op_j,n,p} \right) + \right. \qquad (3.8)$$

$$\left. \sum_{\substack{\forall (op_i \rightarrow op_j) \\ op_j \in Fu_t \\ edge \in wrap}} \left(\sum_{p=ASAP(op_i)}^{s} \sum_{n=1}^{Nti} X_{op_i,n,p} + \sum_{p=s+1}^{ALAP(op_j)} \sum_{n=1}^{Ntj} X_{op_j,n,p} \right) \right) - Maxovlap_{(Fu_t)} \le 0$$

$$K < 1 \qquad \forall s \qquad \forall Fu_t$$

The first term in the objective function, eqn. (3.9), accounts for the total execution time. The second term accounts for the number of motifs. Minimization of incompatible motifs is done using term 3 of objective function. Nonpipelinable motifs are minimized by term 4, and finally term 5 accounts for the overlap measure.

Minimize *Objective Function* **(3.9)**

$$(C1 \times Cstep) \qquad term\,(1)$$

$$+\left(C2 \times \sum_{Fu_t} \sum_{n=1}^{Nt} \sum_{Fu_{t'}} \sum_{n'=1}^{Nt'} Motif_{(Fu_t,n,Fu_{t'},n')} \right) \qquad term\,(2)$$

$$+\left(C3 \times \sum_{Fu_t} Incomp_{(Fu_t)} \right) \qquad term\,(3)$$

$$+\left(C4 \times \sum_{Fu_t} \sum_{n=1}^{Nt} \sum_{Fu_{t'}} \sum_{n'=1}^{Nt'} Nonpipe_{(Fu_t,n,Fu_{t'},n')} \right) \qquad term\,(4)$$

$$+\left(C5 \times \sum_{Fu_t} Mavovlap_{(Fu_t)} \right) \qquad term\,(5)$$

3.3 Synthesis Results

Our extended formulation and minimizing the structural complexity was used for the 5th order elliptic filter and FIR filter to demonstrate a number of points regarding our approach of combined scheduling and binding. Our ILP model was implemented on GAMS/CPLEX solver and ran on a SPARC10. All results reported here for these filters have been proven optimal by CPLEX and times reported is the sum of Linear Programming (LP) time and Mixed Integer Programming (MIP) time. Our synthesizer accepts the range of the operations and the precedence relations along with the list of

recursive edges as the input, and produces optimal scheduling and binding according to the weights of coefficients in the objective function.

3.3.1 Elliptic Filter

Table 3.1 represents the various synthesis results for both retimed [17] and non-retimed [27] 5th order elliptic filter. It is important to note that *we include the cost of storage of all recursive variables (z^{-1} delays in the filter specification) used* in the elliptic filter example unlike a number of other published solutions.

Table 3.1: 5th order Elliptic Filter

	No Retiming			Retimed		
No. of adders used	2	2	2	2	2	2
No. of Multipliers used (cycle, non/pipelined)	1(2,p)	2(2,np)	1(2 np)	1(2,p)	2(2,np)	1(2,np)
Cycles	19	19	21	17	17	20
Types of Motifs	6	8	6	6	8	6
Non-pipe. Motifs	5	7	5	5	7	5
Incomp. $Motifs_A$	3	3	3	3	3	3
Incomp. $Motifs_M$	1	1	1	1	1	1
$Max\_overlap_A$	13.5	13.5	13.5	12.6	12 6	12.6
$Max\_overlap_M$	1.8	1.9	1.9	1.8	1.8	1 8
Solution Time (s)	249	1642	1810	21.3	73.2	48 7
K=0.9, C1=3, C2=2, C3=C4=C5=C6=1						

The scheduling and binding of operations using two adders and two multipliers (column 2 of Table 3.1) is shown in Figure 3.4. If only the first 4 constraints are considered, similar to [10], the result can be as high as 12 motifs compared to 8 motifs

obtained by our formulation.

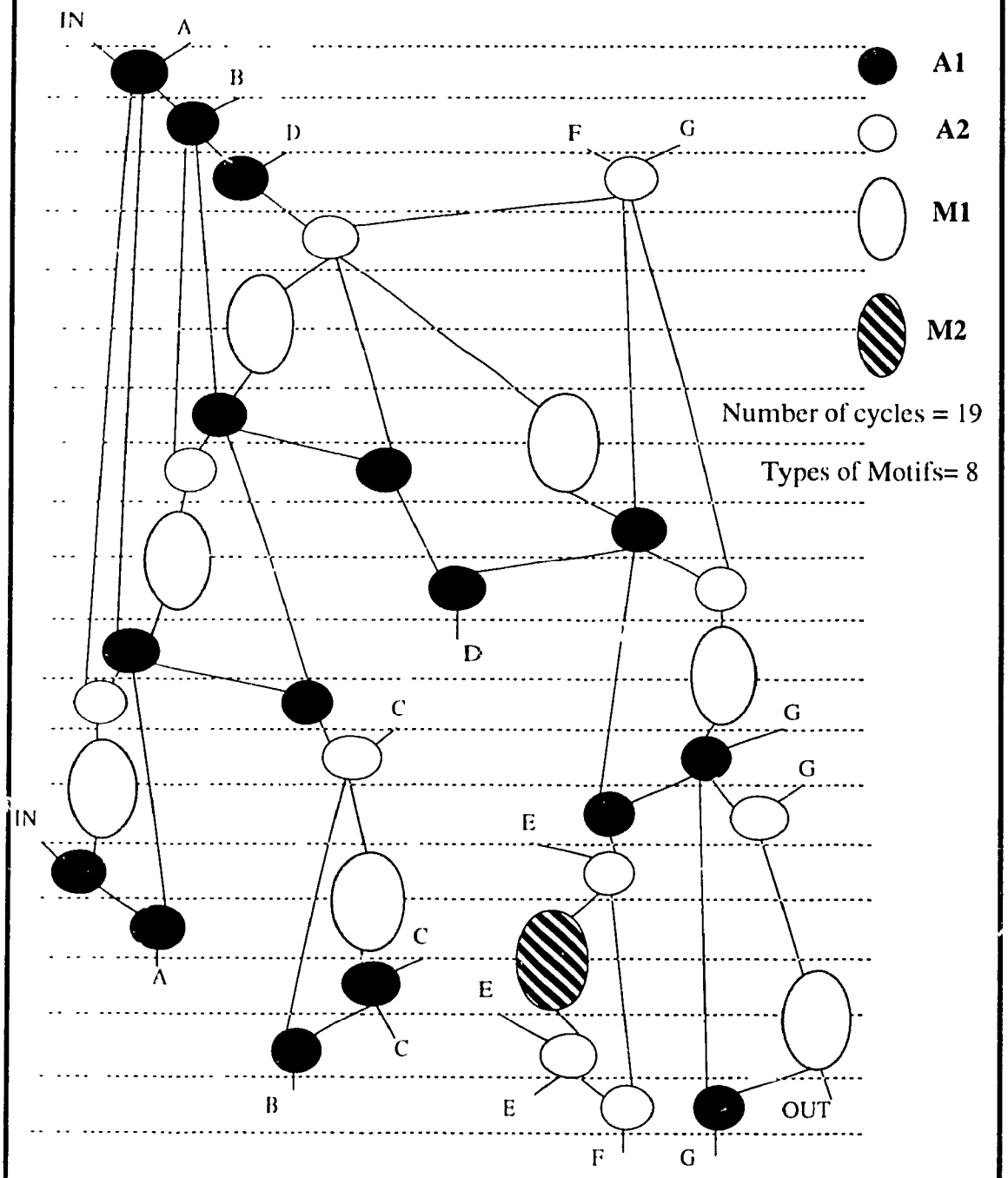
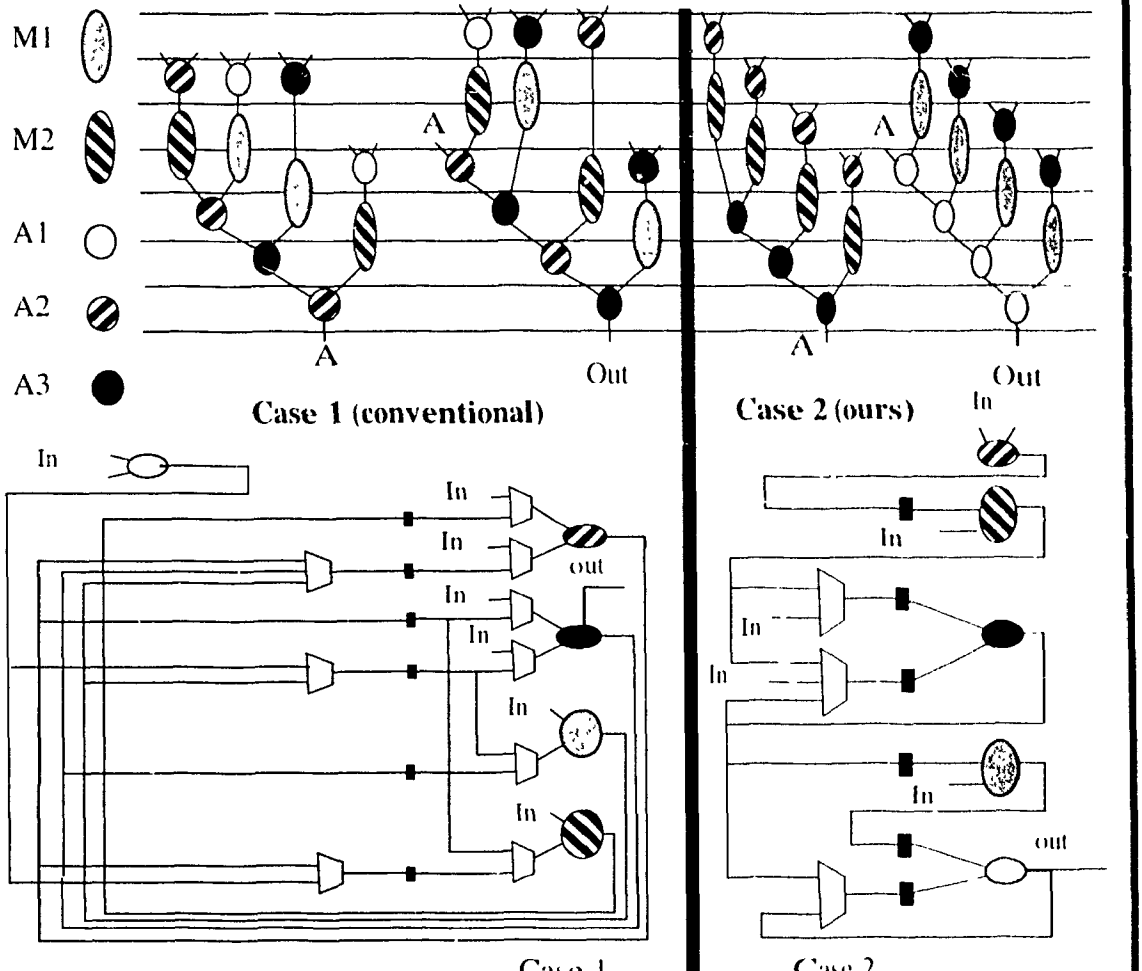


Figure 3.4: Our scheduling and binding for the 5th order elliptic filter using 2 adders and 2 two-cycle, non-pipeline multipliers. For this binding a minimum of 11 registers can be assigned including recursive edges (A, B, C, D, E, F, G). Number of inter-connects with fan out>1 = 8.

3.3.2 *FIR Filter*

Synthesis results of FIR filter, Figure 3.5, emphasizes that without motif minimization



	Case 1	Case 2
Motif types	10	7
Mux inputs	19	7
Registers	6	6
Max Fan-out	3	3
Inter connect	20	14
Interconnect. fan out>1	7	4
Solution time (sec.)	---	109

Figure 3.5: 16 Point functionally pipelined FIR filter example. The schedules of case 1&2 use same # of cycles, FUs and REGs. But, minimizing motifs in combined scheduling and binding results in a better architecture (case 2, our approach) than separate scheduling followed by binding with wire minimization (case 1, obtained by using a similar ILP approach in [22]). This example demonstrates a single phase architecture.

combined with scheduling (the approach in [22]) a more complex architecture may result compared to ours. It is difficult for us to compare architectural features with other optimal ILP formulations as their final architectures are not published.We have attempted to do such a comparison, by removing the structural complexity terms from our cost function (leaving only the maximum overlap and function unit binding [10] followed by wire minimization [22]). Hence we obtain the architecture shown in Case-1 of Figure 3.5. A close comparison of the final data paths using our approach and the conventional ILP, clearly illustrates the reduced complexity of interconnection of the data path. An important consequence of using the structural complexity is the reduced number of networks with high fan out and the maximum fan out of networks in our resulting data paths. This can imply higher speed as well as reduced power dissipation because of reduced loading capacitance in the architecture.

Another important conclusion can be drawn for this FIR example. The add-multiply accumulate structure of this algorithm is very regular. Yet, none of the previous synthesis approach could produce a regular schedule for this algorithm and prove its optimality. Heuristics were previously used to cluster the algorithms for extracting regularity, but that was done independent of scheduling [30]. In our case, the regular schedule combined with an efficient binding was a direct consequence of using the structural complexity in our cost function.

3.3.3 Fast Discrete Cosine Transform

For FDCT [31] (which is a medium example, with high parallelism), using 2 ALUs and 2 Multipliers, 14 cycles and 10 motifs are obtained in 328sec (proven optimal within 6%(=OPTCR) of optimal cost). Nevertheless, from our experience the running time is dependent on the optimization criteria (OPTCR, in CPLEX). This time increases significantly for highly parallel algorithms, when the upper bound on CSTEP is much

larger than the critical path and when the total number of functional units is large (>8). In a number of situations a feasible solution is always obtained within minutes but the optimizer would run for hours without reaching the optimal solution. To prove optimality, we had to use a lower bound on the cost function and an upper bound on CSTEP to reduce running time. Good heuristic solutions can be used for obtaining such bounds.

3.4 Chapter Summary

In this chapter, we detailed an optimal approach for the construction of structured data paths with minimum interconnections using ILP. The main contribution of this section is the novel integer linear programming which is based on a mux based data path. Our formulation is more general than previous ILP formulations and our architectural results are more comprehensive, have better structure and are viable for small to medium examples.

Chapter 4: ILP Synthesis of Partitioned Signal Processing Architectures for Multiplexer Based FPGAs

4.1 Introduction

The intention of this chapter is to introduce an integer linear programming formulation for optimally mapping partitioned signal processing algorithms on a multiple FPGA board. Also, an architectural model that can be used for multiple FPGA boards, multiplexer based data paths is presented.

Previous approaches addressing partitioned systems in general have not allocated hardware ports nor performed the binding of data transfers to the ports, hence no constraints on the pin count or the structure of the interconnection between multiple FPGAs (or chips) was imposed. Our approach by performing operation scheduling, function unit allocation, interconnections, data transfers scheduling between FPGAs & input/output port allocation simultaneously produces an architectural model suitable for multiple FPGA boards and accounts for limitations in the structure of interconnections of MFPGAs.

The plan of this chapter is as follows. Section 4.2 describes the properties of our architectural model used in the synthesis process along with structural complexity measures for multiple data paths. Section 4.3 introduces the ILP formulation for simultaneous allocation, scheduling and binding of both operations and data transfer operations between partitions. Section 4.4 demonstrate the quality of the results of our approach for typical signal processing synthesis benchmarks including the difficult example of discrete and inverse discrete cosine transforms. Finally, the concluding remarks are stated in section 4.5.

4.2 Modeling and Optimization Criteria

4.2.1 Architectural Model:

Our synthesis technique supports the following general structural style (Figure 4.1) as a model for the architecture [2].

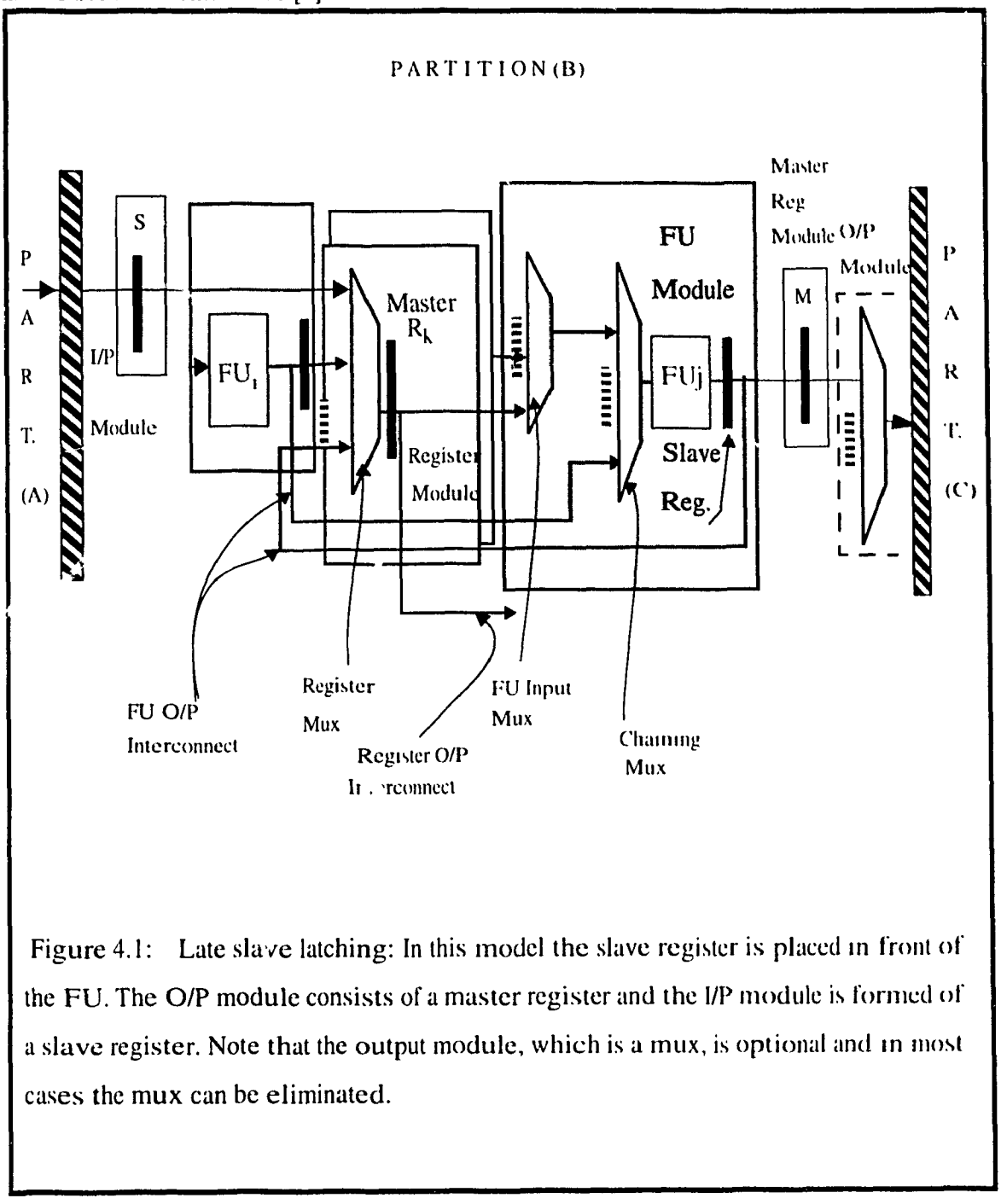


Figure 4.1: Late slave latching: In this model the slave register is placed in front of the FU. The O/P module consists of a master register and the I/P module is formed of a slave register. Note that the output module, which is a mux, is optional and in most cases the mux can be eliminated.

This model is suitable for FPGAs since the slave registers are already provided at the output of FUs. But the draw back is that the master register should stay valid for the duration of FU which may result in utilizing more registers in the case of multi-cycle operations.

Note that the worst case delay between any master and slave register is due to either:

a) muxes and FU inside the partition, or

b) off-chip delay in between partitions.

For an example of a full data path using this two phase clocking scheme see Figure 4.8.

When the FPGA technology supports storage by using latches (e.g. Actel), there is an advantage for a two phase clocking scheme as it reduces the number of latches in the data path. This is because two latches are used to make an edge triggered flip/flop (register).

For a single phase clocking scheme, every register module uses twice the number of latches as a master register module in a two phase implementation. The slave registers, in case of the two phase clocking, offset that difference (in number of latches) by only the number of functional unit outputs. The number of slaves is usually much smaller than the number of master registers in a typical data path.

On the other hand, in case of a two phase clocking scheme, the clock period is divided into two phases that are not usually equal in terms of the critical path delay in each phase. Moreover, the total clock cycle duration is the sum of the worst case of two different critical paths delays as opposed (in case of single phase clocking) to the delay of a different critical path. Hence, it is possible that single phase implementation may have faster clocks (smaller clock duration). This argument is explained in Figure 4.2 (next page).

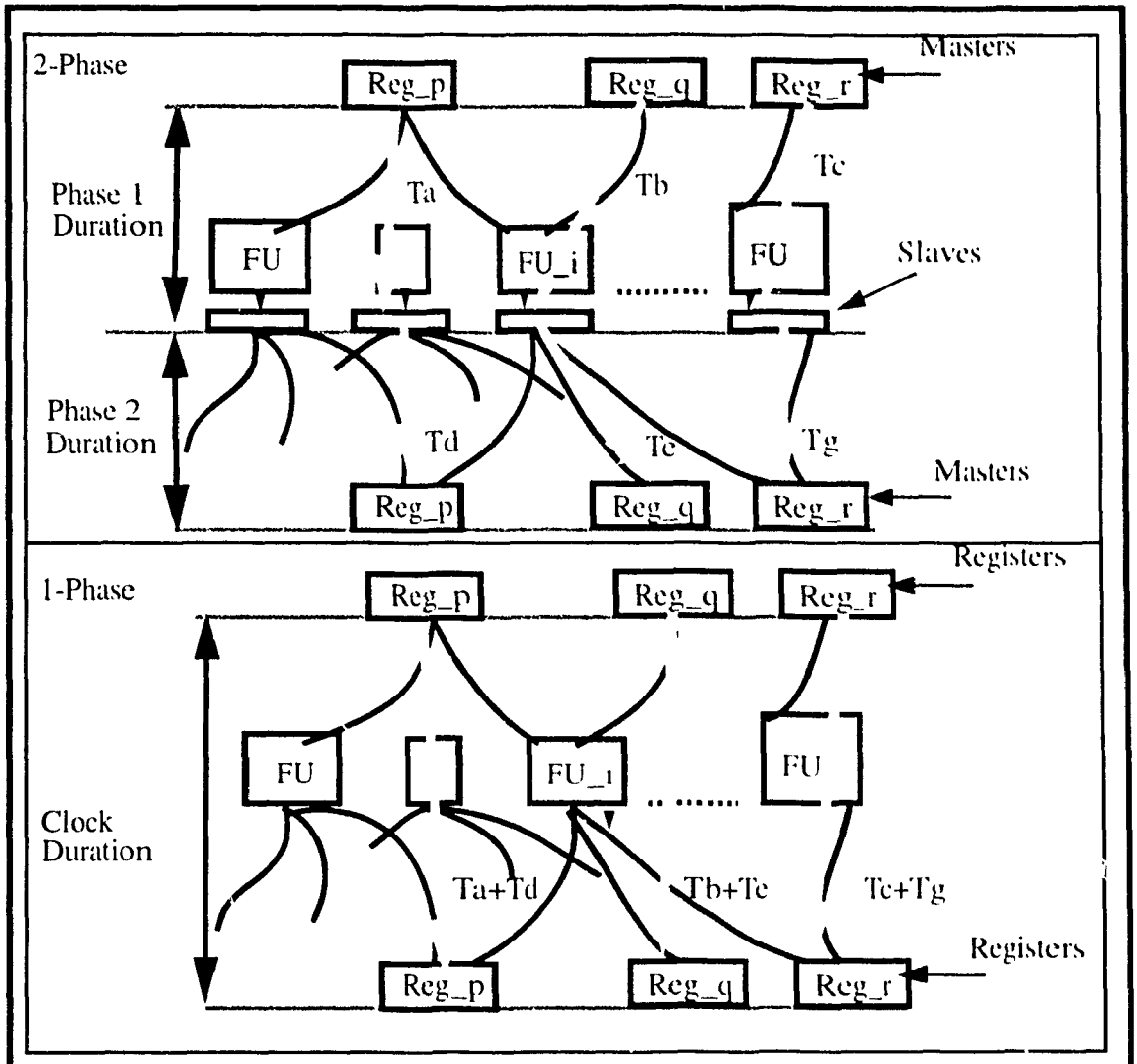


Figure 4.2: shows a generic example of some delays in a data path for a 2-phase and a single phase implementation. To see that a two phase clocking scheme has potentially a worse clock cycle duration than single phase duration for a data path, we make the following assumptions: Ta>Tb, Ta>Tc, Tg>Td, Tg>Te, Tb+Te>Ta+Td, Tb+Te>Tc+Tg. By analysing the critical paths we find that: The cycle duration for the 2-phase case; T2phase=Ta+Tg, and for the single phase is, T1Phase=Tb+Te. Since Ta>Tb and Tg>Te therefore: T2phase> T1phase. We assumed here that the delay of the latches and registers are negligible relative to the FU, multiplexer and routing delays (if a delay of register is close to the delay of two latches then our arguments are still valid).

4.2.2 Structural Complexity Criteria:

Again, we delay the register binding to a phase where detailed placement and routing of the final architecture is performed. This delayed binding of storage, makes it essential to account for interconnection and storage cost while doing scheduling and binding. Accounting for interconnections is done through the use of the *structural complexity criterion*. A special formulation for this *criterion* was initially presented in Chapter 3 for non-partitioned systems. In this and the following sections a re-formulation of the structural complexity criterion for partitioned systems is introduced.

The first step in optimizing the structural complexity of a partition system is to minimize the number of motifs or interconnections. As we explained earlier a motif in the graph is an item representing all edges that have the same FU bound to its source operation and the same FU bound to its destination operation. *By minimizing the number of motifs used, we are minimizing the interconnects, and we are also maximizing the use of these connections by the data transfers.*

However, for partitioned systems, we have introduced two types of motifs:

1) **Intra-partition connection** which is a direct path for data from a function unit to another function unit within the same partition. This type of motif accurately counts the number of interconnects within any partition.

2) **Inter-partition connection** represents the direct data path for data from any function unit to the port or from the port to a function unit. Inter-partition motifs are used for accounting the pin and interconnection limitation between partitions.

The next step in optimizing the structural complexity of a partitioned system is to minimize the number of ports and binding them efficiently. Our approach accounts for two types of ports as shown in Figure 4.3: 1)Bi-directional ports, where a data transfer from one partition to another partition is achieved through a common bus. 2)Uni-directional ports, which contain two buses for each direction of the data transfers.

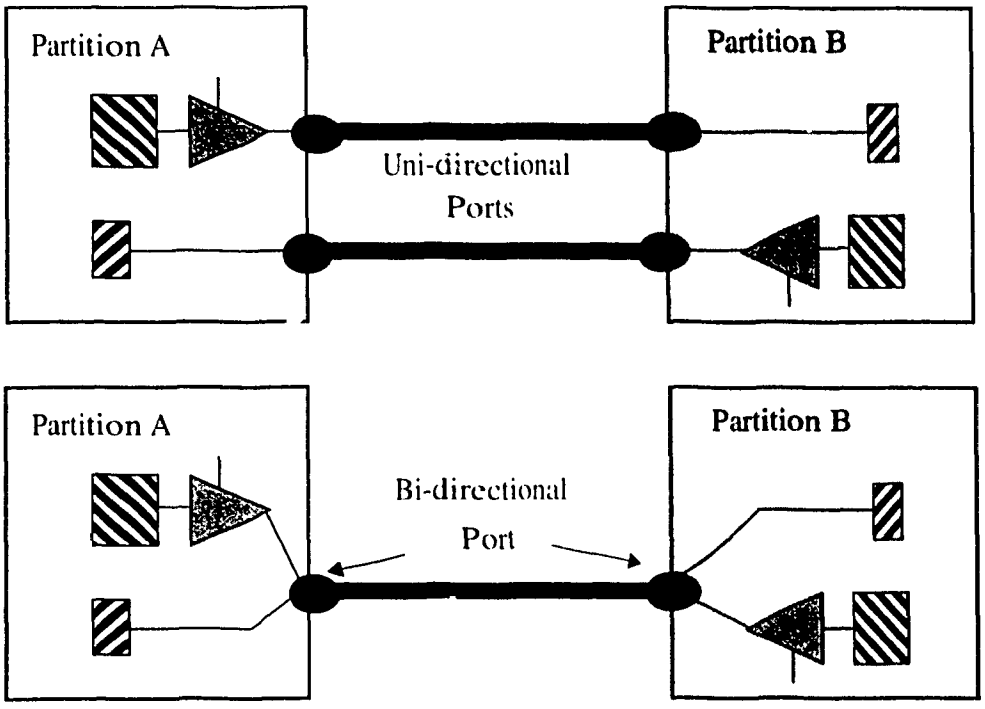


Figure 4.3: The configurations of Bi-directional and Uni-directional ports.

By considering each port as a FU, extra motif instances (inter-partition motifs) are created with a port as one of its nodes (source or termination). The maximum number of ports is $Nport^q$ (for partition q) and ports are indexed from 1 to $Nport^q$. Inter-partition motifs associated with each port are weighted differently. The higher the port index the higher the weight. By minimizing the total weight of the inter-partition motifs we minimize two things: 1) the interconnections internal to the partitions that connect other FUs to the ports and, 2) minimize the number of ports used.

Therefore, a **structured data path architecture** will result from a binding of the data transfer edges to motifs that **minimizes a weighted sum of** the number of intra-partition motifs, and the sum of the inter-partition motif weights. This sum is the "**structural complexity**" measure for partitioned systems.

4.3 ILP Formulation for an Optimal Architecture

The objective is again to minimize a weighted combination of the total execution time and the "**structural complexity**" measure. The notations used along with the list of variables are summarized in Figure 3.1 and Figure 3.1 respectively

Fu_t: Different type function units used in CDFG.

$FU_{(port)}$: function unit of type port

N_t: Max. number of FU type t allowed at each cycle.

Op: Set of all operations.

W: Set of all operations without successors.

Range(op): the cycles in which the operation can be scheduled.

Part(op): the partition in which the operation can be scheduled.

D(op): Delay of each operation.

L(op): Latency of operation on a FU.

Op(i)<Op(j): precedence relation; Op(i) should occur before Op(j).

$Op(i) \rightarrow Op(j)$ **:** edge with Op(i) as source and Op(j) as destination.

Figure 4.4: Notations used in our formulation.

$\mathbf{X_{(op,n,s,q)}}$: =1 only if operation "op" is assigned to function unit "n" and is scheduled into cycle "s", in the partition "q", otherwise = 0.

$\mathbf{IntraPmotif_{(q,Fut,\ n,q,\ Fut',\ n')}}$: =1, if there is a direct path for data from function unit "n" to function unit "n'" within the same partition, otherwise = 0.

$\mathbf{InterPmotif_{(q,Fut,\ n,q',\ Fut',\ n')}}$: =1, if there is a direct path for data from function unit "n" to the port "n'" or from the port "n" to the function unit "n'", otherwise = 0.

$\mathbf{K_{(Fu(port),n)}}$: weights associated to each motif with port as its source or destination.

$\mathbf{FaninPort_{(op,n)}}$: Integer variables representing the maximum number of fan in for O/P Ports.

Figure 4.5: Integer Variables used in our formulation.

Our formulation consists of a number of constraints and an objective function. The first 4 constraints are similar to the ones presented in chapter 3 with an additional index on the variables. This index belongs to the set "Part(op)" which represents the partition where the operation (op) is scheduled.

Equation (4.1) ensures that each operation within a partition is only assigned to one control step and one function unit.

$$\sum_{s \in Range(op)} \sum_{n=1}^{Nt} X_{op,n,s,q} = 1 \qquad \forall op \qquad q \in part(op) \qquad \textbf{(4.1)}$$

The second constraint guarantees that within any partition at every cycle, only one

operation is being assigned to each function unit.

$$\sum_{p=s}^{s-L(op)+1} \sum_{\substack{op \in Fu_t \\ q = part(op)}} X_{op,n,p,q} \leq 1 \qquad \forall s \qquad \forall q \qquad \forall qn \qquad (4.2)$$

The next inequality is used to maintain the precedence relationship between the operations. This prevents an operation from being assigned after another operation if an actual or virtual edge exists between these operations.

$$\sum_{p=s-D(op_i)+1}^{ALAP(op_i)} \sum_{n=1}^{Nti} X_{op_i,n,p,q} + \sum_{p=ASAP(op_j)}^{s} \sum_{n=1}^{Ntj} X_{op_j,n,p,q'} \leq 1 \qquad (4.3)$$

$$op(i) < op(j), \forall (i,j) \qquad q = part(op_i) \qquad q' = part(op_j)$$
$$ALAP(op_i) + D(op_i) - 1 \geq \forall s \geq ASAP(op_j)$$

Constraint (4.4) ensures that no operation is scheduled after CStep.

$$\sum_{s \in Range(op)} \sum_{n=1}^{Nt} s \times X_{op,n,s,q} - CStep \leq (-D(op) + 1) \qquad \begin{matrix} \forall op \in W \\ q \in part(op) \end{matrix} \qquad (4.4)$$

The above 4 constraints ensure scheduling precedence and binding conflict resolutions. However, using only these four constraints in conjunction *with only* overlap minimization (similar to [10]), results in a number of equivalent optimal bindings that may eventually produce data paths with a high number of interconnects.

In our formulation, we add a transfer operation on all the edges with their source and destination nodes in different partitions. These transfer operations represent the ports of the partitions. The delay of the transfer operations "D(op)", can be used to set the delay of data-transfer between different partitions to some technology dependent value. In our example, we set "D(op)=1", which implies that at least one cycle is required to transfer data between partitions. Figure 4.6 illustrates how the two types of ports (uni or bi-

directional) can be supported by assigning the transfer operation in the source or destination partition.

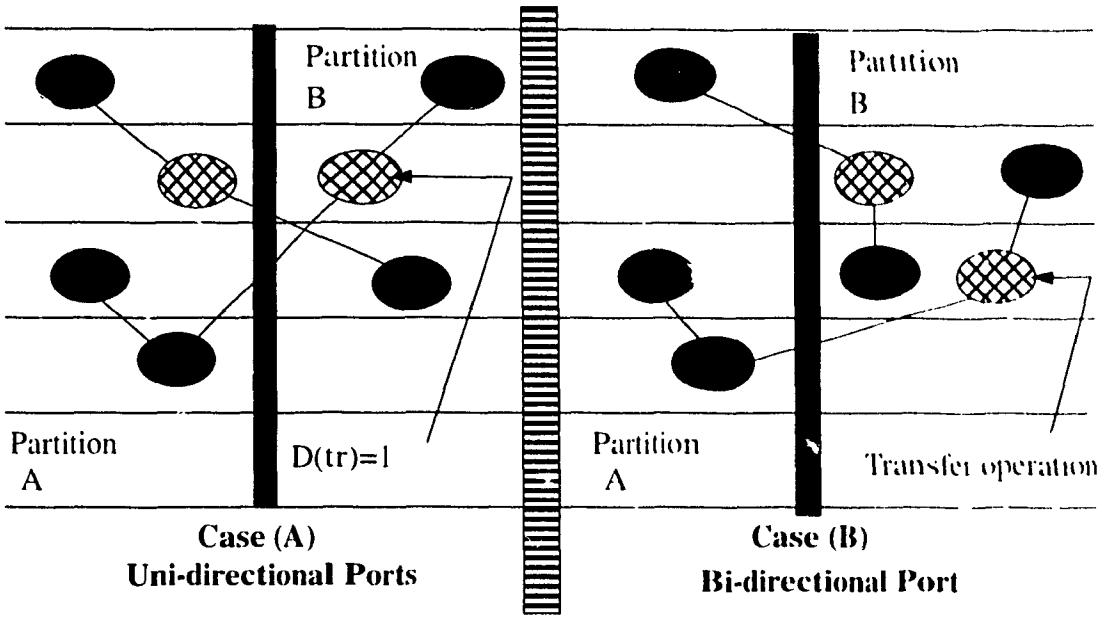


Figure 4.6: **Case A** (uni-directional ports) shows that transfer operations are assigned to the partition of the source data. **Case B** (bi-directional port) shows that the transfer operations are assigned in one of the partitions (it does not matter which, since both partitions are scheduled simultaneously).

Inequalities (4.5) and (4.6) are used to count for the two types of motifs. The first one accounts for the interconnections within a partition and the second inequality is used to count the data transfers between different partitions of a FPGA.

$$\sum_{\substack{s \in Range(op_i) \\ op_i \in Fu_i}} X_{op_i,n,s,q} + \sum_{\substack{s \in Range(op_j) \\ op_j \in Fu_{i'}}} X_{op_j,n,s,q} - IntraPmotif_{(q,Fu_i,n,q,Fu_{i'},n')} \leq 1 \qquad \textbf{(4.5)}$$

$$q = part(opi) = part(opj)$$

$$op_i \rightarrow op_j, \forall (i,j) \qquad \forall n\,(n = 1 \ldots Nt) \qquad \forall n'\,(n' = 1 \ldots Nt')$$

$$\sum_{\substack{s \in Range(op_i) \\ op_i \in Fu_i}} X_{op_i,n,s,q} + \sum_{\substack{s \in Range(op_j) \\ op_j \in Fu_i}} X_{op_j,n,s,q'} - InterPmotif_{(q,Fu_i,n,q',Fu_i,n')} \leq 1 \tag{4.6}$$

$$q = part(opi) \qquad q' = part(opj) \qquad q \neq q'$$
$$op_i \rightarrow op_j, \forall (i,j) \qquad \forall n\,(n = 1 \ldots Nt) \qquad \forall n'\,(n' = 1 \ldots Nt')$$

To minimize the maximum number of fan in to the output port (O/P), an integer variable $\mathbf{FaininPort_{(op,n)}}$ is introduced. This variable is set in constraint (4.7) to the maximum number of edges that overlap in time and have the same destination port. Minimization is done by including term 4 in the objective function, eqn. (4.8) which will be explained later. This variable $\mathbf{FaininPort_{(op,n)}}$ may also be constrained with an upper bound to limit the number of registers that are connected to any output port (typically this upper bound = 1).

$$\sum_{\substack{\forall (op_i \rightarrow op_j) \\ op_i \in FU_{(P,n)}}} \left(\sum_{p = ASAP(op_i)}^{s} \sum_{n=1}^{Nt} X_{op_i,n,p} + \sum_{p=s+1}^{ALAP(op_j)} X_{op_j,n',p} - 1 \right) - FaninPort_{(op_j,n')} \leq 0 \tag{4.7}$$

$$\forall n' \qquad \forall s$$

The objective function, eqn. (4.8), consists of 4 terms. The first term reduces the total execution time (Cstep). The second and third terms minimize the two different types of motifs stated above (IntraPmotif and InterPmotif). Inter-partition motifs associated with each port can be weighted differently. The higher the port index the higher the weight. Therefore, by minimizing the total weight of inter-partition motifs, we minimize two things:

1) The interconnections internal to the partitions that connect other function units to the ports.

2) Minimize the total number of ports needed. The last term in the objective function is used to minimize the maximum number of fan in to the O/P port.

Using an appropriate coefficient for each term in the objective function, we can minimize a weighted combination of the total execution time and the structural complexity.

Minimize *Objective Function* **(4.8)**

$$(C1 \times Cstep) \qquad term(1)$$

$$+\left(C2 \times \sum_{Fu_{in}=1}^{Nt} \sum_{Fu_{i'}} \sum_{n'=1}^{Nt'} \sum_{q} InterPmotif_{(q, Fu_i, n, q, Fu_{i'}, n')} \right) \qquad term(2)$$

$$+\left(C3 \times \sum_{Fu_{in}=1}^{Nt} \sum_{Fu_{i'} n'=1}^{Nt'} \sum_{q} \sum_{\substack{q' \\ q \neq q'}} K_{Fu(port),n} \times IntraPmotif_{(q, Fu_i, n, q', Fu_{i'}, n')} \right) \qquad term(3)$$

$$+\left(C4 \times \sum_{\substack{n'=1 \\ op_j \in FU_{(Port}}}^{Ntj} FaninPort_{(op_j, n')} \right) \qquad term(4)$$

For multiple FPGA boards with specific structure of interconnections, the number of ports and their types are known per FPGA. This port structure translates to our formulation by restricting the number and type of functional units representing the ports per partition.

For example, Figure 4.7 shows one example of interconnection configuration that is supported by our methodology.

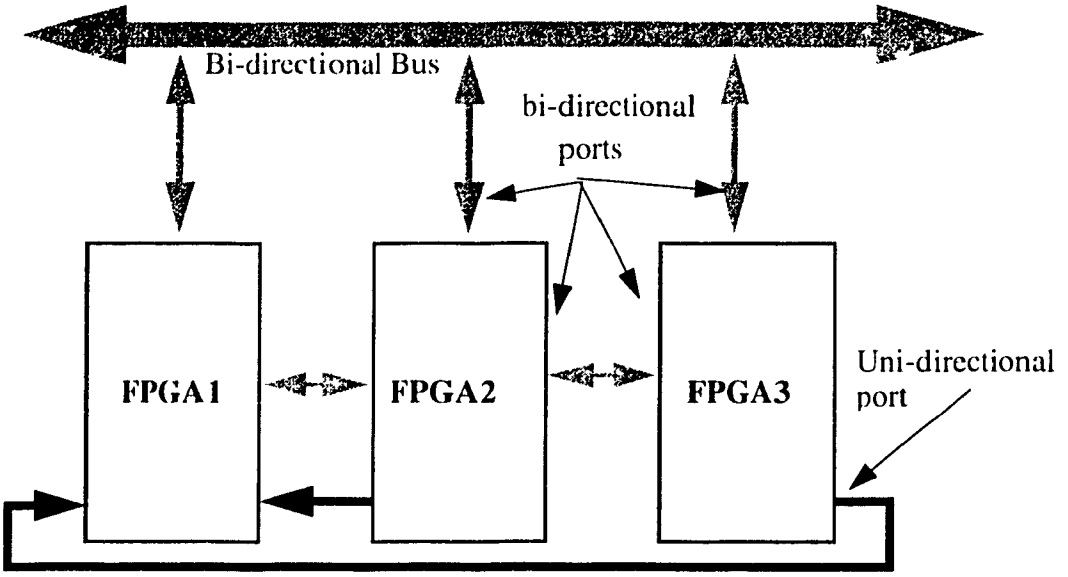


Figure 4.7: Example of Multiple FPGA interconnection structure supported by our ILP formulation. Both bi-directional buses and unidirectional connections are shown.

4.4 Synthesis Results

Since most FPGAs would not support a large number of function units (especially multipliers), it is a common approach to partition the algorithm to be executed on a number of partitions. The partitioning simplifies the complexity of the data path design and therefore, reduces the size of the search per FPGA. Each partition is restricted to have few functional units (<8). Using our formulation, we still account for the inter-partition scheduling dependencies and I/O port restrictions. Therefore, all of the operations in all partitions are being considered in parallel with a restriction of being confined within their pre-assigned FPGA and its function units.

4.4.1 Discrete / Inverse Discrete Cosine Transform

We use the Discrete Cosine Transform (partitioned into 3 subgraphs to demonstrate a

number of points regarding our approach. Table 4.1 represents the various synthesis results for both DCT and also the IDCT (Inverse Discrete Cosine transform). Because the number of operations and FUs per partition is small the running time is reasonable. However, the running time is still dependent on the optimization criteria (OPTCR, in CPLEX). This time increases significantly for highly parallel algorithms, when the upper bound on CSTEP is much larger than the critical path and when the total number of functional units is large (>8). Good heuristic solutions can be used for obtaining such bounds.

Table 4.1: Synthesis Results of Partitioned DCT and IDCT

		DCT			IDCT		
Different Outcomes		1	2	3	4	5	6
# of ALUs per partition	Partition 1	2	2	2	2	2	3
	Partition 2	2	2	2	2	2	2
	Partition 3	2	2	2	2	1	2
# of multipliers per partition	Partition 1	1	1	1	1	1	2
	Partition 2	1	2	2	1	2	2
	Partition 3	1	1	2	1	2	2
# of ports per partition	Partition 1	0	0	0	2	2	2
	Partition 2	1	1	2	2	2	2
	Partition 3	1	1	1	1	1	1
total # of motifs between partitions		2	2	4	5	5	7
total # of motifs in all partitions (including motifs to I/O ports)		19	18	25	17	19	29
# of cycles		9	9	8	10	9	8
OPTCR (Optimization Criteria) (0.006 = 0.6% of optimal solution)		proven optimal	proven optimal	0.006	0.005	0.005	0.01
Solution time (sec.)		82.6	124	13.3	4.9	107	26.2
C1=8, C2=1, C3=40, C4=0							

Figure 4.8 shows the scheduling and binding for column 1 of Table 4.1. It is difficult for

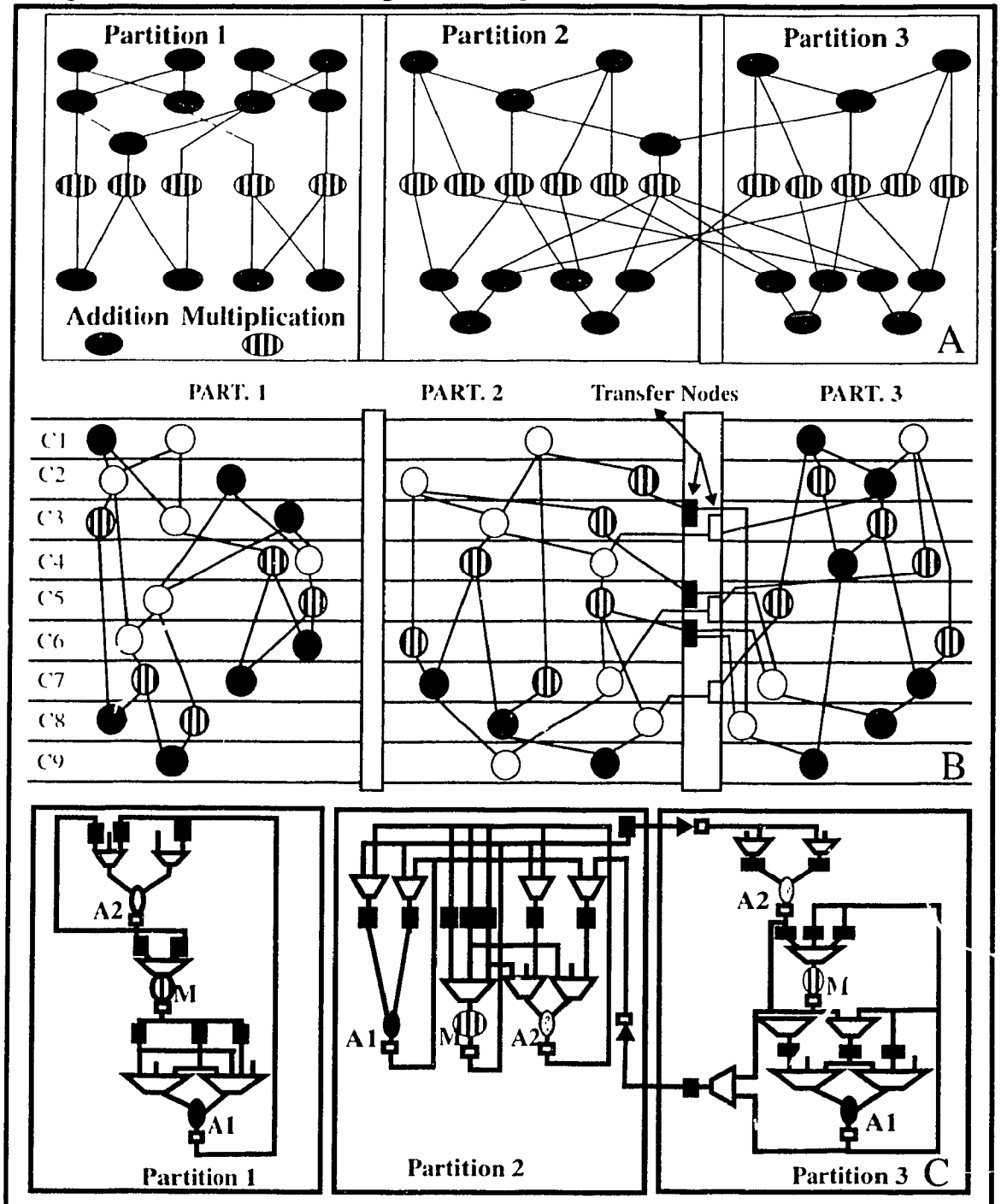


Figure 4.8: A) CDFG of DCT, B) Schedule and FUs Allocation for the 3 Partitions and C) The data paths for each partition. Note that partition 1 and 3 use less motifs than partition 2 and hence have simpler data path structures. Also the I/O registers can be used to store the data for more than one cycle. This example demonstrates the two phase architecture model.

us to compare architectural features with other optimal ILP formulations as their final architectures are not published. In Figure 4.8 we show the detailed results of our scheduling, binding and the final architecture for the DCT example as obtained by the tool described in [32]. Note that more than one architecture can be produced based on the optimization criteria for the final data path depending on clock cycle duration or the area being the primary objectives. The results shown in Figure 4.8 are for an equal weighting of area and clock duration.

For an even larger example, we used our approach to synthesize a single multi-FPGA architecture to execute both DCT and IDCT sequentially. In essence, this architecture can mutually exclusively execute both algorithms. The resulting configuration used two adders, one multiplier and one uni-directional port for each of the 3 partitions. For this example, we obtained a total of 24 motifs within all partitions and 8 motifs between different partitions in 19 cycles (proven optimal within 2% (OPTCR=0.02) of optimal cost). This implies that both DCT and IDCT can be supported on the same multiple FPGA (in our example, three FPGAs) with minimal structural complexity.

4.4.2 Elliptic Filter

For elliptic filter (partitioned to two FPGAs), we obtained the same number of cycles (Te) as reported in [33] (using 1 adder and 1 multiplier per partition). Our solution used only one bi-directional port, while [33] uses one bus and nine TIO pins. This example is small and not sufficiently rich for comparison of FU bindings performed on partitioned architectures. There are no large examples published, to our knowledge, with enough details of the architectures that support both scheduling and binding of FU, I/O ports as well as minimize interconnects concurrently and optimally on all partitions.

4.5 Chapter Summery

In this chapter, an architecture model suitable for multiple FPGA implementations of partitioned signal processing data paths was presented. We also introduced an optimal ILP approach to concurrently synthesize, on each partitions of the system, structured data paths with minimum interconnections and I/O ports. Moreover, this chapter expands the "structural complexity" criterion to partitioned systems and introduces the scheduling and operation binding formulation to produce structured partitioned architectures.

Chapter 5: ILP Synthesis of Signal Processing Structured Data paths for FPGAs Supporting RAMs and Buses

5.1 Introduction

This chapter presents an integer linear programming formulation for optimally mapping a signal processing algorithm on a BUS/RAM based FPGA based on the bus model presented in chapter 1.

This technique assumes that the following synthesis bindings are initially performed: simultaneous operation scheduling and function unit allocation and binding while minimizing the structural complexity of the interconnections, storage and multiplexing using a modified ILP formulation for multiplexer based FPGAs. The results from this formulation is then fed to another ILP formulation which optimally selects and assigns data transfers to buses and at the same time schedules the bus transfers to minimize the use of the buses, bus loading and data storage for both registers and register files.

The plan of this chapter is as follows. Section 5.2 is a brief overview of data transfer model along with modified structural complexity measure for a bus based architecture. Section 5.3 presents the ILP formulation for bus transfers. Results of our formulation on a number of examples are shown in section 5.4 and finally conclusions are drawn in 5.5.

5.2 Data Transfer Model For Bus Based Architecture

Given the architectural model presented in chapter 1, there are a number of different cases for which an edge in the Data Flow Graph (a data transfer) can be bound. These cases are encompassed by the two generic transfers shown in Figure 5.1. Notice that the bus in Figure 1.4. (b or c) (see also Figure 5.6.b) communicates only with registers. In

Figure 5.1, a bubble indicates the clock cycles at which the data is transferred from a register (R_n) to the bus and the square indicates the cycle where the data is read from the register file through a bus to a register (R_p). These cycles are determined in step-2 of Figure 2.2, together with bus allocation. In a sense, a scheduling of bus transfers is performed at step-2. *This flexibility of scheduling bus transfer as shown in Figure 5.1 is*

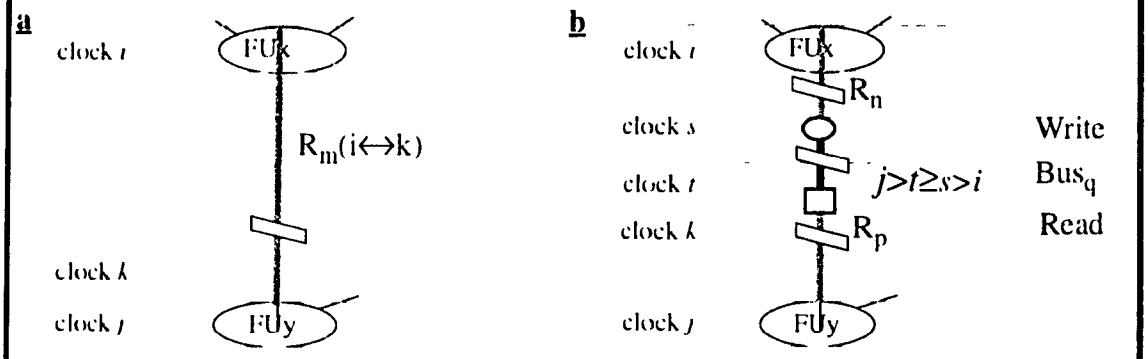


Figure 5.1: (a) Shows a data transfer using local interconnections through individual register Rn. (b) Shows a data transfer using a pipelined bus. Rn and Rp do not have to be different, but have to have at least a life time of one cycle. The variable is stored in a register file between clock cycle s and k through Bus_q.

very different from all previous architecture models in the literature using busses. All other models have to transfer data immediately at the end of an operation, or upon its start. Our model allows relaxing this constraint which is one of the contributions of this paper. This relaxation in using busses can result in significant reduction in the number of busses required.

5.2.1 Modified Structural Complexity Criteria

We assume that operation scheduling and binding have been performed (step-1, Figure 2.2) while minimizing interconnections and maximizing the number of transfers with life-time > 1 using the structural complexity measure as described in chapter 2.

Since bus binding is relegated a separate step (step-2 Figure 2.2), we would like to incorporate a mechanism in "structuring" to account for the possibility of a data transfer edge being re-assigned from an initial motif instance (hence local interconnection) to a bus in step-2.

If all the edges that are assigned to a motif instance are re-assigned to a bus in step-2, a local interconnection is eliminated. When this happens, we say a motif instance *has merg*ed into a bus. For pipelined busses, we know that a data transfer occurring between operations that are one cycle apart cannot be done on a bus (Figure 1.4.b). Therefore, a motif instances can be merged into a pipelined bus (or ***mergeable***) iff all the data transfers assigned to this motif instance have a life time > one cycle.

A motif instance is said to be ***non-mergeable*** if one or more data transfers with life-time of one cycle have been assigned to it. Hence, to reduce local interconnections by merging them into pipelined busses, we have to *decrease the number of non-mergeable motif instances.*

The remaining criterias presented in Chapter 3 are still valid. Minimization of local interconnections is achieved by minimizing the total number of motifs. Another objective is to minimize *the number of incompatible motifs* as a cost estimate of multiplexors as it was explained previously. And finally, by reducing the total overlap of data transfers, we are also minimizing the lower bound on the number of registers to be used. But this is not accurate since not all data variables are going to be stored in a registers, some are going to be stored in a cheaper (less area or #CLB) register files or RAMs. Since only the variables with life-time > 1 are eligible for register file storage the cost function should account for the different cost of storage by a register and a possible storage in a register file. It is required then to minimize the maximum overlap of variables in all cycles. The overlap is computed with a suitable cost per data variable. This formulation will be explained in more details in the following section.

Hence, from the above properties of motif instances, we propose here a criteria (for the first step) that will result in more "structured" or regular architecture. A **structured architecture** will result from a binding of the data transfer edges to motif instances that uses the **least** number of: motif instances, overlaps in data transfers assigned to a motif instance, incompatible motif instances and non-mergeable motif instances. Minimizing "structural complexity" is called **"Structuring"**.

Hence, the "**structural complexity**" measure of step (1) should be modified to a weighted sum of four terms: The first term is the sum of the number of overlapping edges assigned to each motif instance. The second term is the number of different motif instances. The third is the number of *incompatible* motif instances. The fourth term is the number of *non-mergeable* motifs.

The above measures provide us a structured architecture which can be easily adapted to support RAMs and Busses. Then, the ILP formulation of the second step optimally selects and assigns data-transfers to busses *while scheduling* the bus transfers to minimize the use of the busses, bus loading and data storage for both registers and register files [5]. This formulation will be explained in more details in the following section.

5.3 ILP Formulation for an Optimal Architecture

5.3.1 Modified ILP Formulation of Step 1

The formulation for combined scheduling and binding of operations while minimizing the number of motif instances have been presented previously in Chapter 3. We here concentrate on two components of the structural complexity, which is the formulation for non-mergeable motifs and the maximum overlap which are adapted to account for bus based architectures. The other two components (number of motifs and incompatible motifs) of the structural complexity remain essentially the same as before. The additional

integer variable are presented in Figure 5.2.

Nonmerg$_{(Fut, n, Fut', n')}$: = 1, if motif(Fut, n, Fut', n') is non-mergeable.

Figure 5.2: Additional Integer Variable used in our formulation of step 1.

A-The non-mergeable motifs: We count the number of motif instances that have any data transfers assigned to them with life-time = 1. Modified equation 3.7A is used to count the number of non-mergeable motifs,

$$X_{op_i,n,s} + X_{op_j,n',(s+D(op_i))} - Nonmerg_{(Fu_i,n,Fu_{i'},n')} \leq 1$$
$$\forall (op_i \rightarrow op_j) \qquad \forall n\,(n = 1 \ldots Nt) \qquad op_i \in Fu_i \qquad (3.7A)$$
$$\forall (s \in Range(op_i)) \qquad \forall n'\,(n' = 1 \ldots Nt') \qquad op_j \in Fu_{i'}$$

B-The maximum overlap: We approximate the different costs of storage between registers and register files. The formulation presented is not exact but suffices for discriminating between two schedules with the same cost for all other cost components other than register overlap. We count the storage cost for each data variable as (1) for a data transfers of lifetime = 1 cycle. For the data variables of life time of two or more cycles, a storage cost of (1-K2) at the cycle of the source operation and at the cycle preceding the destination operation is used. The variables in these cycles use a master register for storage. For all other cycles the storage cost per data variable is K2 to account for the possibility of that variable being stored in a register file. Where K2 <1 and is usually small (~0.1).

$$K2 \times \sum_{\substack{\forall (op_i \rightarrow op_j) \\ op_j \in Fu_i}} \left(\sum_{\substack{p = \\ ASAP(op_i)}}^{s} \sum_{n=1}^{Nti} X_{op_i,n,p} - \sum_{\substack{p = \\ ASAP(op_j)}}^{s} \sum_{n=1}^{Ntj} X_{op_j,n,p} \right) \qquad (3.8A)$$

$$+ \left((1-K2) \times \sum_{n=1}^{Nti} X_{op_i,n,s} \right) + \left((1-K2) \times \sum_{n=1}^{Ntj} X_{op_j,n,(s+1)} \right) - Maxovlap_{(Fu_i)} \leq 0$$

$$\forall s \qquad \forall Fu_i$$

Now, the objective function of step (1) is modified to account for the non-mergeable and overlap measures as shown below:

Minimize *Objective Function* (**3.9***A*)

$$(C1 \times Cstep) \qquad term(1)$$

$$+\left(C2 \times \sum_{Fu_t,n=1}^{Nt} \sum_{Fu_{t'},n'=1}^{Nt'} Motif_{(Fu_t,n,Fu_{t'},n')} \right) \qquad term(2)$$

$$+\left(C3 \times \sum_{Fu_t} Incomp_{(Fu_t)} \right) \qquad term(3)$$

$$+\left(C4 \times \sum_{Fu_t,n=1}^{Nt} \sum_{Fu_{t'},n'=1}^{Nt'} Nonmerg_{(Fu_t,n,Fu_{t'},n')} \right) \qquad term(4)$$

$$+\left(C5 \times \sum_{Fu_t} Maxovlap_{(Fu_t)} \right) \qquad term(5)$$

5.3.2 ILP Formulation of Step 2

The objective of step-2 is to minimize: (a) the number of parallel bus transfers which reduces the number of buses allocated, (b) the maximum overlap in registers which reduces the number of registers allocated, (c) the maximum overlap in life-times of variables assigned to buses which reduces register file storage locations, (d) the number of registers having tri-state access to the buses hence minimizing tri-state output loading on each bus and (e) the number of destination registers that a bus is connected to, this also minimizes bus loading. Note that in step-2, no register binding is made, only the scheduling of the bus transfers, selecting which transfer goes on a bus and which bus that transfer uses.

The notations and variables are summarized in Figure 5.3 and Figure 5.4 [3].

E: set of edges with lifetime >= 2cycles. **N_B:** Max. number of buses allowed.

D(Bus): delay of the bus. **EdgeOverlap(s):** Set of edges that overlap at each cycle.

Multoutput: Set of operations with multiple edges at their output.

Single: Set of edges that do not belong to Multiple output operations.

Range(e_X): The cycles in which the Z-O var "x" exists ("x" could be RB or WB or...).

Figure 5.3: Notations used in the ILP formulation of bus scheduling and binding.

$WB_{(e,s,n)}$: =1 only if clock cycle "s" is used to write the variable (edge) "e" in the register file through bus "n", otherwise = 0.

$RB_{(e,s,n)}$: =1 only if clock cycle "s" is used to read the variable (edge) "e" from the register file through bus "n", otherwise = 0.

$WBM_{(op,s,n)}$: =1 only if clock cycle "s" is used to write any of the variables (edges) at the output of operation "op" in the register file through bus "n", otherwise = 0.

$RBM_{(op,s,n)}$: =1 only if clock cycle "s" is used to read any of the variables (edges) at the output of operation "op" from the register file through bus "n", otherwise = 0.

$Regused_{(op,s)}$: =1 only if any of the edges at the output of operation "op" is alive at clock cycle "s", otherwise = 0.

MinReg: An integer variable used to count the lower bound on the number of registers needed to implement the architecture.

TotReg: An integer variable used to count the maximum lifetimes of all the registers.

$MaxRegFile_n$: an integer variable used to count the maximum required number of RAM locations per register file.

Tristate: The number of tristate drivers for every bus is calculated and the maximum of them is assigned to this integer variable.

Busload: The number of mux inputs connected to every bus is calculated and the maximum of them is assigned to this integer variable.

Figure 5.4: Integer var. used in the ILP formulation of bus scheduling and binding.

The first constraint ensures that only one clock cycle is used to write the variable in the register file through one bus.

$$\sum_{n=1}^{N_B} \sum_{s \in Range(e_{WB})} WB_{e,s,n} \leq 1 \qquad \forall e \in E \tag{5.1}$$

The next constraint guarantees that only one cycle is used to read the variable back from the register file through one (same or another) bus.

$$\sum_{n=1}^{N_B} \sum_{s \in Range(e_{RB})} RB_{e,s,n} \leq 1 \qquad \forall e \in E \tag{5.2}$$

Constraint (5.3) ensures that every variable assigned to a bus transfer is written in and read from the same register file through one bus. Without constraint 5.3, use of multi-ported register files are enabled where one variable is written through one port (from one bus) and read from another port (and another bus).

$$\sum_{s \in Range(e_{WB})} WB_{e,s,n} \sum_{s \in Range(e_{RB})} RB_{e,s,n} = 0 \qquad \begin{matrix} \forall e \in E \\ \forall n = 1 \ldots N_B \end{matrix} \tag{5.3}$$

Inequality 5.4 ensures that a read occurs after a write to a register file for any variable.

$$\sum_{p=s-D(Bus)+1}^{ALAP(e_{WB})} \sum_{n=1}^{N_B} WB_{e,p,n} + \sum_{p=ASAP(e_{RB})}^{s} \sum_{n=1}^{N_B} RB_{e,p,n} \leq 1 \tag{5.4}$$

$$ALAP(e_{WB}) + D(Bus) - 1 \geq s \geq ASAP(e_{RB}) \qquad \forall s \qquad \forall e \in E$$

Constraint 5.5 and 5.6 are intended for operations with multiple edges at its output. We assign a new set of zero-one (0-1) variables (WBM, RBM) for these output edges (as shown in Figure 5.5). These 0-1 variables are forced to "1" if there is a read from a register file or a write to a register file at any cycle for any of the multiple output edges. For

instance in Figure 5.5, $WBM_{1,2}$ is set to "1" when at least one of the variables $WB_{1,2}$, $WB_{2,2}$, $WB_{2,2}$ is equal "1". The same argument applies to the RB and RBM variables.

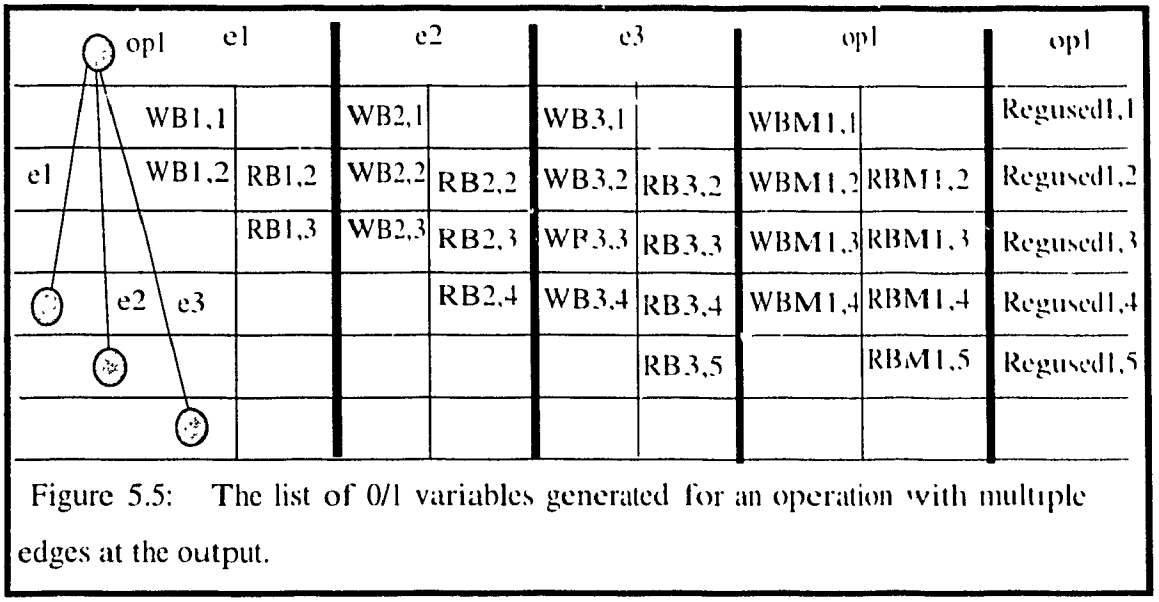


Figure 5.5: The list of 0/1 variables generated for an operation with multiple edges at the output.

$$WB_{e,s,n} - WBM_{op,s,n} \le 0 \qquad \forall op \in Multoutput \quad \forall e \in op \quad \forall n \quad \forall s \in Range(e_{WB}) \qquad (5.5)$$

$$RB_{e,s,n} - RBM_{op,s,n} \le 0 \qquad \forall op \in Multoutput \quad \forall e \in op \quad \forall n \quad \forall s \in Range(e_{RB}) \qquad (5.6)$$

To ensure that at every cycle only one data variable can be read and only one data variable written through any one specific bus, we enforce constraints 5.7 and 5.8. In these constraints we use WB and RB as 0-1 variables for edges that do not belong to multiple output operations and WBM and RBM as 0-1 variables for the edges of the multiple output operations.

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} WB_{e,s,n} + \sum_{op \in Multoutput} WBM_{op,s,n} \le 1 \qquad \forall n \quad \forall s \qquad (5.7)$$

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} RB_{e,s,n} + \sum_{op \in Multoutput} RBM_{op,s,n} \le 1 \qquad \forall n \quad \forall s \qquad (5.8)$$

For a multi-ported register file data path, equations 5.7 and 5.8 are replaced by equation 5.9 to ensure that either a read or a write is done on one bus per cycle.

$$\sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} WB_{e,s,n} + \sum_{op \in Multoutput} WBM_{op,s,n} + \sum_{\substack{e \in Single \\ \wedge EdgeOverlap(s)}} RB_{e,s,n} + \sum_{op \in Multoutput} RBM_{op,s,n} \leq 1 \qquad \forall n \quad \forall s \qquad (5.9)$$

To account for the register cost in the cost function, edges of multiple output operations are assigned a set of 0-1 variables ($Regused_{op,s}$) per cycle "s". These 0-1 variables are set to "1" using constraint 5.10, when any of the multiple edges for an operation is alive in cycle "s".

$$1 - \sum_{n=1}^{N_B} \sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} + \sum_{n=1}^{N_B} \sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n} - Regused_{op,s} \leq 0 \qquad (5.10)$$

$$\forall op \in Multoutput \qquad \forall e \in op$$
$$\forall s \in (Range(e_{WB}) \cup Range(e_{RB}))$$

Constraint 5.11 determines a lower bound (integer variable MinReg) on the number of registers that can be assigned in step-3 of the synthesis tool.

$$\sum_{\substack{e \in \\ Single \\ \wedge EdgeOverlap(s)}} \left(1 - \sum_{n=1}^{N_B} \sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n} + \sum_{n=1}^{N_B} \sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n} \right) + \sum_{\substack{op \in \\ Multoutput}} Regused_{op,s} + reg_s - MinReg \leq 1 \qquad (5.11)$$

$$\forall s$$

Constraint 5.12 indirectly ensures that the total register life-time is reduced.

$$\sum_{e}\left(1-\sum_{n=1}^{N_R}\sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n}+\sum_{n=1}^{N_R}\sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n}\right)-TotReg\le 1 \qquad (5.12)$$

Constraint 5.13 is implemented to count the maximum required number of RAM locations per register file. Integer variable "MaxRegFile" is set to this maximum value.

$$\sum_{e\in EdgeOverlap(s)}\left(\sum_{p=Asap(e_{WB})}^{s} WB_{e,p,n}-\sum_{p=Asap(e_{RB})}^{s-1} RB_{e,p,n}\right)-MaxRegFile_n\le 0 \qquad (5.13)$$

$$\forall s \qquad \forall n$$

For all the busses, constraints 5.14 and 5.15 count the maximum bus loading due to tri-state drivers and mux connections per bus. Integer variable "Tristate" is set the maximum number of tri-states that is needed for every bus. Integer variable Busload performs the same for mux connections.

$$\sum_{e\in Single\ \wedge EdgeOverlap(s)}\sum_{p=s}^{Alap(e_{WB})} WB_{e,p,n}+\sum_{op\in Multioutput}\sum_{p=s}^{Alap(op_{WBM})} WBM_{op,s,n}\le Tristate$$

$$\forall n \qquad \forall s \qquad (5.14)$$

$$\sum_{e\in Single\ \wedge EdgeOverlap(s)}\sum_{p=Asap(e_{RB})}^{s} RB_{e,p,n}+\sum_{op\in Multioutput}\sum_{p=Asap(e_{RBM})}^{s} RBM_{op,s,n}\le Busload$$

$$\forall n \qquad \forall s \qquad (5.15)$$

The objective function to be minimized has eight components (Equation 5.16, next page). The first five terms contribute directly to the final data path structure. The last three

terms are essential for the correct assignments of the 0/1 variables in the ILP formulation.

Minimize *Objective Function* **(5.16)**

$$(C1 \times MinReg) \quad \text{(Term 1)}$$

$$+ (C2 \times TotReg) \quad \text{(Term 2)}$$

$$+ \left(C3 \times \sum_{n=1}^{N_R} MaxRegFile_n \right) \quad \text{(Term 3)}$$

$$+ (C4 \times Tristate) \quad \text{(Term 4)}$$

$$+ (C4 \times Busload) \quad \text{(Term 5)}$$

$$+ \left(C6 \times \sum_{op \in Multoutput} \sum_{s} \sum_{n=1}^{N_R} WBM_{op,s,n} \right) \quad \text{(Term 6)}$$

$$+ \left(C7 \times \sum_{op \in Multoutput} \sum_{s} \sum_{n=1}^{N_R} RBM_{op,s,n} \right) \quad \text{(Term 7)}$$

$$+ \left(C8 \times \sum_{op \in Multoutput} \sum_{s} Regused_{op,s} \right) \quad \text{(Term 8)}$$

This ILP formulation optimally selects and assigns data-transfers to busses *while scheduling* the bus transfers to minimize the use of the busses, bus loading and data storage for both registers and register files.

5.4 Register and Multiplexer Binding with Data path Generation

For the last step of our synthesis technique, we implemented the ILP tool presented in [22] to optimally minimize the wiring and multiplexer area. However, this ILP tool does not have the capability of accurately computing the total area and actual routing length and network delays.Morever, this tool does not satisfy the constraints imposed on routing resources such as those found in FPGAs.

Our synthesis results, for the final step of synthesis process are based on the tool reported in [32]. In summary, the tool used performs the register binding while at the same time performing a floorplanning of the data path to be able to compute the routing delay and area cost and its effect on the clock cycle duration. Such a tool produces better results than independently determining a register binding followed by a floorplanning. It uses a stochastic (simulated annealing) search while continuously minimizing the critical paths that determine the clock cycle for the data path together with layout area. This tool does not alter the number of functional units and their scheduling. The tool has combined two novel approaches: 1 - A placement & Routing model to handle different architectural topologies (mux. and/or bus based) for various technologies. 2 - An efficient formulation for the binding of register/Interconnect and combined Floorplanning

5.5 Synthesis Results

5.5.1 Elliptic Filter

We use the 5th order elliptic filter to demonstrate a number of points regarding our architecture features of pipelined buses, the operation binding using the structuring approach and our bus transfer scheduling. Figure 5.6 (a) shows a 2 adder one multiplier schedule with 17 cycles. The *operation* schedule is very similar to the one obtained by ALPS [17] (this filter is retimed). Our ILP tool, OSTA running on a SPARC10, produced a

scheduling and operation binding (step-1) in 21.3 CPU seconds. The bus scheduling and binding (step-2) took 1.7 CPU seconds. The detailed register binding was done in conjunction with floorplanning (step-3) and took 210 CPU seconds [32] (and 1700secs for ILP [22]). All results proven optimal. Note that only **one pipelined bus** is used compared to an optimal of **7 buses** for the OASIC[12] architectural model, and **4 buses** of the SPAID-X style architecture used in [24][20] which is equivalent of a 400% saving in the number of busses. Because these buses require global wires which are not abundant in FPGAs, such a saving which is a direct consequence of our approach is very significant in the routability of a data path.

To show how our model compares with others, we highlight a number of measures: number of busses, bus loading, number of networks with fan out >1 (indicates complex interconnections), number of tri-states, fan out of networks, and components of delay on the critical path. Table 5.1 compares different synthesized data paths for the EWF.

Table 5.1: Architecture for EWF. Only nets with fan out > 1 are counted. Our estimate for the number of CLBs for storage includes recursive storage. If recursive edges are added to the other solutions, up to an extra four registers or 2 CLB/bit of word length are needed. (#BC= number of Bus Connections as in [20])

2 Adders, 1 Pipelined Multiplier	#cycles	#tri-states (#BC)	#mux i/ps	#Bus (nets)	Registers (latches)
OASIC[24]	18	N/A	N/A	7(N/A)	9
InSyn [22]	19	N/A	N/A	4(N/A)	8+(5)
SPAID-X[24]	19	19	18	5(3)	-(21) **
IP [20]	19	12(23)	11	4(10)	(10)
Li & Mowchenko- LM1	19	6(12)	19	3 (8)	11
Li & Mowchenko- LM2	19	N/A	25	- (11)	11
STAR[22]	19	16(28)	17	5 (N/A)	13
OSTA (our tool)	17	2 (6)	22	1(7)	7**(6)

** Recursive edges have been implemented.

For the EWF architecture shown in [20], the maximum bus loading is 3 inputs and three tri-state drivers, which is the same for the architecture in Figure 5.6.

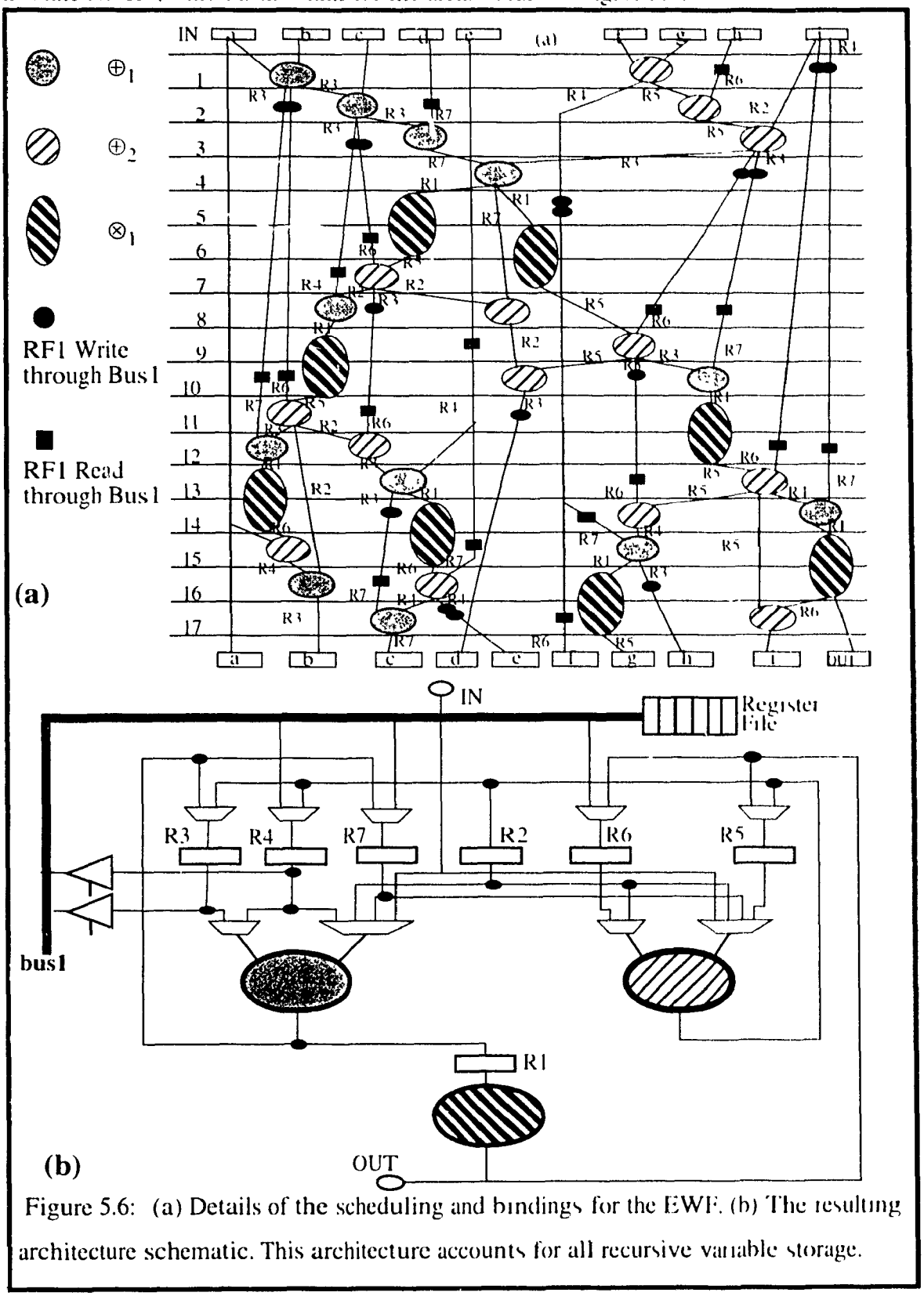


Figure 5.6: (a) Details of the scheduling and bindings for the EWF. (b) The resulting architecture schematic. This architecture accounts for all recursive variable storage.

Note that, in the architecture shown in [20], bus delays and RAM access time are added to the delay of a functional unit and registers to obtain the critical path determining the cycle time. While for the architecture in Figure 5.6, the RAM access and FU delay are independent as discussed before. It is evident that our architecture is simpler and uses less interconnections.

Regarding storage cost, our architecture uses a total of 7 registers and 6 register file locations (4 CLBs per bit of data path width), compared to an optimum of 9 registers for a mux based (without busses[1]) data path which is equivalent to using 4.5 CLBs per bit of data path width for storage. It is important to note that we include the cost of storage of all recursive variables (z^{-1} delays in the EWF filter specification) unlike almost all other published solutions (the 8 recursive variables b-i, of Figure 5.6 are not bound to storage in other references). The result by Li & Mowchenko [29] accounts for the recursive edges and uses 11 registers or 5.5 CLBs per bit of data path width for storage. This data path can either be viewed as a bus based architecture (LM-1) or a multiplexer based architecture (LM-2) in Table 5.1. We use less multiplexer inputs (> 12%), less CLBs for storage (>40%) and 300% less for the number of busses.

Our architecture has a maximum loading of three drivers on the bus and 3 outputs (mux inputs). The maximum fan out of any register output is 4. The maximum number of mux inputs is 4. These values are the best values for any published EWF architecture.

5.5.2 Cascaded-Elliptic Filter

An alternative example that requires significantly more storage and is suitable to highlight the advantages of RAM/bus based architectures is a filter composed of two elliptic filters connected in cascade (output of first is directly the input of the second).

The operation scheduling and binding (solution time 58 sec.) as well as the bus scheduling (solution time 33 sec.) are shown in Figure 5.7 (next page). For storing all

variables including the recursive edges the resulting data path uses 11 registers at a cost of

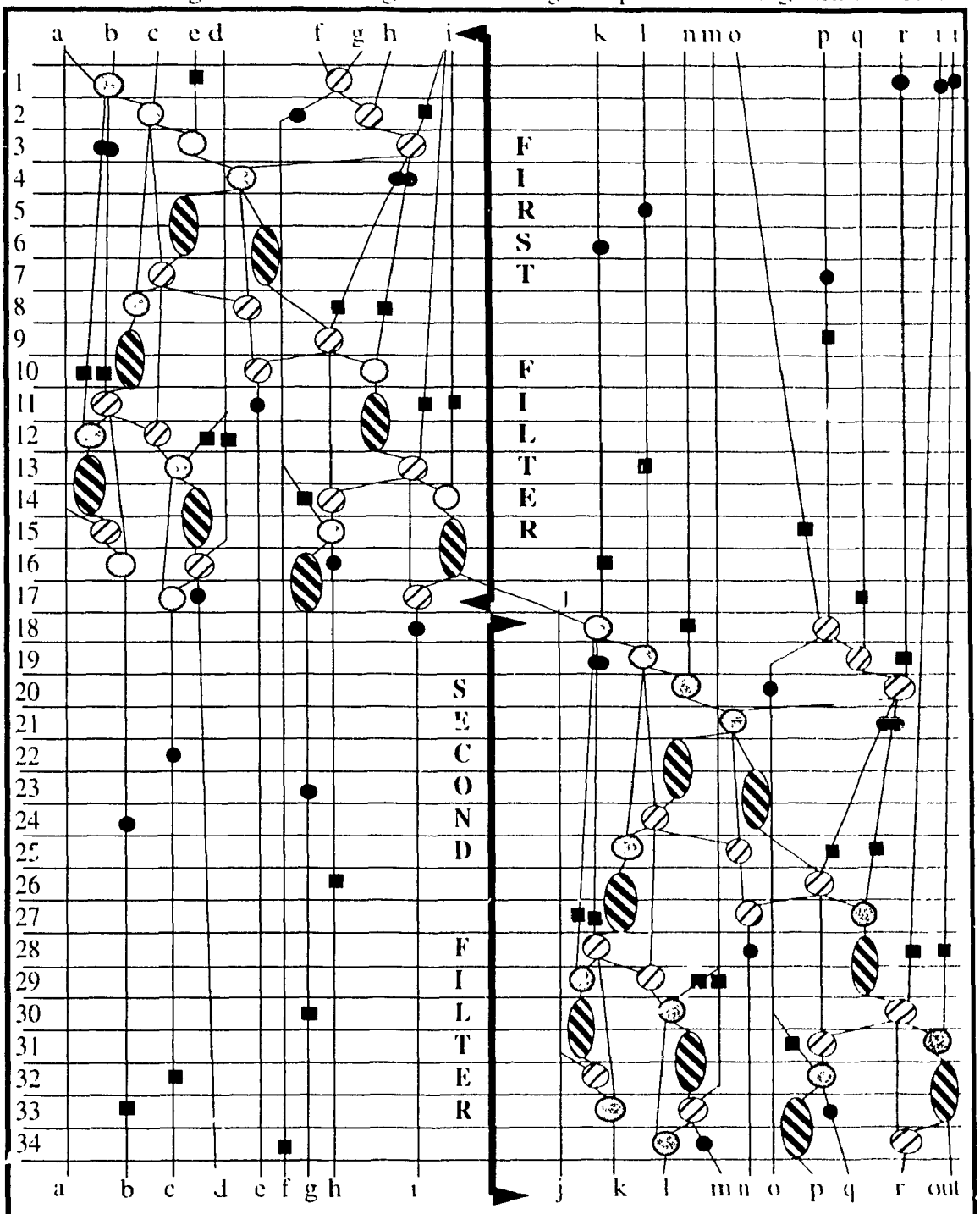


Figure 5.7: This is the scheduling and binding of two elliptic filters connected in series (output of the 1st is the input of the 2nd). The architecture uses 2 adders and 1 pipelined multiplier. The operation schedule and binding was done using our ILP formulation for step-1. The bus transfer scheduling was done using our step-2 ILP formulation for a 1 bus solution.

(1/2 CLB /bit/register) and 11 register file (RAM) locations at a cost of (1/2 CLB/bit for all 11 registers). In comparison, a multiplexer based solution would use 18 registers at a cost of (1/2 CLB /bit/register). Hence, the bus based solution saves an equivalent of 3 CLBs per bit of the word length.

For a 32 bit width data path our estimate for our data path is: (64 CLBs for the adders and their multiplexers, 206 CLB for a 12x16 booth re-coded multiplier (synthesized), and 192 CLBs for storage, total of 462 CLBs). An extra 3x32=96 CLBs are required for a multiplexer based solution with a total of 558 CLBs. A saving of 20% in terms of the total number of CLBs required for our data path. Since our architecture uses less interconnections and only one bus, the efficiency due to routing is higher.

For a two bus solution (not shown) the number of registers used in our solution was 8 and the number of total register files locations are 14. This results in a saving of 4 CLBs per bit of the word length. For a 32 bit width data path our estimate is 430 CLB and the saving is 30% in terms of the total number of CLBs required for this data path.

Note that if a more efficient multiplier is used, these percentage gains can be increased.

5.5.3 Fast Discrete Cosine Transform

This example is used to demonstrate that ILP can handle medium size graphs that have high parallelism (parallelism is limited in EWF example). Step-1 took 33 seconds. For a mux based solution the number of registers used is 11, while for the bus schedule shown in Figure 5.8 (which took 1.3 seconds), the architecture uses 8 registers and 4 register file locations. This results in a saving of 1 CLB per bit of the data path width.

It is to be noted that the running time of the ILP solution for the bus scheduling and assignment is very small (compared to operation scheduling and binding). This is due to three factors; 1) The range of each bus transfer is less than the original graph since all operations are already scheduled, 2) All edges with life time <2 are eliminated from the

bus search, and 3) The ILP formulation used is tight. This is evident from the number of branch and bound trials did not exceed few branches taken in all preceding examples. In some instances where the register costs are not added to the cost function, no branching was observed and the integer optimal solution is obtained directly from the linear program solution.

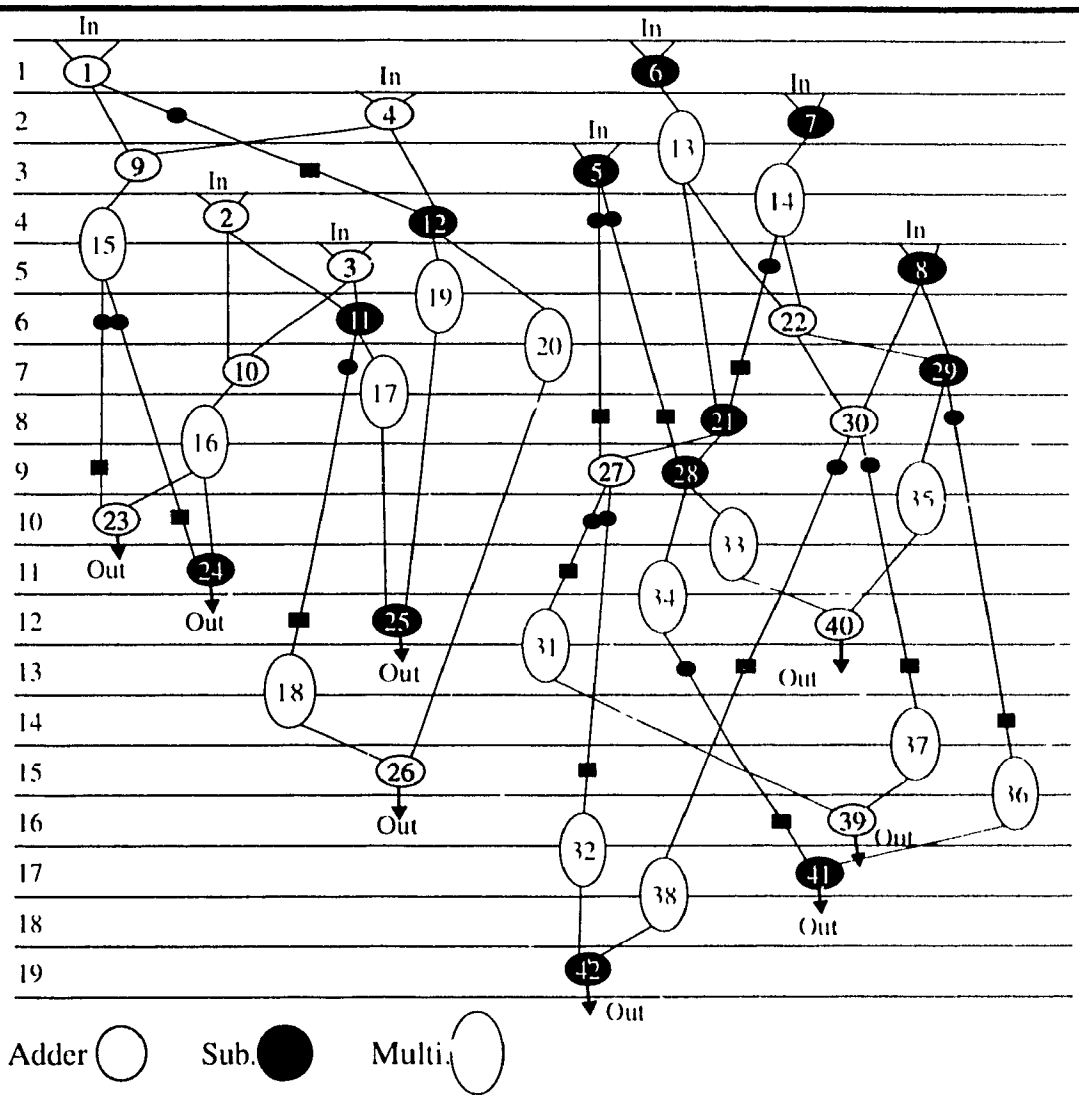


Figure 5.8: FDCT schedule and binding of operations to one adder, one subtracter and one multiplier, bus transfer scheduling to 1 bus. The maximum overlap of life-times of variables assigned to registers is 8. Total overlap of lifetimes assigned to the register file is 4. Without the bus assignment the number of registers used are 11. The three register in the non register file based architecture cost three times as much (in number of CLBs) as the cost of the register file in the bus based architecture

5.6 Chapter Summary

In this chapter, we have demonstrated that any schedule and binding for a multiplexor based data path can be transformed to support a bus/RAM based data path by properly scheduling the bus-transfers. Our novel ILP formulation optimally selects and assigns data transfers to buses while scheduling the bus transfers to minimize the use of buses, data storage for both registers and register files and finally the bus loading in terms of number of tri-states and mux-inputs that are connected to every bus. We have also shown by our low running times the tightness of the formulation we have presented. Moreover, our constraints and cost functions can be extended to other heuristic architecture search techniques that may have better running times for larger problems.

Chapter 6: Conclusion

Recently, many synthesis systems have been developed for design of high performance architectures for digital signal processing applications. Basically, these tools start from a high level language or a data flow graph and perform the scheduling of operations to control steps and allocating of hardware resources and finally the control unit in order to produce the complete architecture. Due to the complexity of these tasks the process has been divided into subtasks and each part is approached separately. This decomposing reduces the total design space and simplifies the complexity of the search. However, this decomposing does not necessarily produce a global optimal solution and it is only intended to reduce the iteration loops.

In this thesis, we have introduced a novel integer linear programming formulation for performing the scheduling, allocation and binding of operations simultaneously which produces a much closer solution to the global optimal design within a reasonable running time. Our formulation uses much tighter constraints than previous approaches and also an effective objective function which can generate good results faster.

This thesis makes three main contribution to the synthesis of (1) multiplexer based data paths, (2) multiple partitioned data paths, and (3) bus based data paths. The first main contribution is the integer linear programming formulation which is based on a mux based data path model and a structural complexity criteria. Our formulation is more general than previous ILP formulation and is suitable for multiplexer based FPGAs. We demonstrated how structured architectures can be derived from the regularity of the description of behavior by proper operation binding to functional units using a structural complexity criterion with an ILP formulation and a commercial solver within reasonable times. The reasonable running times show the tightness of the formulation presented. The resulting architectures proved that this measure when implemented within our objective function

can be used to optimize interconnections, multiplexers and storage while performing simultaneous scheduling and operation allocation and binding.

In order to generalize the above concept to a larger design scale implemented on multiple field programmable gate arrays, the second contribution of this thesis introduced a new ILP formulation for partitioned signal processing data paths. This new ILP formulation which is based on a similar architecture model as previous section, has the capability of accurately accounting for interconnections concurrently in their operation scheduling and binding inside each partition as well as accurately accounting for pin and interconnection limitations between partitions. By introducing the concept of inter/intra motifs, we were able to produce an objective function which can efficiently reduce both local interconnects within every FPGA along with the ports connecting different FPGAs. We showed how this ILP formulation can be easily adapted for both Bi and Uni-directional ports. The flexibility of the formulation constraints also allows to adjust the delay of data-transfers between different partitions to some technology dependent value. Results of using this formulation on number of examples proved the importance and efficiency of the adapted structural complexity measure used on partitioned signal processing data paths.

The third contribution of this thesis introduced a novel ILP formulation for optimally transferring a given scheduled and bound signal processing algorithms for a multiplexer based data path to a BUS/RAM based FPGA data path. This formulation optimally selects and assigns data transfers to buses while scheduling the bus transfers to minimize the use of the buses, data storage for both registers and register files and finally the bus loading in terms of number of tri-states and the mux-inputs that are connected to every bus. Our results using this approach have achieved the smallest published architecture for a given performance for the famous EWF benchmark.

We also implemented the ILP formulation for register binding presented in [22] to

obtain the final architecture for the number of benchmarks. This was done to demonstrate a number of points regarding the architecture features of our optimization tool (OSTA).

This thesis demonstrates that our synthesis approach using these ILP formulations when applied to a number of benchmark examples produced architectural solutions that outperformed all other previous tools regarding its performance as well as resources used. This result is due to the tight constraints and efficient objective functions used in our ILP formulations and also the flexible architecture model used in our synthesis methodology

Nevertheless, our synthesis tool has limitation in terms of running time. Our experience shows that the running time is dependent on the optimization criteria used in our ILP solver (OPTCR in CPLEX). This time increases significantly for highly parallel algorithms, especially when the total number of function units is large and also when the upper bound on total number of control steps is much larger than the critical path. This causes the number of 0/1 variables to increase significantly which in turn increases the size of design space. In these cases our ILP solver finds a feasible solution within minutes but the optimizer would run for hours without reaching the optimal solution. To prove optimality, we had to use lower bounds on cost and upper bounds on the total number of control steps in order to reduce the running time. These synthesis results indicate that generally ILP can handle small/medium size applications and is a viable approach to architecture synthesis. However, for large problems, heuristic approaches maybe suitable. In this case our structural complexity criterion with the cost measure can be effective and may as well be suitable for other search techniques such as simulated annealing.

There are other modifications that can be made to the synthesis tool in order to improve its performance. More research needs to be done on the coefficients of different terms in the objective functions. A primary study showed that varying these coefficients had a major effect on the running time. Also a better interface to our ILP formulations can facilitate the data input and output for the optimizer. Another extension which has already been implemented in the first ILP formulation is allowing the chaining of operations. Also,

minimization of clock cycle and therefore the total execution time for the whole DSP algorithm has been formulated and implemented in [37]. Future research can focus on addressing very large memory required for loop execution and multiple FPGA computation accelerator systems as well as methods of relegating some of the bus binding decision to register binding and floorplanning of the synthesis approach to achieve more efficient mapping and better performance.

References

[1] B. Haroun, B. Sajjadi, "ILP Synthesis of Signal Processing Architectures with minimum Structural Complexity", CICC, May 1994, pp. 11.2.1-11.2.4.

[2] B. Haroun, B. Sajjadi, "Optimal Data path Synthesis of Partitioned Signal Processing Algorithm for Multiple FPGAs", ICCD-94, pp. 237-240.

[3] B. Sajjadi, B. Haroun, "Synthesis of Signal Processing Structured Data paths for FPGAs Supporting RAMs and Busses", FPGA-Symposium-95, Motorey, CA, Feb. 1995.

[4] B. Haroun, B. Sajjadi, "Synthesis of Optimal Structured Data paths for Single and Multiple FPGA Signal Processing Architectures", Submitted to IEEE Tran. on CAD.

[5] B. Haroun, B. Sajjadi, "Transforming Multiplexor Based Structured Data paths to FPGAs Supporting RAMs and Busses", Submitted to IEEE Tran. on CAD.

[6] D. Gajski, N. Dutt, A. C-H Wu, S. Y-L Lin, "High-Level Synthesis Introduction to Chip and System Design", Kluwer Academic Publishers, pp. 2-24.

[7] P. Michel, U. Lauther, P. Duzy, "The Synthesis Approach To Digital System Design", Kluwer Academic Publishers, pp. 1-14.

[8] D. Gajski, R. Kuhn, "Guest Editors' Introduction: New VLSI Tools", IEEE Computer, Dec. 1983, pp. 6(12):11-14.

[9] D. Gajski, N. Dutt, A. C-H Wu, S. Y-L Lin, "High-Level Synthesis Introduction to Chip and System Design", Kluwer Academic Publishers, pp. 17-19.

[10] C. Gebotys, M. I. Elmasry, "Global Optimization Approach for Architectural Synthesis", IEEE Tr. on CAD, Vol. 12, No. 9, Sept. 1993, pp. 1266-1278.

[11] B. Haroun, M. I. Elmasry, "Synthesis of Multiple Bus Architectures For DSP Applications" in: "VLSI Design Methodologies for DSP Architectures" ed. M. Bayoumi, Kluwer Academic Publishers Boston, MA, 1994, pp. 93-100.

[12] C. Gebotys, M. I. Elmasry, "Optimal Synthesis of High Performance Architecture", JSSC, March 1992, pp. 389-397.

[13] A. Safir, B. Haroun, "Full Placement and Routing for fast Architectural Synthesis", submitted to ISCAS-95.

[14] J. Rose, A. ElGamal, A. Sangiovani-Vincentelli, "Architecture of Field Programmable Gate Arrays", Proc. IEEE, Vol. 81, No. 7, July 1993.

[15] D. Van Den Bont et al., "AnyBoard: An FPGA-Based, Reconfigurable Systems", IEEE Design and Test of Computers, Sept. 1992, pp. 21-30.

[16] D. Thomas et al., "A Model and Methodology for Hardware-Software Codesign", IEEE Design and Test of Computers, Sept. 1993, pp. 6-15.

[17] C.T. Hwang, J.H. Lee, Y.C. Hsu, "A Formal Approach to the Scheduling Problem in HLS", IEEE Tran. CAD, Vol. 10, No. 4, April 1991, pp. 464-475.

[18] B. Haroun, M. I. Elmasry, "VLSI Architecture Synthesis and Implementation of HiFI Digital Filters", Proc. Canadian Conference on VLSI, Oct. 1989, pp. 107-114.

[19] B. Haroun, M. I. Elmasry, "Synthesis of Multiple Bus Architectures For DSP Applications" in: "VLSI Design Methodologies for DSP Architectures", ed. M. Bayoumi, Kluwer Academic Publishers, Boston, MA, 1994, pp. 101-130.

[20] C. Gebotys, "Synthesizing Optimal Register file Architectures for FPGA Technology", CICC, May 1994, pp. 233-236.

[21] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III: Architecture-Driven High-Level Synthesis for High Throughput DSP Applications", 28th Design Automation Conference, 1991, pp. 597-602.

[22] M. Rim, R. Jain, R. Deleone, "Optimal Allocation & Binding in HLS", DAC-92, pp. 120-123.

[23] G. Goosens, J. Rabaey, J. Vandewalle, H. De Man, "An Efficient Microcode Compiler for Application Specific DSP Processors", IEEE Transaction on CAD, Vol. 9, No. 9, 1990, pp. 925-937.

[24] C. Gebotys, "Synthesizing Optimal Application Specific DSP Architectures" in: "VLSI Design Methodologies for DSP Architectures" ed. M. Bayoumi, Kluwer Academic Publishers, Boston, MA, 1994, pp. 43-92.

[25] J. J. Rabaey et.al., "Fast Prototyping of data path Intensive Architecture.", IEEE Design & Test, Vol. 8, No. 2, 1991, pp. 4051.

[26] F.S. Tsai, Y.C. Hsu, "STAR: An Automatic Data Path Allocator", IEEE Trans. on CAD, Vol. 11, No. 9, September 1992.

[27] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's", IEEE Trans. on CAD, Vol. 8, No. 6, June 1989.

[28] N. Park, A. C. Parker, "SEHWA: A software package for synthesis of pipelines", IEEE Trans. on CAD, Vol. 7, Mar. 1988, pp. 356-370.

[29] T. Ly, J. Mowchenko," Applying Simulated Evolution to HLS", IEEE Trans. on CAD, March 1993, pp. 389-409.

[30] D. S. Rao, F. Kurdahi, "Partitioning by Regularity Extraction", DAC' 92, pp. 235-238.

[31] D. Mallan, P. Denyer, "A New Approach to Pipeline Optimization", IEEE EDAC, March 1990, pp. 83-88.

[32] A. Safir, B. Haroun, K. Thulasiraman, "A Floorplanner for Data path Optimization", IEEE Internatinal Symposium on Circuits and Systems, 1995, pp. 41-44.

[33] C. H. Gebotys, "Optimal Synthesis of Multi-chip Architectures", IEEE Trans. on CAD, 1992, pp. 238-241.

[34] K. S. Hwang, A. E. Casavant, C. T. Chang, M. A. d'Abreu, "Scheduling and hardware sharing in pipelined data paths", Proc. ICCAD-89, Nov 1989, pp. 24-27.

[35] H. De Man, J. Rabeay, P. Six, L. Claesen, "Cathedral-II: A silicon compiler for digital signal processing", IEEE Design Test, Dec. 1986, pp. 13-25.

[36] S. Davidson et al., "Some experiments in local microcode compaction for horizontal machines", IEEE Trans. Comput., July 1981, pp. 460-477.

[37] S. Shehata, A. Khailtash, B. Haroun, "A Comparative Study of Synthesized FPGA implementation of the Elliptic Wave Filter Benchmark", Submitted to VLSI Conf.