Interconnect-Aware Scheduling and Resource Allocation for High-Level Synthesis

Awni Itradat

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

January 2009

Your file Votre référence
ISBN: 978-0-494-63451-6
Our file Notre référence
ISBN: 978-0-494-63451-6

ABSTRACT

Interconnect-Aware Scheduling and Resource Allocation for High-Level Synthesis

Awni Itradat, Ph.D.
Concordia University, 2009

A high-level architectural synthesis can be described as the process of transforming a behavioral description into a structural description. The scheduling, processor allocation, and register binding are the most important tasks in the high-level synthesis. In the past, it has been possible to focus simply on the delays of the processing units in a high-level synthesis and neglect the wire delays, since the overall delay of a digital system was dominated by the delay of the logic gates. However, with the process technology being scaled down to deep-submicron region, the global interconnect delays can no longer be neglected in VLSI designs. It is, therefore, imperative to include in high-level synthesis the delays on wires and buses used to communicate data between the processing units i.e., inter-processor communication delays. Furthermore, the way the process of register binding is performed also has an impact on the complexity of the interconnect paths required to transfer data between the processing units. Hence, the register binding can no longer ignore its effect on the wiring complexity of resulting designs. The objective of this thesis is to develop techniques for an interconnect-aware high-level synthesis. Under this common theme, this thesis has two distinct focuses. The first focus of this thesis is on developing a new high-level synthesis framework while taking the inter-processor communication delay into consideration. The second focus of this thesis is on the developing of a technique to carry out the register binding and a scheme to reduce the number of registers while taking the complexity of the interconnects into consideration.

A novel scheduling and processor allocation technique taking into consideration the inter-processor communication delay is presented. In the proposed technique, the communication delay between a pair of nodes of different types is treated as a non-computing node, whereas that between a pair of nodes of the same type is taken into account by re-adjusting the firing times of the appropriate nodes of the data flow graph (DFG). Another technique for the integration of the placement process into the scheduling and processor allocation in order to determine the actual positions of the processing units in the placement space is developed. The proposed technique makes use of a hybrid library of functional units, which includes both operation-specific and reconfigurable multiple-operation functional units, to maximize the local data transfer.

A technique for register binding that results in a reduced number of registers and interconnects is developed by appropriately dividing the lifetime of a token into multiple segments and then binding those having the same source and/or destination into a single register. A node regeneration scheme, in which the idle processing units are utilized to generate multiple copies of the nodes in a given DFG, is devised to reduce the number of registers and interconnects even further.

The techniques and schemes developed in this thesis are applied to the synthesis of architectures for a number of benchmark DSP problems and compared with various other commonly used synthesis methods in order to assess their effectiveness. It is shown that the proposed techniques provide superior performance in terms of the iteration period, placement area, and the numbers of the processing units, registers and interconnects in the synthesized architecture.

To My Loving Family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and heartfelt thanks to my thesis supervisor Dr. M.O. Ahmad for his interest, guidance and constructive criticism throughout this work. I am grateful for his extremely careful and thorough review of my thesis. I feel privileged for having the opportunity to work with him. His advice and close support have been invaluable. He guided my research with his profound knowledge and deep insight. In fact, without his support and encouragement, I could not have reached the stage where I am today.

I would like to thank all of my colleagues in the Center for Signal Processing, Department of Electrical and Computer Engineering, Concordia University. I am also grateful to all my family members in Jordan for their constant prayers, love and support, and for the sacrifices they made in their lives in bringing me up.

Special thanks are due to my wife, Dalal, and my children, Mohammad, Rama, Zain, and Meeral for their patience, encouragement and love. Dalal, really, provides me with the peace of mind and the determination to complete this work.

Finally, I would like to dedicate this thesis to the souls of my father who passed away during the first year of my PhD program and to my mother who left this world a few months prior to my thesis defense. Abi and Ummi, I promise you that I will do my best to live to your expectations.

Table of Contents

List of Figures

List of Tables

List of Symbols

V	: A set of vertices or nodes
E	: A set of elements called edges
v_i	: Node i
e_{ij}	: A directed edge incident out of the node v_i and incident into the node v_j
N_e	: A nonnegative integer represents the ideal delay in cyclic data flow graph
$p_{v_0 v_k}$	: A direct path of a finite sequence of distinct nodes $v_0 \ldots v_k$
l	: A loop
T	: The iteration period
T_0	: The Iteration period bound
$EFT(w/v)$	: Earliest firing time of node w relative to the firing time of the node v
$LFT(w/v)$	: Latest firing time of node w relative to the firing time of the node v
$EFT(w)$	: Earliest firing times of node w relative to the previously scheduled nodes
$LFT(w)$	: Latest firing times of node w relative to the previously scheduled nodes
$FT(v)$	: Firing time of a node v
d_{v_i}	: The computational delay of node v_i
N_C	: The total number of ideal delays in circuit C
D_C	: The computational delay of the nodes in circuit C
$len[P]$	: The length of a path P
d_{c_i}	: The delay of communication node c_i
Q^0	: An $N \times N$ matrix
Q^f	: The longest path matrix
Q^f_{ij}	: The longest distance from node v_i to node v_j in matrix Q^f
P_{type}	: The number of processors of a certain type
$Slk(v_i)$	: The slack time of a node v_i
ICD_C	: The total delay of the communication nodes in a circuit C
ICD_P	: The total delay of the communication nodes in a path P

$ST(C)$: The slack time of a circuit C

NP_{type} : The number of processing units of a certain type of nodes

$\mu_{i,j} \geq 0$: Mobility of node v_i at stage j

$LP_{ij}(t)$: A set of nodes of type t with a criticality level i *and index* j

$A(t)$: Allocation matrices

$c_{e_{ij}}$: A communication node

$EST(c_{e_{ij}})$: The earliest scheduling time of the communication node $c_{e_{ij}}$

$LST(c_{e_{ij}})$: The latest scheduling time of the communication node $c_{e_{ij}}$

$M(c_{e_{ij}})$: The mobility of inserted node $c_{e_{ij}}$

N_{ins} : Number of inserted cycles

C_{ins} : The control step at which extra cycles are inserted

t_{setup} : Setup time

t_{clk2Q} : Clock to Q time of a register

$M(U,E,T)$: Delaunay triangular mesh

tr : A triangular

r_i : Rectangle i

El : The length of a mesh edge connecting the centers of two rectangles

El_r : The length of the part of the mesh edge lying outside the two rectangles

ce_j : The number of corner vertices u_1 to u_4 touching a mesh edge e_j

$cnps$: The candidate position in placement space

$crps$: The current position in placement space

$comm(a,b)$: The interconnect delay between positions a and b

$level_{type}$: A level determine the number of functional units

$RT_{a \to m}$: The reconfiguration time of the morphable multiplier

α : A user-defined positive values used for the evaluation function

τ : A user-defined positive values used for the evaluation function

S : A placement space

A_{after} : Area of S after the placement

A_{before} : Area of S before the placement

T_{after} : Iteration period of S after the placement

T_{before} : Iteration period of S before the placement

$ENGL$: A constant positive value used to enlarge the placement space

S_w : The width of S

S_h : The height of S

Y_v : A token produced by a node v

t_{Y_v} : The at which the token Y_v is produced

L_v : A lifetime of a token produced by a node v

C_{Y_v} : The control step at which the token Y_v is produced

S_i : A sub-period of a token

X_i : The control step at which the sub-period S_i is produced

β_{Y_v} : A set of sub-periods

NR_{Y_v} : The number of registers required for allocating a token Y_v

Wid_{c_i} : The width of lifetime chart at control step c_i

$Wid_{\max}$: The maximum width of the lifetime chart

dtime : The death time

btime : The birth time

$w_{(n_{Y_v}, n_{Y_u})}$: A weight of an edge connecting two sub-periods in a flow network

$CC_{(n_{Y_v}, n_{Y_u})}$: The number of consumers common to a pair of tokens

k : A flow from the source node to the sink in a flow network

NR_{p_1} : A normalized number of registers

v_{new} : A regenerated node

$t_{v_{new}}$: firing time of a regenerated node v_{new}

$R(v_{new})$: The range of possible control steps for v_{new}

BG_t : A bipartite graph of type t

CI : The numbers of common inputs

CO : The numbers of common outputs

IDLE : A set of idle processing units

$PU(v)$: A processing unit to which the node v is assigned

α : A constant used to increase weight of a particular edge

β : A constant used to increase weight of a particular edge

$p_j(c(v_1), \ldots)$: An assignment compatible path

$w_{(v_i, v_{i+1})}$: The weight assigned to an edge $v_i \rightarrow v_{i+1}$ in the modified bipartite graph

$W_{(v_n, p_j)}$: The weight assigned to an edge $v_n \rightarrow p_j(\cdot)$

List of Abbreviations

DSP	:	Digital Signal Processing
DFG	:	Data Flow Graph
VLSI	:	Very Large Scale Integrated Circuits
RTL	:	Register Transfer Level
ASAP	:	As Soon As Possible
ALAP	:	As Late As Possible
FDS	:	Force-Directed Scheduling
ASIC	:	Application Specific Integrated Circuit
FPGA	:	Field Programmable Gate Array
ILP	:	Integer Linear Programming
ALU	:	Arithmetic Logic Unit
ICD	:	Inter-Processor Communication Delay
EFT	:	Earliest Firing Time
LFT	:	Latest Firing Time
MG	:	Modified graph
DCT	:	Discrete Cosine Transform
MILP	:	Mixed Integer Linear Programming
MAC	:	Multiplier Accumulator
SoC	:	System on Chip
PPRT	:	Partial Product Reduction Tree
FU	:	Functional Unit
RCPM	:	Reconfigurable Pipelined Multiplier
CLC	:	Circular Lifetime Chart
CG	:	Compatibility Graph
3D	:	3 Dimensions

Chapter 1

Introduction

1.1 General

VLSI technology has now advanced to a stage where it would be rather complex to design a digital system starting at the transistor level or logic level. High-level synthesis of digital systems consists of transforming a behavioral (algorithmic) description of a design into a register transfer level (RTL) description of the design, rather than dealing with the components of the design at the logic gate level. At the RTL level, an adder, for instance, is viewed as a functional unit instead of as a series of NAND gates. The factors such as the latency, area and the type of functional units associated with a design, however, can be taken into consideration at a high-level synthesis without resorting to a low-level implementation. Scheduling, processor allocation, and register binding are the key tasks that influence these factors in the high-level synthesis. A general goal of a high-

level synthesis is to find hardware structures that minimize these design metrics subject to certain constraints.

Some of the applications that need a high-level synthesis are digital signal processing (DSP), communications, and image processing. These applications are among the most important applications that demand high computational power, and must be executed at a very high speed to enable real-time processing. Due to the parallelism within the DSP applications, parallel processing architectures are a natural choice for the synthesis of these applications [1- 3].

In the past, it has been possible to focus simply on the delays of the processing units in a high-level synthesis and neglect the wire performance, since the overall delay of a digital system was dominated by the delay of the logic gates. However, with the process technology being scaled down to deep-submicron region, the global interconnect delays can no longer be neglected in VLSI designs. It is, therefore, imperative to include in high-level synthesis the delays and complexity of wires and buses used to communicate data between the processing units, even though, in recent years, there have been considerable developments in the interconnect technology.

Since scheduling and resource allocation are the most critical tasks in the high-level synthesis, the following sections provide a brief review of the necessary background material for the high-level synthesis. In order to provide the motivation for the research work contained in this thesis, the weaknesses and shortcomings of the research schemes for the scheduling and resource allocation of DSP application in the high-level synthesis are also discussed. It is to be noted that the related research corresponding to each

particular problem dealt with in the thesis will be provided in the other chapters. This chapter is concluded by providing the scope and the organization of this Thesis.

1.2 High-Level Synthesis

The hardware design starts from the behavioral description of algorithm as input to the high level synthesis and proceeds downwards to the logic level and finally the physical level which produces the circuit layout for the implementation, each time adding some additional information needed for the next level of synthesis. A high-level synthesis can be described as the process of transformation of a behavioral description into a structural one that consist of a set of connected components collectively called the data path and a controller that sequences and controls the functionality of these components.

The high level synthesis takes the specification of the behavioral requirement of a system and a set of constraints and goals to be satisfied, and then to find a structure that implements the behavioral requirement while satisfying these goals and constraints. The behavior means the way the system and its components interact with their environment, i.e., the mapping from the inputs to the outputs. The structure refers to the set of components and their interconnection that is used to implement the system. Usually there are many different structures that can be used to realize a given behavior. One of the tasks of the synthesis is to find the structure that best meets the constraints, such as the limitations on the cycle time, area or power, while minimizing other costs. For example, the goal might be to minimize the area while achieving a certain minimum-processing rate.

In recent years there has been a trend toward carrying out the synthesis at increasingly higher levels of the design hierarchy. There has been considerable interest shown in the High-level synthesis in the industry. There are a number of reasons for this:

a. **Shorter design cycle.** Since much of the cost of the chip is in the design development, by carrying out the synthesis at higher levels and automating the part of the design at lower levels can lower the design cost as well as make it possible to hit the market window.
b. **Fewer errors.** Having more automation in the design process eliminates the errors due to human factors and reduces the verification time of the design.
c. **The ability to search the design space.** A good synthesis system can produce several designs for the same specification in a reasonable amount of time. This allows the developer to explore the trade-offs between the cost, speed, power, and so on of the different designs including that of an existing one.

It is expected that the trend toward a high level synthesis will continue. Already there are a number of research groups working on high-level synthesis, and great deal of progress has been made in finding good techniques for optimizing and for exploring design trade-off. These techniques are very important because decisions made at the high level tend to have a much greater impact on the design than those at lower levels.

In general there are many advantages of investigating a design at a high-level. First, the designer can concentrate on studying the design behavior rather than the detailed implementation. Second, the RTL design of a digital system is usually less complex than the design details at the logic level. Thus, its simulation can be done faster.

Further, studying an RTL design also allows the designers to quickly explore the design space and decide as to which architecture fits their needs best.

1.3 The Synthesis Tasks

Scheduling, resource allocation, placement represent the core of transforming the behavior of an application into a structure. They are closely interrelated and depend on each other. Scheduling consist in assigning the operations to the control steps to be executed. The control steps are fundamental sequences in a synchronous system; they correspond to clock cycles. Allocation consists in assigning the operations and variable to generic hardware resources.

1.3.1 Scheduling

Scheduling is one of the basic tasks in a high-level synthesis to produce an execution order of each operational node. The aim of the scheduling is to minimize the amount of time or the number of the control steps needed to complete the application, given certain constraints on the available hardware resources [4-5].

Scheduling is significant in view of the fact that the relative execution order of the operations has an effect on the speed, throughput, or any other performance measure of the system design. Thus, an important purpose of the scheduling process is to achieve some objective functions, while satisfying some design constraints, e.g., iteration period, throughput, hardware resources, input-output delay, area cost, and power [6].

Some basic concepts in scheduling

When the operations that have to be scheduled and the precedence relations are known beforehand, the scheduling can take place at the compile time. This is known as *static scheduling*. Static scheduling differs from *dynamic scheduling,* which schedules the operations at the run time.

Another characteristic of a scheduling method is whether or not it allows operations to be interrupted once their execution has begun. If it is so possible and the interrupted operations can be resumed at a later time, the scheduling is called *pre-emptive scheduling*. In contrast, *non pre-emptive scheduling* requires that the operations are executed without interruptions.

When an algorithm is scheduled for execution on architecture, several optimization goals can be set. It is possible to minimize the *throughput delay (*or *latency)* which is the time between the consumption of an input sample and the production of the corresponding output sample. This optimization goal is typical for *resource-constrained scheduling* in which the hardware is specified and it is independent of the type of the scheduling used. In contrast, in a *time-constrained scheduling,* given the execution speed, the hardware is minimized.

Scheduling methods exploit the parallelism that exists between operations of the same iteration of a cyclic data flow graph (*intra-iteration parallelism*). However, the cyclic data flow graphs often contain parallelism between the operations from different iterations (*inter-iteration parallelism*). Scheduling algorithms can also exploit this parallelism by allowing operations from different iterations to be executed in parallel. The schedules that are then produced are called *overlapped schedules*. These schedules

contrast the *non-overlapped schedules*, where for every iteration period only operations belonging to that iteration are executed.

Cyclo-static schedules form a special class of overlapped schedules. In a cyclo-static schedule an operation does not have to be executed on the same processing element for every iteration period. Cyclo-static schedules differ from *fully-static schedules* in which each operation is assigned to the same processing element for all the iterations.

Determining the time frames of the schedule

In general, DFGs expose parallelism in DSP applications. Each node has a range of possible control steps that can be assigned to it. Most of scheduling algorithms require the earliest and the latest bounds within which an operation in the DFG can be scheduled (time frames). The first and simplest schemes that are used to determine these bounds are called as *soon as possible* (ASAP) [7][8] and as *late as possible* (ALAP) algorithms [9], respectively.

1.3.2 Resource allocation

Allocation is a task of determining generic resources (functional units and registers) on which operational nodes are executed. It involves assigning operations and variables to hardware resources and registers and specifying their usage while trying to minimize the amount of hardware resources needed. It is assumed that a unit of generic resource could only start the execution of one operation at a time. In particular, this includes two subtasks, which are to determine the number of generic resources used and to bind nodes and variable to resources. A generic resource type may be, for instance, an adder unit, a

multiplier unit, or an ALU which is capable of performing multiple operations such additions, multiplications, etc.

In order to minimize the number of hardware resources required for the implementation of a digital system, the operations (nodes) of a DFG representing the DSP algorithm can be grouped to share a single hardware unit if they have mutually exclusive schedule or life time. Sets of these mutually exclusive nodes are formed. A single hardware resource is then allocated to each distinct set. Thus, the minimization problem is the problem of decreasing the number of sets. This type of allocation is known as folding. Folding is usually affected by the types of hardware resources.

Data path allocation involves the mapping of the operations onto the functional units and also assigning values to registers, and providing interconnections between the functional units and registers using buses and multiplexers. The optimization goal is usually to minimize some objective function, such as the total interconnects length, the total number of registers, and multiplexer cost, or the critical path delays. There may also be one or more constraints on the design which limit the total area of the design, the total throughput, or the delay from input to output.

1.4 Mapping of DSP Applications onto Hardware Models

The scheduling methods should use a detailed realistic model of the targeted architecture. When a scheduling method does not consider a complete and realistic model of the architecture, the resulting schedule will result in an inefficient implementation. It is worthwhile to explicitly consider a realistic multiprocessor architecture. For then the schedules produced will result in a latency that is on the average more twice than those

obtained from the schedules that are based on a less realistic model and therefore the latter had to be modified before they could be executed on the actual hardware.

Thus, it is desirable that a scheduling method is based on a realistic model of a targeted architecture. A realistic model should for instance support: communication delays, contention of communication links, the structure of the data path, allocation of registers (register binding), pipelining, etc. It is unlikely that a scheduling method based on a complex hardware model will always produce schedule that is close to optimal.

The most popular hardware implementations of DSP applications are: (1) application specific integrated circuit (ASIC), (2) field programmable gate arrays (FPGAs), or (3) a set of instructions running on an application-specific processor. Various implementations of the same application allow trade-offs for optimizing the hardware in terms of multiple design parameters such as power consumption, area, processing speed, and re-configurability of the system.

There are a number of advantages and disadvantages for these three implementations. An ASIC implementation has fully customized data paths and logic. It allows designers to optimize the hardware resources for one or more of the design parameters. However, an ASIC implementation is not flexible, since it does not allow reconfiguring itself and cannot be used in a wide range of applications. FPGAs, on the other hand, consist of arrays of prefabricated logic blocks. FPGAs can provide a reconfigured implementation of a certain design. The property of reconfigurability allows the designers to reuse the resources in variety of applications. Although FPGAs have the capability of programming functional units and wires, it has several inherent limitations. FPGAs usually consume much higher power than an ASIC implementation. They also

have higher performance penalty and require larger silicon area because of their generic reconfigurable platform. Another common method to implement a complex application is to use application-specific processors such as a DSP processor. DSP processors are designed for general-purpose DSP applications; and hence, they are not area, performance, or power efficient.

1.5 Related Research in the High Level Synthesis for Iterative DSP Applications

When a high-level synthesis is not based on a realistic model of targeted architectures, the resulting schedule may lead to an inefficient implementation. A realistic model should, for instance, support the inter-processor communication delays (ICDs), allocation of registers (register binding), type of functional units, interconnects networks, and structure and organization of the architecture. Moreover, that model must consider the interaction between the decision taken at high level of synthesis and those taken at a lower level of synthesis such as the placement which involves deciding where and how to place functional components, circuitry, and logic elements in a generally limited amount of space. Most of the techniques for scheduling of real-time DSP applications consider simplified architectural models in which the inter-processor communication delay, interconnect complexity and other structural requirements are not taken into consideration; thus, they eventually produce schedules that lead to unrealistic implementations.

In high-level synthesis, the model of the targeted architecture can consist of homogeneous or heterogeneous processing units. When processing units are

homogeneous, they all have identical behavior. Heterogeneous processing units do not have identical behavior and they can, for example, differ in execution speed or they are not all able to handle the same set of operations. Arithmetic logic units (ALUs) are generally used for the design of homogenous multiprocessors. On the other hand, processing units that can support a single type of operations are commonly employed for heterogeneous designs.

In [10], the problem of assignment and scheduling for heterogeneous multiprocessor systems has been addressed by proposing a two-phase approach to solve it. In the first phase, the heterogeneous assignment problem is solved by assigning the best functional unit to a node so that the total cost can be minimized while satisfying the timing constraint. In the second phase, based on the node assignments thus obtained, a minimum resource scheduling algorithm that has been developed in this paper is used to generate a schedule and produce a feasible configuration with as little resources as possible. Functional units with different execution times and costs are employed in this technique. However, they cannot be considered to be fully heterogeneous, since they all handle the same set of operations. In [11], a list-based scheduling algorithm has been proposed. Since in this technique an acyclic data flow graph (DFG) is used to represent a DSP application, the inter-iteration precedence cannot be exploited, and hence, a rate-optimal schedule cannot be produced. Both [10] and [11] do consider the interprocessor communication delay and interaction with the lower level of synthesis

In the techniques presented in [12]-[17], different solutions to deal with the problem of scheduling of DSP applications mapped onto multiprocessor systems have been proposed by taking into account the inter-processor communication delay. In the

range-chart technique proposed in [12], the flexibility in the scheduling of the nodes in a DFG has been represented in the form of a chart that specifies the possible range within which a node can be executed. This technique is very successful in scheduling and allocation of a DSP application targeting homogeneous multiprocessor systems and can produce an optimal solution with a polynomial time complexity. The inter-processor communication delay is dealt with by converting the cyclic DFG representing a DSP application into an acyclic one. However, this technique ignores the optimization of the memory during the scheduling. The technique proposed in [13] has considered the inter-processor communication delay and shown that it is essential for a realistic development of multiprocessor schedules. The objective of this technique is to use the structure of a multiprocessor system that takes into consideration the location of the operands, the number of registers and the inter-processor communication delays to find rate optimal schedules. However, the method is computationally expensive and suitable for small size problems. In [14], a scheduling technique called cyclo-compaction scheduling has been proposed to deal with the problem of inter-processor communication delay. First, a non-overlapped schedule is constructed and then transformed into an overlapped one, while at the same time, control steps are inserted into the time schedule to deal with the inter-processor communication delay. In this technique, the optimization of the number of processors or the memory size cannot be achieved, since the DSP application is mapped onto a pre-specified multiprocessor topology. The techniques proposed in [15] and [16] use integer linear programming (ILP) to take into account the inter-processor communication delay during the scheduling of a DSP application mapped onto a homogeneous multiprocessor system with a pre-specified topology ranging from a

weakly connected configuration to a strongly connected one. It is well-known that ILP is a very time-consuming technique for large size problems. Despite the fact that the techniques in [15] and [16] can produce good results in terms of the throughput, they cannot optimize the number of processors. Furthermore, the problem of optimizing the memory size of the implementation has not been considered. In [17], an iterative algorithm to compute the theoretical minimum initiation interval for the time schedule of a given DSP application has been proposed with a fixed inter-module communication delay for a two-module implementation. Unfortunately, in this technique the inter-processor communication delay between the sub-modules within each module has been neglected. In fact, none of the above techniques consider a close interaction between the placement and the high level synthesis.

With the advances in ASIC synthesis technique, the idea of heterogeneous multiprocessor architectures for the implementation of DSP applications is gaining widespread usage because of the area efficiency of such systems. Scheduling an iterative signal processing algorithm onto a heterogeneous multiprocessor system imposes a greater need to consider the inter-processor communication delays of the target architecture, since, in heterogeneous system nodes of different types and having data precedence cannot be assigned to the same processor. To the best of the authors' knowledge, no algorithm for scheduling of cyclic data flow graphs mapped onto a heterogeneous multiprocessor system with inter-processor communication delay exists in the literature.

The way the process of register binding is performed also has an impact on the complexity and performance of the interconnect paths required to transfer data between

the processing units. Hence, the register binding can no longer ignore its effect on the wiring complexity of resulting designs in a high level synthesis. In FDLS [17], ALPS [19], OSAIC [20], InSyn [21], and MARS [22] different solutions and heuristics to the scheduling problem have been provided in which the memory optimization, commonly referred to as register binding, has been also considered, but the inter-processor communication delay and interconnect are ignored. In [18]-[21], an acyclic DFG model has been used to represent a DSP application and thus the inter-iteration precedence cannot be exploited. In [22], the DFG is mapped onto a heterogeneous multiprocessor with structural pipelining to produce a rate-optimal schedule. This novel technique implicitly performs algorithmic transformations, such as pipelining and retiming, on the DFG's during the scheduling process to produce optimal or near-optimal solutions. In this method, life-time analysis [23] is incorporated during the data path synthesis to generate structures using the minimum number of registers for a given time schedule. In this technique, the loops of the DFG need to be enumerated. Since, the number of loops can be of an exponential order, algorithms to find all the loops could have an exponential time complexity. To summarize, most of the techniques for scheduling a DSP recursive application, in which the iteration period and throughput, is crucial have simplified the model used for the high level synthesis. One possible reason is that it is really more complicated to satisfy inter-iteration dependency [24] in recursive DSP data flow graphs during a high level synthesis in presence of the interprocessor communication delay and other interconnect and structural constraints.

1.5 Scope and Organization of the Thesis

The objective of this doctoral thesis is to devise efficient interconnect aware techniques for the high-level synthesis of DSP applications leading to the implementations with ASIC technology. Under this common theme, the thesis has two distinct focuses. The first focus is on developing new techniques for scheduling and processor allocation while taking into consideration the interprocessor communication delays. To this end, two techniques are proposed. In the first technique, the interprocessor communication delay in the tasks of scheduling and processor allocation is estimated or taken from an already placed architecture. While in the second technique, a placement process is integrated into the high level synthesis in order to, more accurately, consider the impact of the positions of the processing units in the placement space and the corresponding interconnects delays on the building of the time and processor schedules. The second focus of this thesis is on developing a technique to carry out the register binding while taking into consideration the complexity of the interconnects. Since the lower bound on the number of registers resulting from any register binding technique gets fixed once the DFG is scheduled, a node regeneration scheme is proposed to reduce the number of registers to a value that is even lower than this bound and at the same time to lower the interconnect complexity.

This thesis is organized as follows. In Chapter 2, a new static scheduling technique for DSP algorithms mapped onto fully connected heterogeneous register based architectures with non-negligible inter-processor communication delays is proposed. The proposed technique operating on the cyclic DFG of a DSP algorithm is designed to determine the relative firing times of the nodes by using a longest path algorithm so that

not only the inter-processor communication delays are taken into consideration, but also the throughput is aimed to be maximized and the number of hardware resources in terms of processors to be minimized.

In the above technique for scheduling and processor allocation, the interprocessor communication delay is assumed to be taken from feedback placement information or from an estimated value of the interprocessor communication delay. In Chapter 3, a technique in which a placement process is integrated into the high level synthesis is developed. The information about the physical positions of the processing units in the placement space, the interconnect delays between the processing units, and the candidate positions for placing new processing units, are utilized during the building of the time schedule and processor allocation. Furthermore, most of the other techniques for high level synthesis use only operation-specific functional units, i.e., adders or multipliers, in the allocation process. In order to maximize the local data transfers and reduce the interprocessor communication delays, the proposed technique makes use of a hybrid library of arithmetic functional unit composed of both fixed operation-specific units and reconfigurable functional units capable of executing multiple operations.

The way in which the register binding is carried out affects the data transfers between the processing units. In Chapter 4, the problem of register binding in a high-level architectural synthesis is studied. A technique for binding the tokens produced by the nodes of a scheduled DFG is proposed while aiming at minimizing the number of interconnects. First, a segmentation scheme in which the lifetime of a token is appropriately divided into multiple segments is developed. Then, the register binding problem is formulated as a min-cost flow problem so that the tokens having the same

source and/or destination are bound into the same register and results in a reduced numbers of registers and interconnects.

Chapter 5 proposes a node regeneration scheme that generates multiple copies of the original nodes in a given scheduled DFG with the resulting variables having lifetimes shorter than those of the variables produced by the corresponding original nodes. The freedom provided by having multiple copies of nodes is then further exploited to assign each copy to a processing unit that results in minimizing the complexity of the interconnect network thus obtained.

Finally, Chapter 6 concludes the thesis by highlighting the contribution made in this investigation and suggesting some possible future research work.

Chapter 2

Scheduling and Allocation of DSP Data Flow Graphs with Inter-Processor Communication Delays

2.1 Introduction

In an architectural synthesis problem, an efficient schedule is the one that respects the time requirements of a given DSP application and at same time, produces RTL architecture with an optimal or near-optimal number of functional units (resources). The problem of optimally scheduling and resource allocation of the DSP application mapped onto a multiprocessor architecture at compile time has been proven to be NP-hard, that is, a problem that is not solvable by deterministic algorithms in a polynomial time [24]. The situation becomes more complicated when the synthesis process targets a heterogeneous architecture in which the inter-processor communication delay are taken into

consideration. For such a problem, a heuristic solution is necessary. In this chapter [25-29], a new static scheduling technique for DSP algorithms mapped onto fully connected heterogeneous architectures with non-negligible inter-processor communication delays is proposed. The proposed technique operating on the cyclic DFG of a DSP algorithm is designed to determine the relative firing times of the nodes by using Floyd-Warshall's longest path algorithm [30] so that not only the inter-processor communication delays are taken into consideration, but also the throughput is aimed to be maximized and the number of hardware RTL resources in terms of processing units is aimed to be minimized.

The chapter is organized as follows. The data flow graph model used to represent DSP applications and the underlying terminologies and definitions are briefly introduced in Section 2.2. The target architecture and the design flow of the proposed architectural synthesis scheme are given in Section 2.3. In Section 2.4, a theoretical formulation to build an initial time schedule taking into consideration the inter-processor communication delay between the nodes of different type is presented. Then, in Section 2.5, based on this initial time schedule, an initial processor allocation matrix is developed. Next, in Section 2.6, the firing times and processor assignments of a pair of nodes of the same type are tested and, if necessary, modified to satisfy the inter-processor communication delay between the two nodes. In Section 2.7, the proposed technique is applied on some benchmark DSP problems. In Section 2.8, the results of the proposed technique are then compared with those obtained by other techniques in the literature when applied to the some intensive DSP benchmark problems. Section 2.9 summarizes the work presented in this chapter and highlights some of the salient features of the proposed technique.

2.2 Data Flow Graph Model for DSP Applications

The data flow graph is proven to be an efficient representation of the system specification due to its ability to expose the hidden concurrency between the operations of the underlying algorithm. Since DSP applications are known for their inherent parallelism, the DFG model is thus suitable for the behavioral representation of DSP applications [13], [22]. A graph *G* can be represented by the pair *(V, E)*, where *V* is a set of nodes, and *E* is a set of elements called edges. Each edge is associated with a pair of nodes.

The symbols $v_1, v_2, ..., v_n$ are used to represent the nodes and the symbols e_1, e_2, ...are used to represent the edges of a graph. A directed edge $e_{ij} = (v_i, v_j)$ is incident out of the node v_i and incident into the node v_j. A directed edge is usually called an *arc*. The nodes v_i and v_j are called the end nodes of the edge e_{ij}. The node v_i is called the initial node, and the node v_j the terminal node of the edge e_{ij}. If $v_i = v_j$, then the edge e_{ij} is called a self-loop. The arcs of DSP graphs represent the precedence constraints between their end nodes.

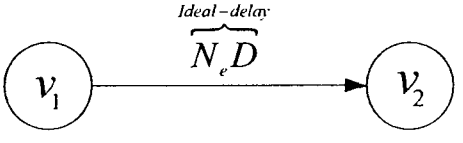


Figure 2. 1: A graph showing an edge with an ideal delay of N_e

If the nodes v_1 and v_2 are, respectively, the initial and terminal nodes of an edge, and the execution of v_2 at iteration *i* is dependent upon the availability of the output of v_1 at iteration $i - N_e$, where N_e is a nonnegative integer, then the edge e_{12} is said to have

associated with it N_e ideal delays. The edge, as shown in Fig. 2.1, is marked as $N_e D$ (i.e., N_e ideal delays). In this case, then the ideal delay N_e represents the inter-iteration dependency between the pair of nodes in question. In contrast, an edge with no ideal delay represents the intra-iteration dependency between the two associated nodes.

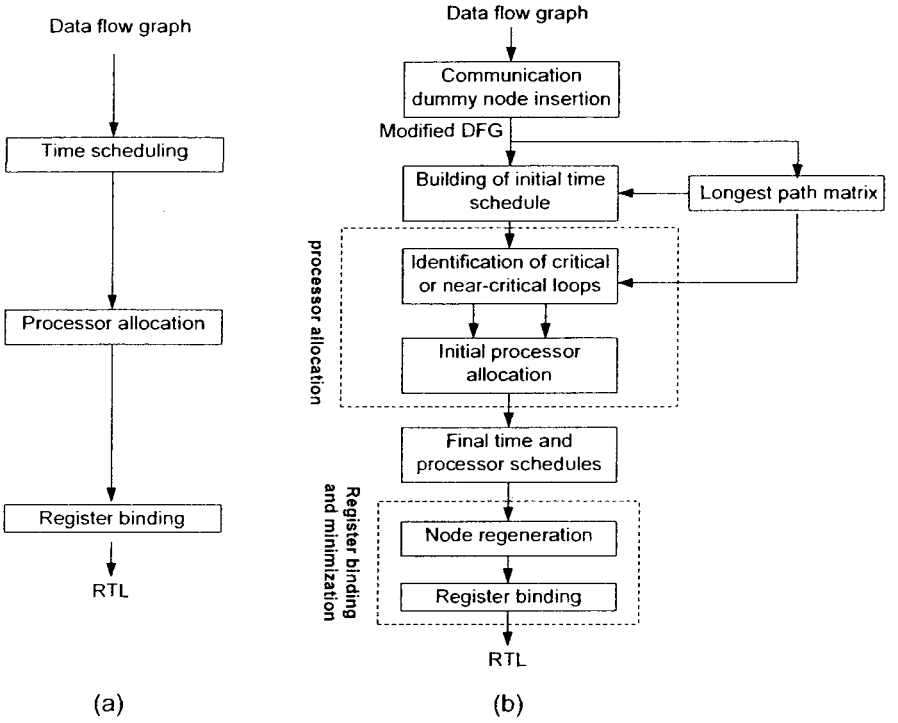


Figure 2. 2: (a) The conventional flow of synthesis (b) The proposed design flow

A direct path $P_{v_0 v_k}$ is a finite sequence of distinct nodes $v_0, v_1, \ldots, v_k$ and distinct edges such that the edge (v_i, v_{i+1}) is present in the path $P_{v_0 v_k}$. If $v_0 = v_k$ then this path is called a directed circuit or loop. Each loop in a DSP graph must contain at least one ideal delay element for the graph to be computable. The data flow graph that contains at least one directed circuit is called the cyclic graph, otherwise it is acyclic.

2.3 Proposed Scheme of Architectural Synthesis

2.3.1 Target architecture

In the traditional architectural synthesis targeting centrally-shared-register based architectures, processing units read data from or write data to a centrally-shared register file through a relatively long interconnect [31]. Such a data transfer is responsible for occupying a portion of the clock cycle. This portion of the clock cycle has now become comparable to the rest of the cycle in the deep submicron technology and could be even larger when technology shifts deeper in the submicron region [32]. Therefore, the interconnect delay plays an important role in determining the cycle time for centrally-shared-register based architectures. Unfortunately, the existing techniques for high-level synthesis for such architectures do not take into consideration the interconnect delays during the synthesis. Instead, the interconnect delay is taken care by adding it to the operation delay part of the clock period thus making the processing unit to remain idle during the period of the data transfer, i.e., during the period of inter-processor communication, which in turn significantly increase the overall execution time of the synthesized centrally shared architecture. However, in a distributed register-based architecture [31][32], registers are distributed so that each processing unit has its dedicated registers that are placed close to it. Accordingly, each processing unit performs two types of data transfer, namely, local and global. The global data transfer between different processing units takes one or more cycles. In the distributed-register based architecture it is possible to separate of the inter-processor communication delay for data transfer from the computation delay of the processing units. The data transfers can then

be dealt with in a way similar to the computations of the processing unit and, hence, can be performed in one or more clock periods. Therefore, in the proposed synthesis scheme, the distributed register-based architecture is chosen as the target architecture. Even though in such a distributed architecture, the number of control steps (cycles) in their time schedule are, generally, higher than that in the time schedule of a centrally-shared-register based architecture, it uses smaller clock period and thus, the wasted slack time, which is the difference between the clock period and computation time, gets reduced. Moreover, a distributed-register based architecture allows operations of the data commutation and data computations in parallel. However, if these operations are not efficiently scheduled to avoid global data transfers, an increase in the number of control steps (clock cycles) in the time schedules and thus an increase in the iteration period will result. Consequently, this may significantly increase the overall execution time of a synthesized recursive DSP system. As done in conventional approaches, the inter-processor communication delay may be taken care in a post-synthesis step. However, the hidden concurrency of the data communication and data computation will not be efficiently exposed if this delay is not included during the synthesis process itself. Therefore, in the proposed synthesis scheme, by targeting a distributed architecture, we attempt to address the resolution of problems which have not been possible by using centrally shared architectures. In the proposed synthesis scheme, the scheduling of data communication and data computation is performed during the synthesis itself, and in order to avoid global data transfer, locality of data computations is maximized.

2.3.2 Design flow

Fig. 2.2 gives an overview of the conventional architectural synthesis flow and the design flow of the proposed scheme. From the behavioural specification of a DSP algorithm such as a DFG, the conventional synthesis produces a structural description at the register-transfer level (RTL) by performing three main tasks, namely, the time scheduling, processor allocation and register binding as shown in Fig. 2.2(a). In view of the fact that ICD need to be taken into consideration, in the proposed scheme, the three synthesis steps are performed in an environment in which the parallelism between the data communication and data computation gets maximally exposed and utilized. The communication delay between a pair of nodes of different types (e.g. the delay between a node performing addition and that performing multiplication) is represented by a non-computing node, whereas that between a pair of nodes of the same type is taken into account by re-adjusting the firing times of the appropriate nodes of the DFG. At the front end of the proposed design flow shown in Fig. 2.2(b), a DFG is taken as input and then modified by inserting communication dummy nodes between nodes of the different types. An initial time schedule is then built to determine the relative firing times of the nodes by using Floyd-Warshall's longest path algorithm [30]. Further, in order to reduce the ICD between nodes of a critical or near-critical loop, such nodes are aimed to be assigned to a single processor. Hence, critical and near-critical loops are first identified. Then based on this identification process and the initial time schedule, an initial processor allocation matrix is developed. Next, the firing times and processor assignment of a pair of nodes of the same type are tested and, if necessary, modified to satisfy the inter-processor communication delay between the two nodes. Finally, by using the scheduled DFG, an

algorithm to carry out the register binding is proposed (to be presented in Chapters 4 and 5.

2.4 Building of Initial Time Schedule

In this section, an algorithm to produce an initial time schedule taking into consideration the inter-processor communication delay (ICD) between the nodes of different type is proposed. A theoretical formulation to build the initial time schedule is presented. It is shown that the problem of finding such initial time schedule can be reduced to the problem of finding the longest-paths between all pairs of nodes of the given DSP graph. Moreover, the feasibility of the produced initial time schedule is proved in this section.

In a heterogeneous distributed-register based architecture, nodes of different types are executed on different processing units, whereas two or more nodes of the same type may or may not be executed on different processors. Recall that in a distributed-register based architecture, the processing unit and its dedicated registers are closely placed and thus communication overhead for data transfer between the nodes assigned to a single processing unit is almost zero, and therefore, can be neglected. However, the data transfers between different processing units are regarded as global communications via long interconnect delay that may take one or more cycles. Thus, the ICD between any two nodes, having precedence dependency, and executed on two different processors has to be taken into consideration, whereas the ICD between any two nodes being executed on the same processor can be neglected. In the proposed technique, in order to take into consideration the ICD between a pair of nodes of different types and having a direct precedent, the original DFG, G, is first altered to give a modified graph, MG, by inserting communication (dummy) nodes as shown in Fig. 2.3. These dummy nodes are not

scheduled; they are used while scheduling other nodes. However, during the building of the initial time schedule in this section, the ICD between a pair of nodes of the same type having a direct precedence is not taken into consideration, since, in order to do so, this would require that the nodes are assigned to processing units beforehand. Furthermore, at this stage of synthesis it is not yet possible to determine whether the two nodes in question can be assigned to a single or to two different processors.

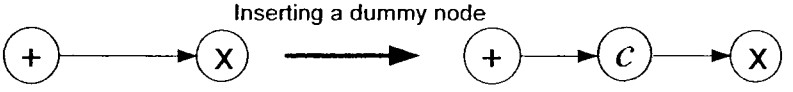


Figure 2. 3: An example of inserting a dummy node.

Let us first define certain terms that are used in this section to build the initial time schedule. Let $EFT(w/v)$ and $LFT(w/v)$ be, respectively, the earliest firing time (EFT) and the latest firing time (LFT) of a node *w* relative to the firing time of the reference node *v*. Further, let $EFT(w)$ and $LFT(w)$ be the earliest and latest firing times of a node *w* relative to all the previously scheduled nodes. The firing time of a node *v* is chosen in the interval of its earliest and latest firing times and denoted by $FT(v)$. The path length, len[$P_{v_i v_j}$], of the path $P_{v_i v_j}$ is defined as the minimum time elapsed between consuming the input operand(s) at its initial node v_i and producing an output at its terminal node v_j, and it is given as

$$\underset{\forall v_i v_j \in V(G)}{len}[P_{v_i v_j}] = \sum_{v \in V(P_{v_i v_j})} d_v - T \cdot N_P \qquad (2.1)$$

where d_v is the computational delay of node v in $P_{v_i v_j}$, T is the iteration period, and N_P is the total number of ideal delays (see Section 2.2) in $P_{v_i v_j}$.

For the modified DFG *MG*, (2.1) becomes,

$$\underset{\forall v_i v_j \in V(MG)}{len}[P_{v_i v_j}] = \sum_{v \in V(P_{v_i v_j})} d_v - T \cdot N_P + ICD_P \qquad (2.2)$$

where $ICD_P = \sum_{c \in ICD(P_{v_i v_j})} d_c$, d_c being the delay of the communication node c in $P_{v_i v_j}$. We define $\underset{\forall v_i v_j \in V(MG)}{len}[P_{v_i v_j}[$ as the path length of path $P_{v_i v_j}$ when the computational delay of its terminal node v_j is excluded. In order to illustrate the computation of a path length, consider the example shown in Fig. 2.4, Assume that the iteration period T is 6 time units, the computational delay for the nodes v_1, v_2 and v_3 are 1, 2, and 1, respectively, and the delay of the communication node (dummy node) c is 1. Using (2.2), $len[P_{v_1 v_3}] = (1+2+1) - (1)(6) + 1 = -1$.

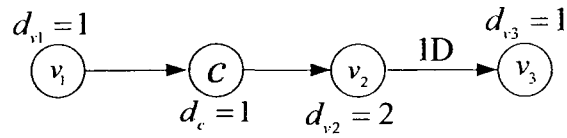


Figure 2. 4: An example for computing the path length.

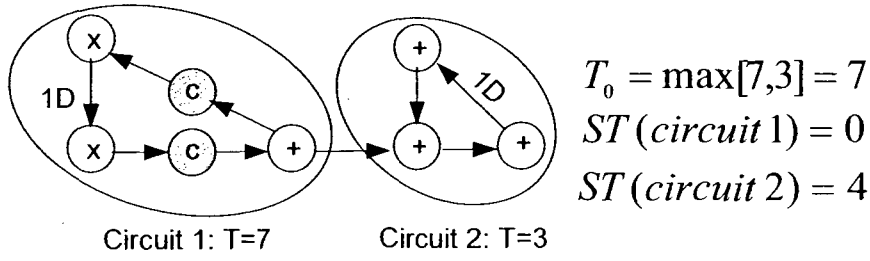


Figure 2. 5: An example illustrating the computations of T_0 and $ST(C)$.

The iteration period bound T_0 for initial time schedule is defined as the minimum time between producing any two successive outputs. For a modified cyclic DFG *MG*, it is given by

$$T_0 = \max_{C \in \text{circuits}} \left\lceil \frac{D_C + ICD_C}{N_C} \right\rceil \tag{2.3}$$

where $D_C = \sum_{v \in V(C)} d_v$, d_v being the computational delay of node v in the circuit C, $ICD_C = \sum_{c \in V(C)} d_c$, d_c being the delay of communication node c in C, N_C is the total number of ideal delays in C, and $\lceil x \rceil$ is the function that returns the smallest integer not less than x.

On the other hand, for a modified acyclic DFG, the iteration period bound is equal to the duration of the longest operation, i.e., $T_0 = \max_{v \in MG}(d_v)$. A circuit C in MG is called critical if its loop bound $\left\lceil \frac{D_C + ICD_C}{N_C} \right\rceil$ is equal to the iteration period bound T_0, A non-critical circuit has a spare time called the slack time. The slack time of a circuit C can be thought of as the total time delay that can be added to the computation encapsulated in the circuit without exceeding the critical loop bound, and is given by

$$ST(C) = T_0 N_C - D_C - ICD_C \tag{2.4}$$

It is clear from (2.2) and (2.4) that

$$ST(C) = - \max_{C \in circuits} len\,[C] \tag{2.5}$$

An example of calculating of T_0 and the slack times of circuits in a given modified DFG is shown in Fig. 2.5. The DFG has two circuits, C1 and C2, which have loop bounds of 7 and 3, respectively. By using (3), T_0 is found to be 7, i.e., the loop bound of C1, and hence, resulting in a slack times of 0 for C1 and 4 for C2.

The theoretical optimal solution for the number of processing units of a certain type of nodes, NP_{type} is given by

$$NP_{type} = \left\lceil \frac{\sum d_{type}}{T} \right\rceil \tag{2.6}$$

In order to determine the effect of the modifying of the original DFG G on the precedence relations of a pair of nodes, and to show that the problem of finding an initial time schedule for the modified DFG MG can be reduced to the problem of finding the longest-paths between all pairs of nodes of the given DSP graph, the following lemmas and theorems are proved.

LEMMA 1 *In a given modified DFG MG, let v be a reference node, and w any other node. Then,*

$$EFT(w/v) = FT(v) + \max_{all\ Pvw} len[P_{vw}[$$

$$LFT(w/v) = FT(v) - \max_{all\ Pwv} len[P_{wv}[$$

where P_{vw} is a path from node v to node w and $FT(v)$ is the firing time of node v

The proof of Lemma 1 is obvious.

LEMMA 2 *The path length between any two nodes of a modified DFG MG is greater than or equal to that between the same two nodes in the original DFG G, that is,*

$$\underset{\forall v_i v_j \in V(G)}{len}[P_{v_i v_j}] \leq \underset{\forall v_i v_j \in V(MG)}{len}[P_{v_i v_j}]$$

Proof. The delay d_c of the communication node c is a positive integer. Subtracting (2.1) from (2.2) results in

$$\underset{\forall v_i v_j \in V(MG)}{len}[P_{v_i v_j}] - \underset{\forall v_i v_j \in V(G)}{len}[P_{v_i v_j}] = ICD_P$$

Since $ICD_P \geq 0$, from the above equation, we have

$$\underset{\forall v_i v_j \in V(G)}{len}[P_{v_i v_j}] \leq \underset{\forall v_i v_j \in V(MG)}{len}[P_{v_i v_j}]$$

Hence, the Lemma □

As mentioned above, the path $P_{v_i v_j}$ is called a loop or a circuit if $v_i = v_j$. Hence, the result in Lemma 2 is also applicable to any pair of nodes of a circuit in the DFG.

THEOREM 1 *The earliest firing time* $EFT\left(v_1/v_0\right)$ *of a node* v_1 *relative to the firing time of node* v_0 *in a modified DFG MG is greater than or equal to that in the original DFG G, and the latest firing time* $LFT\left(v_1/v_0\right)$ *of node* v_1 *relative to the firing time of node* v_0 *in MG is less than or equal to that in G, that is,*

$$\underset{\forall (v_0, v_1) \in V(MG)}{EFT\left(v_1/v_0\right)} \geq \underset{\forall (v_0, v_1) \in V(G)}{EFT\left(v_1/v_0\right)} \tag{2.7}$$

$$\underset{\forall (v_0, v_1) \in V(MG)}{LFT\left(v_1/v_0\right)} \leq \underset{\forall (v_0, v_1) \in V(G)}{LFT\left(v_0/v_0\right)} \tag{2.8}$$

Proof. Assume that $v_0, v_1 \in V(G)$ are two nodes having a path between them. Without loss of generality, assume that v_0 has a fixed time schedule of zero in both G and MG.

Let $EFT\left(v_1/v_0\right)$ and $LFT\left(v_1/v_0\right)$ be, respectively, the earliest and the latest firing times of node v_1 relative to the firing time of the reference node v_0. Assume $P_{v_0v_1}$ is a path between v_0 and v_1. By Lemma 1, we have

$$\underset{\forall(v_0,v_1)\in V(G)}{EFT\left(v_1/v_0\right)} = FT(v_0) + \max_{\text{all } P_{v_0v_1}} len[P_{v_0v_1}[\tag{2.9}$$

$$\underset{\forall(v_0,v_1)\in V(G)}{LFT\left(v_1/v_0\right)} = FT(v_0) - \max_{all\ P_{v_1v_0}} len[P_{v_1v_0}[\tag{2.10}$$

$$\underset{\forall(v_0,v_1)\in V(MG)}{EFT\left(v_1/v_0\right)} = FT(v_0) + \max_{all\ P_{v_0v_1}} len[P_{v_0v_1}[\tag{2.11}$$

$$\underset{\forall(v_0,v_1)\in V(MG)}{LFT\left(v_1/v_0\right)} = FT(v_0) - \max_{all\ P_{v_1v_0}} len[P_{v_1v_0}[\tag{2.12}$$

Using Lemma 2 and comparing (2.9) with (2.11) and (2.10) with (2.12), we have

$$\underset{\forall(v,w)\in V(MG)}{EFT\left(v_1/v_0\right)} \geq \underset{\forall(v,w)\in V(G)}{EFT\left(v_1/v_0\right)}$$

$$\underset{\forall(v,w)\in V(MG)}{LFT\left(v_1/v_0\right)} \leq \underset{\forall(v,w)\in V(G)}{LFT\left(v_1/v_0\right)}$$

Hence, the theorem □

LEMMA 3 *Let MG' be a graph having the same set of nodes and edges as that of the modified graph MG such that the weights assigned to an edge* $e_i = (v_i, v_{i+1}) \in MG'$ *and* $v_i \in MG'$ *are, respectively, given by* $W_{e_i} = d_{v_i} + d_{c_i} - Tn_{e_i}$, *and* $W_{v_i} = 0$, *where* d_{v_i} *is the computational time of node* v_i, d_{c_i} *is the delay of the communication node* c_i *of the edge* e_i *and* n_{e_i} *is the number of ideal delay associated with a given edge* e_i *in MG. Then, the length of any path* $P_{v_1v_m} \subset MG'$ *is equal to the length of the corresponding path*

$P_{v_1v_m} \subset MG$ *excluding the computational delay of the node* v_m, *that is,* $len[P_{v_1v_m} \subset MG'] = len[P_{v_1v_m} \subset MG[$.

Proof. The length of the path $P_{v_1v_m} = (v_1, e_1, v_2, e_2, ..., v_{m-1}, e_{m-1}, v_m) \subset MG'$ is given by

$$
\begin{aligned}
len[P_{v_1v_m}] &= \sum_{e_i \in E(p_{v_1v_m})} W_{e_i} = \sum_{e_i \in E(p_{v_1v_m})} (d_{source(e_i)} + d_{ICD(e_i)} - Tn_{e_i}) \\
&= (\sum_{i=1}^{m-1} d_{v_i} + \sum_{i=1}^{m-1} d_{c_i} - T\sum_{i=1}^{m-1} n_{e_i}) = len[P_{vv} \subset MG'] \\
&= \sum_{v_i \in (V(P_{v_1v_m}) - v_m)} d_{v_i} + \sum_{c_i \in ICD(P_{v_1v_m})} d_{c_i} - T \sum_{e_i \in E(P_{v_1v_m})} n_{e_i} \\
&= len[P_{v_1v_m} \subset MG[
\end{aligned}
$$

Hence, the lemma. □

Now, let Q^0 be an $N \times N$ matrix in which $Q^0_{ij} = W_e = d_{v_i} + d_{ICD(e)} - T \cdot N_e$ for all e=$(v_i, v_j) \in E(MG)$, where N_e is the number of ideal delays associated with the edge $e = (v_i, v_j)$, and $d_{ICD(e)}$ is the delay of the communication node of the edge e. All other entries of the matrix are $-\infty$. Next, applying the Floyd-Warshall's longest path algorithm to Q^0 results in a matrix Q^f, where Q^f_{ij} could be finite or infinite. An infinite value implies that there is no directed path connecting the node v_i to the node v_j, whereas a finite value of Q^f_{ij} represents the longest distance from node v_i to node v_j, i.e., $\max_{\text{all } P_{v_iv_j}} len[P_{v_iv_j}[$.

THEOREM 2 *The difference between the firing times of a pair of nodes in a modified graph MG can be represented in terms of the elements of* Q^f, *i.e., the longest distance between the two nodes.*

Proof. Since $EFT\left(v_j/v_i\right) \le EFT(v_j) \le FT(v_j)$ and $LFT\left(v_j/v_i\right) \ge LFT(v_j) \ge FT(v_j)$, we have

$$FT(v_j) - FT(v_i) \ge EFT\left(v_j/v_i\right) - FT(v_i) \tag{2.13}$$

$$FT(v_j) - FT(v_i) \le LFT\left(v_j/v_i\right) - FT(v_i) \tag{2.14}$$

By Lemma 1, (2.13) and (2.14) become

$$FT(v_j) - FT(v_i) \ge \max_{all\ P_{v_i v_j}} len\,[p_{v_i v_j}[$$

$$FT(v_j) - FT(v_i) \le -\max_{all\ P_{v_j v_i}} len[P_{v_j v_i}[$$

Since $\max_{all\ Pv_i v_j} len[P_{v_i v_j}[= Q_{ij}^f$ and $\max_{all\ Pv_j v_i} len[P_{v_j v_i}[= Q_{ji}^f$, we have

$$Q_{ij}^f \le FT(v_j) - FT(v_i) \le -Q_{ji}^f \tag{2.15}$$

Hence, the theorem □

Using Lemma 1 and Lemma 3, the EFT and LFT for a node v_j relative to that of a reference node v_i are, respectively, given by

$$EFT\left(v_j/v_i\right) = FT(v_i) + \max_{all\ P_{v_i v_j}} len[P_{v_i v_j}[= FT(v_i) + Q_{ij}^f \tag{2.16}$$

$$LFT\left(v_j/v_i\right) = FT(v_i) - \max_{all\ P_{v_j v_i}} len[P_{v_j v_i}[= FT(v_i) - Q_{ji}^f \tag{2.17}$$

To find the earliest and latest firing times of node v_j, the maximum earliest firing time and the minimum latest firing time of the node must be found relative to all the previously scheduled nodes. Thus, EFT and LFT of node v_j are, respectively, given by

$$EFT(v_j) = \max_{all\ i<j} \left(EFT(v_j / v_i) \right) \tag{2.18}$$

$$LFT(v_j) = \min_{all\ i<j} \left(LFT(v_j / v_i) \right) \tag{2.19}$$

The building of the initial time schedule starts by scheduling the input node (an input node in a DFG is the node which consumes data from the input streams) of the modified DFG, and then using it as a reference node for scheduling all other nodes. It is to be noted that without loss of generality, we can assume that a graph has a single input node, since in the case of DFG having multiple input nodes, a zero-delay node can be added to the given graph such as it has a zero-delay edge going to each input node in the original graph. Hence, this newly added node can be considered as the reference node to start building the initial time schedule. The initial time schedule is then built iteratively based on the node mobility. In this technique, the earliest and the latest firing times at which each node can be scheduled to fire are iteratively calculated by using (2.18) and (2.19). The node mobility or the range of control steps at which the corresponding node can be scheduled is equal to the difference between its latest and earliest times. These earliest and the latest firing times are found relative to a reference node and are the result of intra- and inter-iteration precedence constraints.

All the nodes of the modified graph *MG* are first put in a set of non-scheduled nodes, and a schedule is built by selecting a reference node and by calculating the mobility of all the non-scheduled nodes with respect to this reference-node. The node with the minimum mobility calculated thus far is chosen first, and removed from the list of non-scheduled nodes. The chosen node is scheduled to fire at a time that would minimize the communication delay and the number of processors required by examining

all the control steps within the mobility of the node. We define the level of a control step to be the number of nodes of a certain type assigned to fire at this control step. The best firing time is obtained by selecting the control step that has the minimum level. If more than one control step has the same minimum level, one of the following two approaches can be followed to resolve this conflict: (a) For a centrally-shared-register based multiprocessor architecture, the best firing time is obtained by selecting the control step that would minimize the number of registers. (b) For a distributed-register based multiprocessor architecture, the best firing time is obtained by selecting the control step at which the minimum number of nodes having data dependency with the target node are scheduled to be fired. Due to the new firing time of the target node, the time schedule of other non-scheduled nodes may be affected. This newly scheduled node is chosen to be the new reference-node and the earliest and the latest firing times for the rest of the non-scheduled nodes are calculated. A new node is chosen for scheduling and the process is iteratively repeated until all the nodes are scheduled.

It is to be noted that in the above initial scheduling of a modified DFG, if during a given iteration more than one node is found to have the same minimum mobility, then such nodes are treated as special cases for their scheduling. These nodes are chosen as target node for the scheduling according to their predecessor or successor node being a reference node in previous iteration or being already scheduled. Since the mobility of any non-scheduled node may decrease as other nodes are scheduled, the priority order used to select the target node minimizes the consumed mobility of non-scheduled nodes, which, in turn, reduces the impact on the flexibility of scheduling such nodes.

An initial time schedule for a given iteration period T exists, if and only if at all stages of the building of the initial time schedule, the earliest firing time of a non-scheduled node is less than its latest firing time. We will now show by the following theorem that the existence of a valid initial time schedule for a given iteration period $T \geq T_0$ is guaranteed.

THEOREM 3 *At any stage of the time scheduling, the condition* $EFT(v) \leq LFT(v)$ *holds for any non-scheduled node v.*

Proof. Let the nodes be scheduled in the order $v_1, v_2, \ldots, v_N$, that is, node v_i is scheduled only after all the nodes v_k, $k < i$ are scheduled. The node v_i is said to be scheduled at stage i of the scheduling procedure.

Let $\mu_{i,j} = LFT(v_i, j) - EFT(v_i, j)$, where $LFT(v_i,j)$ and $EFT(v_i,j)$ are, respectively, the latest and earliest firing times of node v_i at stage j. Thus, we have to prove that $\mu_{i,j} \geq 0$, for any $i \leq N$ and for all $j \leq i$. However, the mobility of any node may never increase as other nodes are scheduled, $\mu_{i,j} \leq \mu_{i,j-1}$. Hence, we only need to prove that

$$\mu_{i,j} \geq 0 \quad , \; j = i, \; i \leq N \tag{2.20}$$

We will prove the above by employing induction. It is obvious that $\mu_{1,1} \geq 0$, since the graph is computable. Now, let (2.20) be valid for $i = t$, $t < N$. That is, $\mu_{t,t} \geq 0$. We now have to prove the correctness of (2.20) for $i = t+1$. Recall that, at any stage of the proposed scheduling procedure, the node with the minimum mobility is chosen as the target node for scheduling, that is,

$$\mu_{t+1,t} \geq \mu_{t,t}$$

Otherwise, node v_{t+1} would have been chosen before node v_t for scheduling. Hence,

$$\mu_{t+1,t} - \mu_{t,t} \geq 0 \tag{2.21}$$

Obviously, the maximum decrease in the mobility of node v_{t+1} when moving from stage *t to t+1* is $\mu_{t,t}$, which is the final mobility that can be exploited by the scheduling procedure for node v_t just before scheduling it at stage *t*, that is,

$$\mu_{t+1,t+1} \geq \mu_{t+1,t} - \mu_{t,t} \tag{2.22}$$

Combining (2.21) and (2.22), results in

$$\mu_{t+1,t+1} \geq 0$$

Hence, the theorem □

Based on the above theorem and the discussion preceding it, we now give the algorithm for obtaining the initial time schedule. The result of the time schedule is a set of firing times of all the non-communication nodes in the modified DFG, which will be used later in finding the initial processor allocation.

Algorithm 2.1 Initial time schedule

---

1. Calculate the minimum iteration period. Find the longest path matrix Q^f.
2. Take the input node as the reference node and schedule it first to fire at the control step zero.
3. Calculate the earliest and latest firing times of all the remaining nodes with respect to the input node.

4. Calculate the current schedule range or mobility for each of the remaining non-scheduled nodes.
5. Schedule all the nodes that have zero mobility to fire at the only control step in their mobility.

   (Note: There is no need to update the earliest and latest firing times of the remaining nodes after scheduling such a zero-mobility node)
6. Based on the current mobility for each non-scheduled node, choose one node as the target node for the scheduling according to the following priority:
   a. A node that has the minimum current finite mobility. If more than one node has minimum current finite mobility, chose from these nodes the one that is a predecessor or successor to the current reference node.
   b. A node that is a predecessor or successor to the current reference node
   c. A node that is a predecessor or successor of any scheduled node.
7. Within the scheduling range of the target node, find the best firing time position as the control step that has the minimum level. If more than one control step results in the same minimum possible level, proceed as follows: (i) For distributed-register based architecture, choose the control step with the minimum level that results with the minimum number of nodes having a data dependency with the target node. (ii) For centrally-shared-register based architecture[1], choose the control step with the minimum level that would minimize the number of registers.
8. Set the best firing time position found as the time schedule of the target node.
9. Set the target node to be the new reference node.

1. Algorithm 2.1 can also be applied to the synthesis aiming at centrally-shared-register based architectures in which dummy nodes can not be inserted in the DFG

10. Update the earliest and latest firing times of all the remaining non-scheduled nodes.

11. Go to Step 4 until all the nodes have been scheduled.

2.5 Initial Processor Allocation Algorithm

In this section, an initial processor allocation scheme is proposed. Through this scheme, the nodes of the modified DFG are assigned to the processors of a heterogeneous multiprocessor system, and the set of nodes of a certain type in each critical or near-critical loop (a near-critical loop is the loop whose loop bound is close enough to the critical loop bound according to some criterion) are aimed to be assigned to the same processor, which in turn results in reducing the inter-processor communication delay (ICD) between the nodes of the same type. Thus, identification of each critical or near-critical loop is crucial for processor allocation, and for this purpose, we first give an algorithm to perform identification of such loops and use it later for the initial processor allocation. A loop identification scheme can be found in [34].

A critical or near-critical loop is found by identifying the nodes of such a loop. The process of this identification is carried out as follow. The matrix Q^f is used to identify the set of nodes in the longest loops (critical or near-critical) in the modified graph *MG*. Since each diagonal element in Q^f represents the longest path from the corresponding node to itself, the nodes with the largest diagonal entries in Q^f are on a critical loop. The nodes of a near-critical loop have the diagonal entries whose values are less than those of the entries corresponding to the critical loop. The values of the diagonal entries of the

matrix Q^f are used as a guide to find the set of the nodes that form a critical or near-critical loop. Fig. 2.6 gives an example as to how Q^f is used to identify a critical or near-critical loop. In this example, since the diagonal entries of Q^f corresponding to nodes v_1, v_2, v_3, and v_4 of the given DFG are the largest, i.e., zeros, these nodes belong to a critical loop. On the other hand, the diagonal entry corresponding to node v_5 has a smaller entry of value -1 ($Q^f_{55} = \max_{\text{all } P_{v_5 v_5 j}} len[P_{v_5 v_5}[= (1+1+1)-1(4) = -1$). Hence, this node belongs to a near-critical loop. However, node v_5 alone does not form a loop, since there is no self-loop from the node to itself in the graph. Therefore, node v_5 must belong to at least one loop connecting to some of the other nodes v_1, v_2, v_3, and v_4 in the graph. Hence, we need to find the complete set of the nodes that forms a critical or near-critical loop.

The following algorithm gives a scheme that identifies all the nodes belonging to a particular critical or near-critical loop. This algorithm also determines the criticality level of a loop, which is a measure of closeness of the loop to the critical loop bound.

Algorithm 2.2 Identification of critical or near-critical loops

---

1. Choose any node with the largest diagonal entry as target node v_r. Use i to denote the criticality level of the loop and j the number of the loop within the criticality level i. Set $i=0$ *and* $j=0$.
2. Add the target node v_r to the set $LP_{ij}(t)$, where t is the type of the target node; set the current target node as a search node v_s.

3. Among all the nodes next to the node v_s, select the node v_k satisfying the following three conditions:

   (a) Node v_k has the maximum diagonal entry value, (b) $Q^f_{sk} = Q^0_{sk}$, and (c) $Q^f_{rk} + Q^f_{kr} = Q^f_{kk}$. In the case, there is more than one such node v_k, arbitrarily chose one of them. Add the selected node to $LP_{ij}(t)$

4. Repeat Steps 3 until the current target node v_r is reached and a new critical or near-critical loop containing the node v_r is identified. $j=j+1$.

5. Choose a node from the remaining uncovered nodes that have a diagonal value in Q^f equal to the diagonal value of the current target node as the new target node; go to Step 2. If there is no such a node, go to Step 6.

6. Choose any node with the next largest diagonal value to be the new target node. Set $i=i+1$

7. Repeat Steps 2-6 until a specified percentage β of all nodes with a finite diagonal entry in Q^f has been covered. The terminating parameter β is the percentage of nodes regarded as critical or near-critical and it obviously dependents on the choice of the level of criticality that will allow a reasonable percentage of nodes to be considered as critical or non-critical.

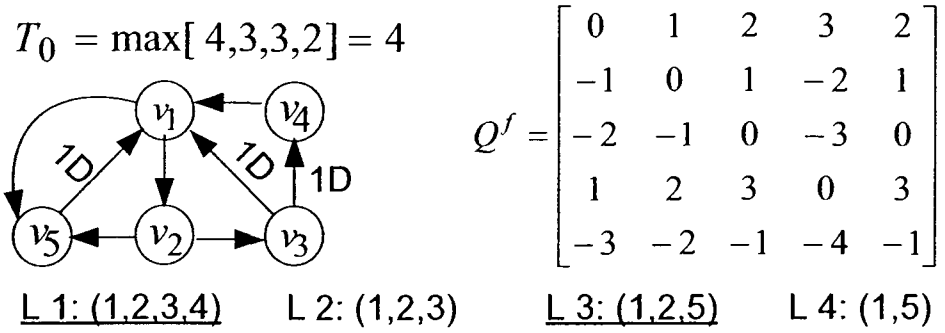


Figure 2. 6: An example of critical and near-critical loops.

Even though the DFG given in Fig. 2.6 has 4 loops, L1-L4, only two loops are identified, namely, L1 (v_1, v_2, v_3, and v_4) and L3 (v_1, v_2, and v_5), by applying Algorithm 2.2. This is so, since the algorithm has been designed to identify only a loop that has at least one node belonging to it that is not covered by any other more critical loop. Thus, loops L2 and L4 are not identified, since the sets of nodes (v_1, v_2, and v_3) in L2 and (v_1 and v_5) in L4 are subsets of the sets of nodes in L1 and L3, respectively. It is to be noted that despite the fact that node v_5 has different diagonal entry from those of v_1 and v_2, v_5 is contained together with nodes v_1 and v_2 in L3, and with node v_1 in L4. In this example, L1 is the only critical loop with criticality level i=0. On the other hand, i=1 for L3.

It is to be noted that, by using the above algorithm, a node may appear in one or more than one loop with the same or different critically levels. This node duplication is removed by making it to belong to only one loop that has the highest criticality level.

Based on the initial time schedule obtained in Section 2.4, and by using the loop identification algorithm developed in this section, an initial processor allocation is now produced. The result of this allocation is an initial processor allocation matrix *A(t),* which

specifies as to which processor of type t a certain node is assigned. This scheme of processor allocation starts by trying to assign the complete set nodes of a certain type in each critical or near-critical loop to a single processor of type t according to the criticality level of the loop to which they belong. The non-critical nodes, i.e., the nodes not belonging to any critical or near-critical loops, are then assigned iteratively to the processors until all the nodes of type t of MG are exhausted. In the initial processor allocation, the complete set of nodes of a certain type in each critical or near-critical loop may or may not be assigned to the same processor. If a pair of nodes in any critical or near-critical loop cannot be assigned to the same processor, then the edges connecting such a pair of nodes are referred to as cutting edges and assigned an urgency level equal to the criticality level of the loop, since more the criticality level of a loop, more the urgency for its nodes to satisfy the ICD. Therefore, the urgency levels of the edges can be used prioritize the testing of the edges for their ICD compatibility. As to be seen in Section 6, the urgency levels of the cutting edges will be needed in building the final time and processor schedule. However, since during the phase of initial processor allocation, we determine which nodes of critical or near-critical loop cannot be assigned to a single processor of the same type, the process of marking of the cutting edges with the urgency levels can be conveniently carried out at the same time. The process of the initial processor allocation and the urgency marking of the cutting edges is described in Algorithm 2.3.

Algorithm 2.3 Initial processor allocation and urgency marking of the cutting edges

1. Create allocation matrices $A(t)$'s of order $P_t \times T$, i.e. one matrix for each type of processors, where t is the type of the processors, P_t is the number of processors of type t and T is the iteration period. Set $P_t=0$, $i=0$, $j=0$.
2. For each node $v \in LP_{ij}(t)$, determine the columns as $FT(v)$ *Modulo* T and determine the row only if one can find the first available processor that is free in these columns and thus it can accommodate all of the nodes in $LP_{ij}(t)$, then set $j=j+1$ and go to Step 6.
3. Add a new processor of type t and create a corresponding new row in the allocation matrix; assign as many of remaining nodes in the set $LP_{ij}(t)$ as possible to this new processor and thus $P_t = P_t + 1$. If each nodes of the of the set $LP_{ij}(t)$ has been assigned to a processor, set $j=j+1$ and go to Step 6.
4. Mark each of the cutting edges arising from Step 3 in the loop correspond to $LP_{ij}(t)$ with an urgency level equal to i

   (Note that the first failure to accommodate all the nodes of a loop to a new added processor will result in two cutting edges whereas succeeding failures will create only one cutting edge)
5. Search for the first available processor that can accommodate the remaining nodes in $LP_{ij}(t)$ and assign them to this processor and set $j=j+1$. Otherwise go to Step 3.

6. If $j \le j_{nl}$, where j_{nl} is the number of loops in *MG* with the criticality level i as determined by *Algorithm 2,* go to Step 2. Otherwise, set $i = i+1$. If $i \le i_{nc}$, where i_{nc} is the total number of criticality levels as determined by *Algorithm 2,* go to Step 2.
7. If *MG* does not have a non-critical node that has yet to be assigned to a processor, stop.
8. Choose one of the non-critical nodes of type t from *MG* that have not been assigned yet to a processor. Determine the corresponding column as $FT(v)$ *Modulo* T and the corresponding row by finding the first available processor of the type t that is free in this column and thus can accommodate this node; assign the node to this processor.
9. If during Step 8, the node cannot be assigned to an existing processor, add a new processor of type t and thus create a new corresponding row in the allocation matrix $A(t)$. Assign the node to this processor and thus $P_t = P_t + 1$. Go to Step 7

2.6 Final Time and Processor Schedule

The initial time schedule and processor allocation as obtained in Sections 2.4 and 2.5, respectively, did not take into account the ICDs of the nodes of the same type. Since the ICDs of such nodes are not negligible, the initial time schedule and processor allocation may not be valid, if in *MG* the nodes of the same type having direct dependency could not be assigned to the same processor. In this section, the ICDs between a pair of nodes

of the same type assigned to two different processors are now taken into consideration in order to find the final time and processor schedule.

The ICD between a pair of nodes of the same type assigned to two processors is tested with respect to its compatibility with the firing times of the two nodes. In order to take this ICD into consideration, a communication dummy node is inserted in the edge that connects the two nodes. Such an insertion may violate the firing times of the two nodes which in turn may also violate those of the other nodes in the modified graph *MG*. If such a violation occurs for a pair of nodes, the firing times of the pair should be modified to satisfy the inter-processor communication delay between them.

To determine the impact of inserting a communication node into an edge $e_{ij} = (v_i, v_j)$ that connects the two nodes v_i and v_j of the same type but assigned to two different processors, the earliest scheduling time (*EST*) and the latest scheduling time (*LST*) of the communication node $c_{e_{ij}}$ with respect to the firing times of the two nodes are calculated as follows,

$$EST(c_{e_{ij}}) = FT(v_i) + d_{v_i}$$

$$LST(c_{e_{ij}}) = FT(v_j) - (d_{c_{e_{ij}}} - Tn_{e_{ij}})$$

where d_{v_i} is the computational delay of the node v_i, $d_{c_{e_{ij}}}$ the delay of the communication node $c_{e_{ij}}$, and $n_{e_{ij}}$ the number of ideal delays associated with the edge e_{ij}. The time difference between the latest scheduling time and the earliest scheduling time of the communication node $c_{e_{ij}}$ is called the mobility, $M(c_{e_{ij}})$, of inserted node $c_{e_{ij}}$, and it is given as

$$M(c_{e_{ij}}) = LST(c_{e_{ij}}) - EST(c_{e_{ij}})$$

The mobility of the communication node $c_{e_{ij}}$ is required to respect the ICD, $d_{c_{e_{ij}}}$, between the two nodes by imposing the constraint,

$$M(c_{e_{ij}}) \geq 0 \tag{2.23}$$

If this condition is satisfied, then there is no need to modify the firing times of the two computational nodes in *MG*. On the other hand, if this condition is not satisfied, then the firing times of the nodes v_i and v_j are adjusted so that the condition given by (2.23) is satisfied with the equality sign. The adjustment of the firing times of the two nodes is carried out by employing the procedure *shift-successor- predecessor,* which is used to widen the mobility of $c_{e_{ij}}$ to accommodate the communication node. The procedure *shift-successor-predecessor* is constructed based on two sub-procedures, *shift-successor-left* and *shift-predecessor-right.* In each of the two sub-procedures, the nodes predecessors to the node v_i and nodes successors to the node v_j are iteratively shifted while satisfying the precedent relation and keeping the current processor allocation unchanged. The procedure *shift-successor-predecessor,* the main procedure, uses the two sub-procedures in an iterative manner. It returns a "*true*", if in any iteration, the mobility of $c_{e_{ij}}$ is widened enough to satisfy the condition in (2.23), otherwise it returns a "*false*".

If the *shift-successor- predecessor* procedure fails in widening the mobility, then it is done so by inserting a number of cycles, N_{ins}, required to satisfy the condition in (2.23), into the allocation matrix starting from the control step $C_{ins} = EST(c_{e_{ij}})$ modulo T. Thus, the iteration period of the schedule is increased

by N_{ins}. As a result of this insertion of the new cycles into the allocation matrix, the firing times of all the nodes in *MG* are modified.

The set of cutting edges that connect a pair of nodes of the same type assigned to two processors are sorted for the testing for ICD compatibility according to the urgency levels of the edges as obtained by Algorithm 2.3 of Section V. An example for illustrating the insertion of new cycles into the allocation matrix is given in Fig. 2.7. In this example, all the nodes are assumed to be of the same type. Nodes v_1, v_2, and v_3 are assigned to a single processor P1, whereas v_4, v_5 and v_6 are assigned to P2. Therefore, there are four cutting edges, namely, e_{24}, e_{41}, e_{15}, and e_{63}, that have to be tested according to their urgency levels as shown in Fig. 2.7. Edges e_{24} and e_{41} have the highest urgency levels, each of which in turn leads to an insertion of one cycle as shown in Fig. 2.7, in order to satisfy the ICDs.

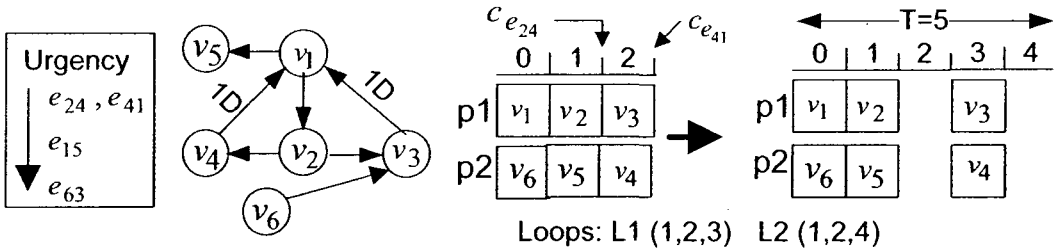


Figure 2. 7: An example of insertion of new cycles into the allocation matrix for ICD compatibility between nodes of the same type.

Inserting new cycles into the allocation matrix would create an additional empty space in each processor. This newly created space may be useful in eliminating the ICD between certain pairs of nodes of the same type, if more nodes could be allocated to same processor. These empty spaces can be filled by moving the firing times of some of the

nodes in the empty spaces of the processor executing these nodes or by moving the firing times of these nodes even to the empty spaces of the processors. Such a moving is carried out, if and only if, it results in no new edges to be added to the list of edges required to be tested for their compatibility with the respective ICDs. A terminal node v_j of the edge e_{ij} in the list of edges to be tested is the candidate nodes to occupy the newly created empty spaces, if it satisfies the condition $FT(v_j)\ modulo\ T < C_{ins}$. Starting from the candidate terminal node v_j of the most urgent edge e_{ij} in the list of edges, we calculate the latest firing time $LFT(v_j)$ of the node in question with respect to all of its successor nodes, i.e., $LFT(v_j) = \min_{v_{succ} \in successor\ nodes} LFT(v_j/v_{succ})$. If $LFT(v_j) \geq C_{ins}$, then the firing time of v_j is moved to a processor chosen from the candidate processors, that results with the largest number of nodes to have direct edges with the node in question v_j. After such a node movement, not only the edge e_{ij} that contains the node v_j is removed from the list of the edges to be tested for ICD compatibility, but also all other edges e_{ij} or e_{jk} for which the nodes v_i or v_k get assigned to the same processor to which the node v_j has been assigned. This node movement procedure, referred to as *move_procedure*, is iteratively repeated until all the candidate nodes to occupy the newly created empty space are tested. If none of the candidate nodes can be accommodated by the created empty space of any of the processors, the created empty space(s) is kept vacant. For the example of Fig. 2.7, this *move-procedure* can be applied to move node v_5 to occupy the empty space at control step 2 which, in turn, satisfies the ICD between nodes v_1 and v_5, and therefore, e_{15} is removed from the list of edges to be tested. The only remaining edge in

the list of the cutting edges to be tested according to (2.23) is e_{63}. This edge is already compatible with respect to the ICD between nodes v_6 and v_3. Based on the above discussions, we now give an algorithm for obtaining the final time and processor schedule.

Algorithm 2.4 Final time and processor schedule

---

1. List all the edges in the modified graph that connect a pair of nodes of the same type assigned to two processors of the same type.
2. For each edge in the list, assign an urgency level equal to the corresponding level determined in the initial processor allocation algorithm (Algorithm 2.3). If no urgency level was assigned to an edge during the initial processor allocation, then assign to each of such edge a fixed urgency level value that is less than the smallest level found in the initial processor allocation.
3. Sort the list of edges according to their decreasing urgency levels. If two or more edges result in being assigned the same urgency level, then these edges are sorted according to the order in which their termination nodes were scheduled to be fired in the initial time schedule.
4. If the sorted list of edges is empty, then stop.
5. Select the first edge in the list; insert a communication (dummy) node $c_{e_{ij}}$ to this edge in *MG*. Then remove this edge from the list
6. Calculate the earliest and latest scheduling time, $EST(c_{e_{ij}})$ and $LST(c_{e_{ij}})$, for the communication node $c_{c_{ij}}$ inserted to the edge e_{ij} in *MG*.
7. Calculate $M(c_{e_{ij}})$ of the communication node.

8. If $M(c_{e_{ij}}) \geq 0$, go to Step 4.

9. Call the procedure *shift-successor- predecessor*. If it returns a *"true"* value, then go to Step 4.

10. Calculate the number of cycles $N_{ins} = -M(c_{e_{ij}})$ required for a valid insertion of the communication node $c_{e_{ij}}$ into the edge e_{ij}.

11. Insert N_{ins} into the allocation matrix starting from control step $EST(c_{e_{ij}}) \ Modulo \ T$. Set $T = T + N_{ins}$

12. Update the firing times of all the nodes in *MG*.

13. Call the node movement procedure, *move_procedure*.

14. Go to Step 4.

2.7 Experimental Results and Discussions

In this section, some well-known benchmark problems of synthesizing DSP filters using the technique presented in this chapter are considered. We have implemented the proposed algorithms in C++ and performed tests on a Pentium IV (1.7 MHz) machine. Starting from the DFG corresponding to a given DSP algorithm, the process of synthesis is applied to obtain the time schedule and processor allocation. In our experiments, a distributed register based architecture and an inter-processor communication delay of 1 cycle are assumed. Moreover, structurally pipelined processing units are used in the synthesis of all of the benchmark problems considered.

2.7.1 A fourth-order all-pole lattice filter

Fig. 2.8 shows an example of synthesizing a fourth-order all-pole lattice filter by applying the proposed technique. The modified DFG of a fourth-order all-pole lattice

filter is shown in Fig. 2.8(a). The initial iteration period bound for the modified DFG is obtained using (2.3) and it consists of 18 cycles. The computational delays of addition and multiplication nodes are assumed to be 1 and 5 cycles, respectively. Adders and multipliers each with a structural pipeline of 1 stage and 5 stages, respectively, are used in this example. The time and processor schedules obtained by applying the proposed technique are given in Fig. 2.8(b), which shows that this schedule results in an iteration period of 19 cycles. The difference between the initial iteration period bound and the one finally obtained is due to the inter-processor communication delay between nodes of the same type.

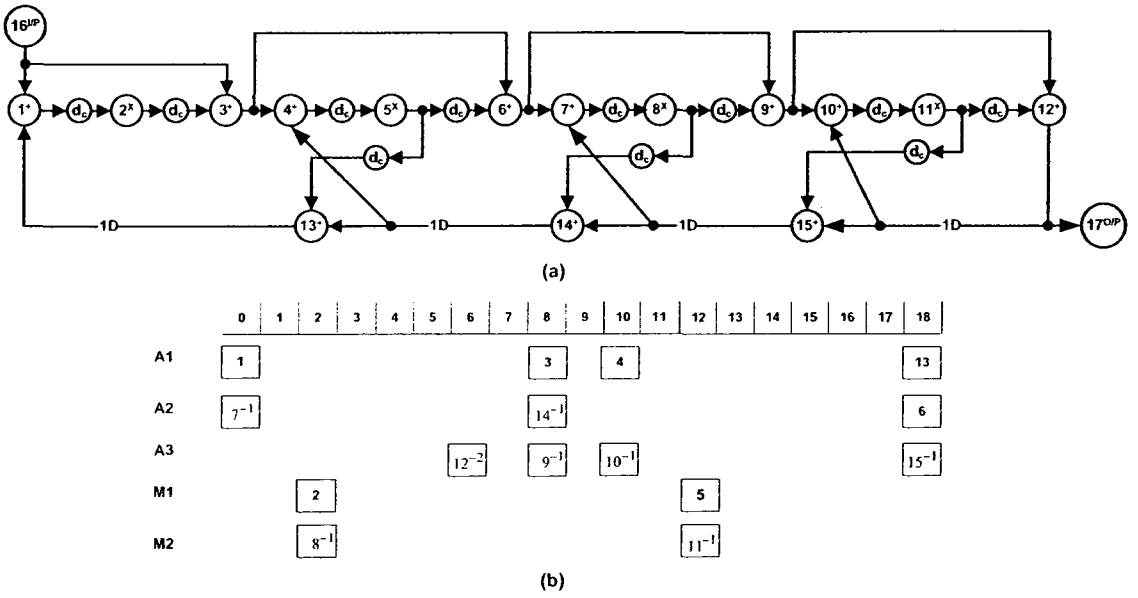


Figure 2. 8: Example of applying the proposed technique for the synthesis of a fourth-order all-pole lattice filter (a) the modified DFG of the filter (b) the time and processor schedules.

2.7.2 A fourth-order Jaumann wave filter

Fig. 2.9 shows an example for synthesis of a fourth-order Jaumann wave filter by applying the proposed technique. The modified DFG of a fourth-order Jaumann wave

filter is shown in Fig. 2.9(a). The initial iteration period bound for the modified DFG consists of 20 cycles. Just as in the previous example, the computational delays of addition and multiplication nodes are assumed to be 1 and 5 cycles, respectively. Adders and multipliers each with a structural pipeline of 1 stage and 5 stages, respectively, are also used in this example. The synthesis of this filter is carried out using the proposed technique giving the time and processor schedules shown in Fig. 2.9(b). It is seen from this figure that the iteration period consists of 21 cycles and the architecture using this schedule would require two adders and one multiplier. In order to minimize the number of registers, the proposed node regeneration scheme is applied to this filter.

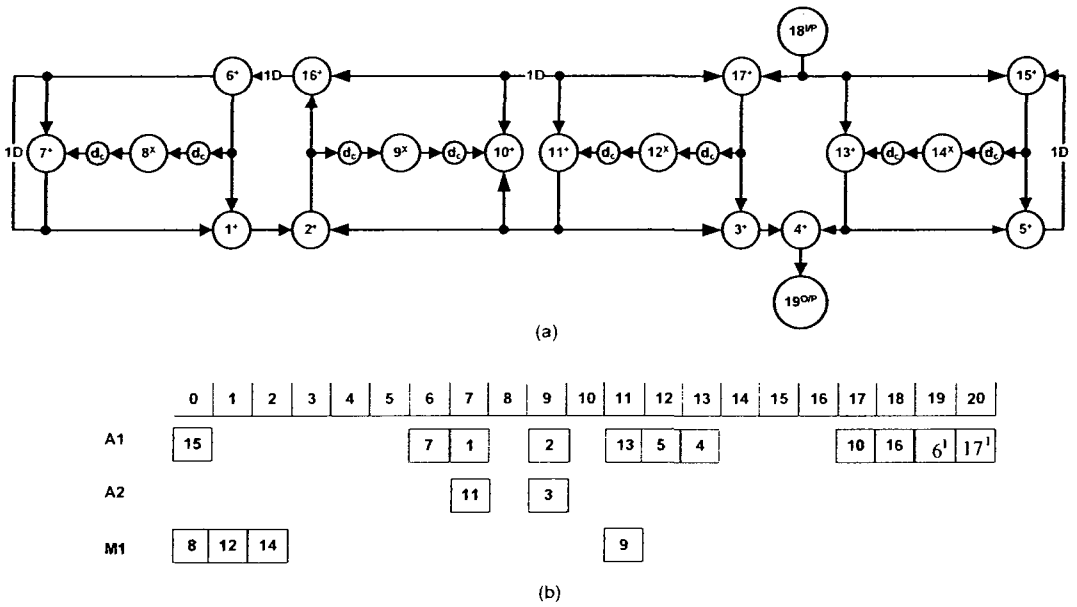


Figure 2. 9: Example of applying the proposed technique for the synthesis of a fourth-order Jaumman filter (a) the modified DFG of the filter (b) the time and processor schedules.

2.7.3 A fifth-order elliptic wave filter

Fig. 2.10 shows an example for the synthesis of a fifth-order elliptic wave filter by applying the proposed technique. The modified DFG of the filter is shown in Fig. 2.10(a).

The initial iteration period bound for the modified DFG consists of 22 cycles. The computational delays of addition and multiplication nodes are assumed to be 1 and 2 cycles, respectively. Adders and multipliers each with a structural pipeline of 1 stage and 2 stages, respectively, are used in this example. Fig. 2.10(b) shows the time and processor schedules obtained by using the proposed technique. In the modified DFG of this example, there are two critical loops having a loop bound of 22 cycles and one near-critical loop with a loop bound of 21 cycles as found by using the Algorithm 2.2.

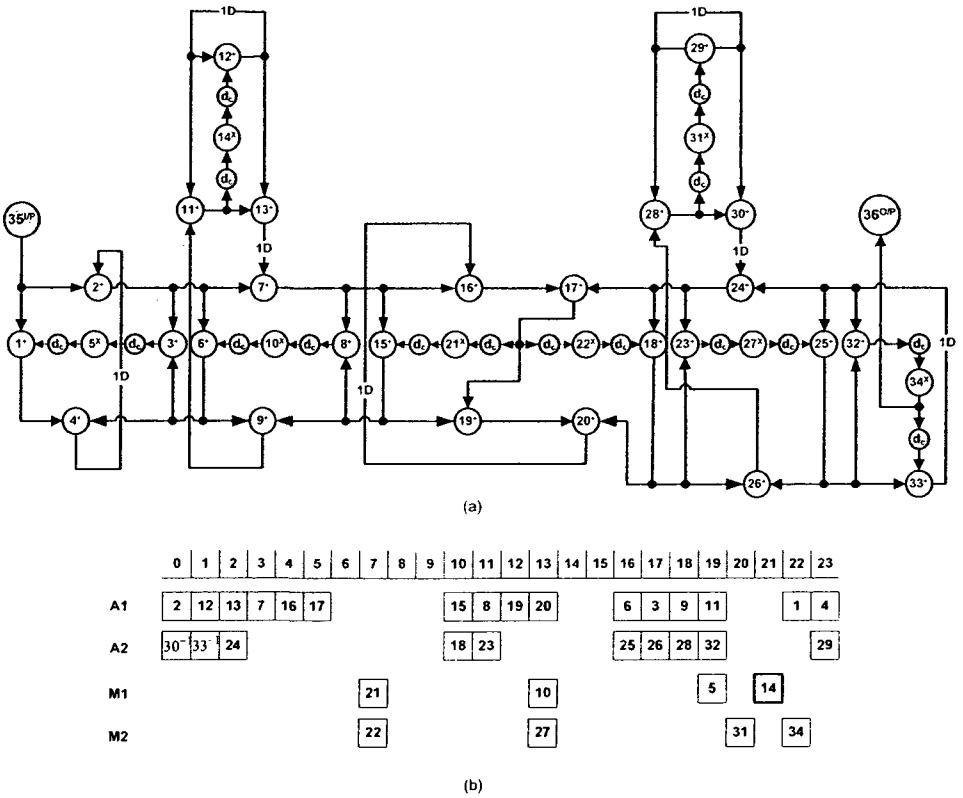


Figure 2. 10: Example of applying the proposed technique for the synthesis of a fifth-order elliptic wave filter (a) the modified DFG of the filter (b) the time and processor schedules.

These three loops have node 17 as a common node, thus making it necessary that a single processor accommodates all the nodes belonging to these loops. However, in

order to do so in the particular example under consideration, the minimum iteration period must comprise at least 26 cycles. Since the process of allocating the nodes to the processors is carried out based on the initial time schedule, which in this example has the initial iteration period of only 22 cycles, all the nodes of the three loops cannot be allocated to a single processor. Hence, the nodes in question are allocated to two processors. Consequently, two inter-processor communication delays are introduced in order for node 17 to communicate with other nodes of the same type having precedent relations with it and assigned to different processors. This, as shown in Fig. 2.10(b), results in a final time schedule with an iteration period consisting of 24 cycles.

2.8 Comparisons with Previous Work

In this section, the proposed technique is compared with some of the other techniques in the literature in terms of the iteration period and number of processing units required for the synthesis of the filter considered in Section 2.7. It is also compared with commonly used scheme in terms of the overall execution time for some intensive DSP benchmarks.

2.8.1 Comparison of various schemes in terms of on the iteration period with and without ICD

The well-known techniques FDLS [18], ALPS [19], OSAIC [20], InSyn [21], and MARS [22] have neglected the ICD when developing their synthesis technique. Hence, in order to compare the results of the proposed synthesis technique with that of these techniques, we have assumed that ICD is zero when applying the proposed technique of synthesis. Table 2.1 gives the iteration period for the fifth-order elliptic filter when synthesized under a given resource constraint by using the various techniques. It is seen

from this table that only the technique of MARS and the proposed one give the lowest iteration period possible, namely, the iteration period bound. It is to be noted that the iteration period obtained by either method is unrealistic, since the ICDs have been neglected; however, unlike the technique of MARS, the proposed synthesis technique is capable of including non-zero ICDs in the technique. Table 2.2 gives the synthesis results in terms of the iteration period and resource requirements obtained for the elliptic wave filter considered in Section 8 by using the proposed technique and the techniques of [35]-[37]. It is seen from this table that the iteration period obtained by using the proposed technique consists of 24 cycles which is larger than the iteration periods obtained by the techniques [35]-[37]. However, this higher value of the iteration period should be viewed in the context that the iteration period bound using the proposed technique consists of 22 cycles for this filter and, as discussed in Section 2.7.3, this bound is unattainable if the ICDs are taken into consideration.

Further, the works in [35]-[37] synthesize a fifth-order elliptical wave filter for a two-chip implementation. Each chip has one adder and one two-stage pipelined multiplier. In these techniques, inter-processor communication delay is considered only between the chips (inter-chip communication) and it is neglected between a pair of internal processing units within each chip. With the progress in deep submicron VLSI technology, the inter-connect delay has become larger than the gate delay, thus making the inter-connect delays a dominant factor of the overall delay in a chip. Thus, neglecting the internal (intra-module) inter-processor communication delay in a chip, as is done in [35]-[37], would be unrealistic or the cycle time in their resulting architectures must be larger to accommodate the intra-module ICD.

Table 2. 1: Results on iteration period with ICD assumed to be zero for a fifth- order elliptic wave filter synthesized by various techniques for a given resource constraint

Processors	Technique					
	FDLS	ALPS	MARS	InSyn	OSAIC	Proposed
1 A, 1 PM	N/A	29	28	29	N/A	28
2 A, 1 PM	19	19	17	19	19	17
3 A, 1 PM	18	18	16	18	18	16

Table 2. 2: Results on iteration period taking ICD into consideration for a fifth order elliptic filter using various synthesis techniques

Technique	Iteration period	Processing units	ICD	
			Inter-module	Intra-module
APARTY [35]	21	2 chips, each with 1 adder 1 multiplier	Taken	Not taken
VULCAN [36]	21	2 chips, each with 1 adder 1 multiplier	Taken	Not taken
Method of [37]	18	2 chips, each with 1 adder 1 multiplier	Taken	Not taken
Proposed	24	2 adders and 2 multipliers	Taken	Taken

2.8.2 Comparison of the proposed and Force-Directed List-Based scheduling [40] schemes in terms of the overall execution time and number of control steps for some intensive DSP benchmarks

In this section, the proposed scheme of synthesis targeting a distributed-register based architecture is first compared with a commonly used scheme, namely, force-directed list-based scheduling [40], targeting a centrally-shared-register based architecture. The comparison is made in terms of the overall execution time, measured as a product of the number of control steps and the duration of the step (i.e., the clock period), of the RTL

architectures resulting from the application of the two schemes on a number of intensive DSP benchmarks [41][42]. The number of operational nodes in the intensive DSP benchmarks considered varies from 34 to 547. For the purpose of this experiment, the delay of an adder is assumed to be 5 ns and that of a multiplier 10 ns. The ICD value is set as 5 ns. The values of the two register parameters, t_{setup} and t_{clk2Q}, are chosen so as to provide $t_{setup} + t_{clk2Q}$=2 ns. The clock period is determined by using the technique of [43]. Fig. 2.11 shows the execution times of the architectures corresponding to the various DSP benchmarks. It is seen from this figure that the proposed scheme provides a significant gain over that of the force-directed list-based scheduling. An average gain of 33.8% is achieved for the intensive DSP benchmarks considered in this experiment with the maximum gain being 40.5% in the case of the specific benchmark, *DCT-dir*. Since the ICD in distributed register based architecture may be taken care in a post-synthesis, we compare the results of the number of control steps (i.e., the iteration period) by obtaining the synthesis results from the proposed scheme for two cases respectively. In the first case the ICD is taken care during the synthesis itself, whereas in the second case it is done in a post-synthesis phase. Fig. 2.12 shows the number of control steps for couple of intensive DSP benchmarks, namely, DCT-feig and DCT-chem, architectures for these two cases of the synthesis. The comparison is carried out for different number of processing units in the target architecture. It is seen from Fig. 2.12(a) and (b) that the proposed scheme of incorporating the ICD during the synthesis provides a significant reduction in the number of control steps over the one in which it is done in the post-synthesis phase.

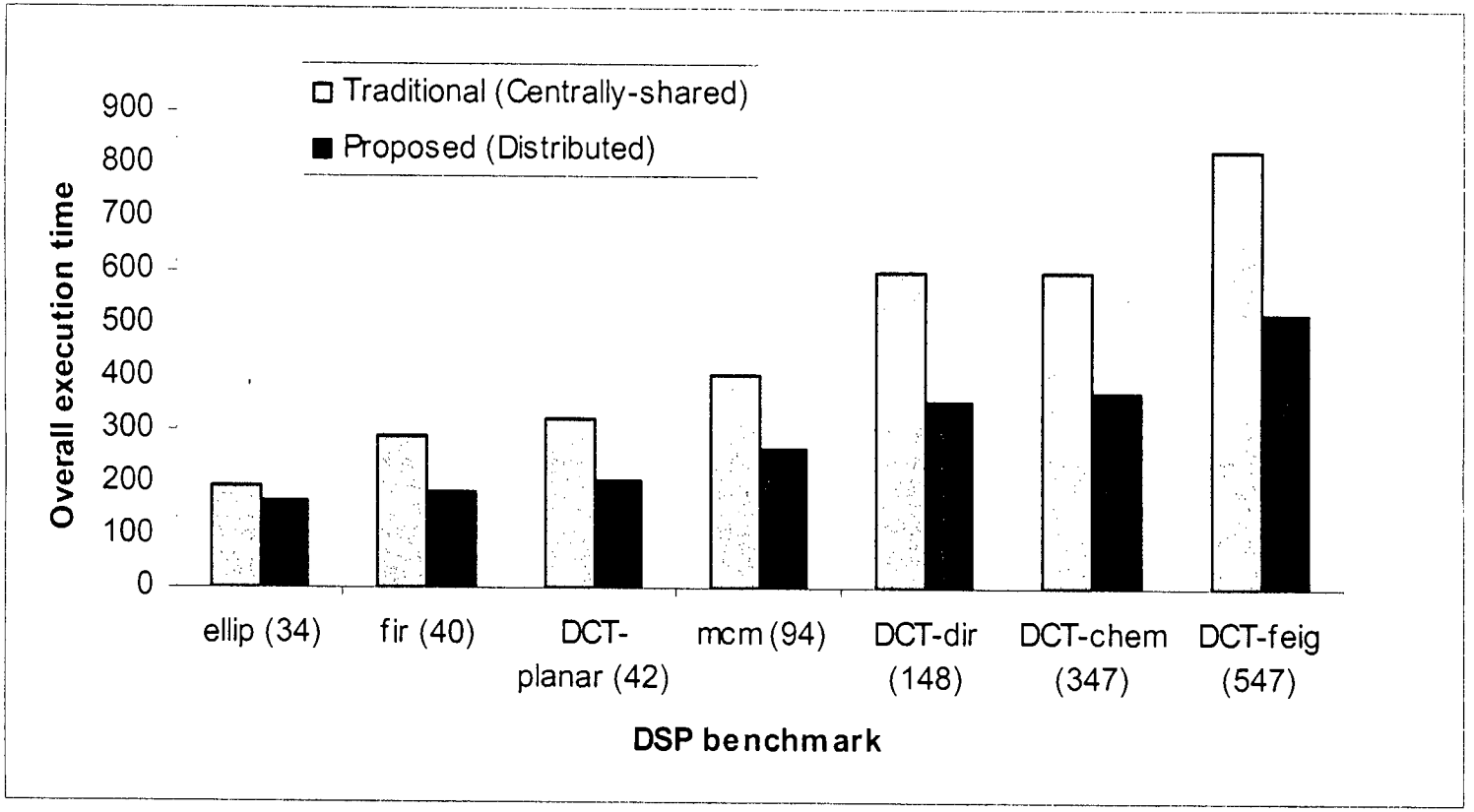


Figure 2. 11: The overall execution time of different intensive DSP benchmarks. The quantity within the parenthesis for each benchmark specifies the number of operational nodes associated with it.

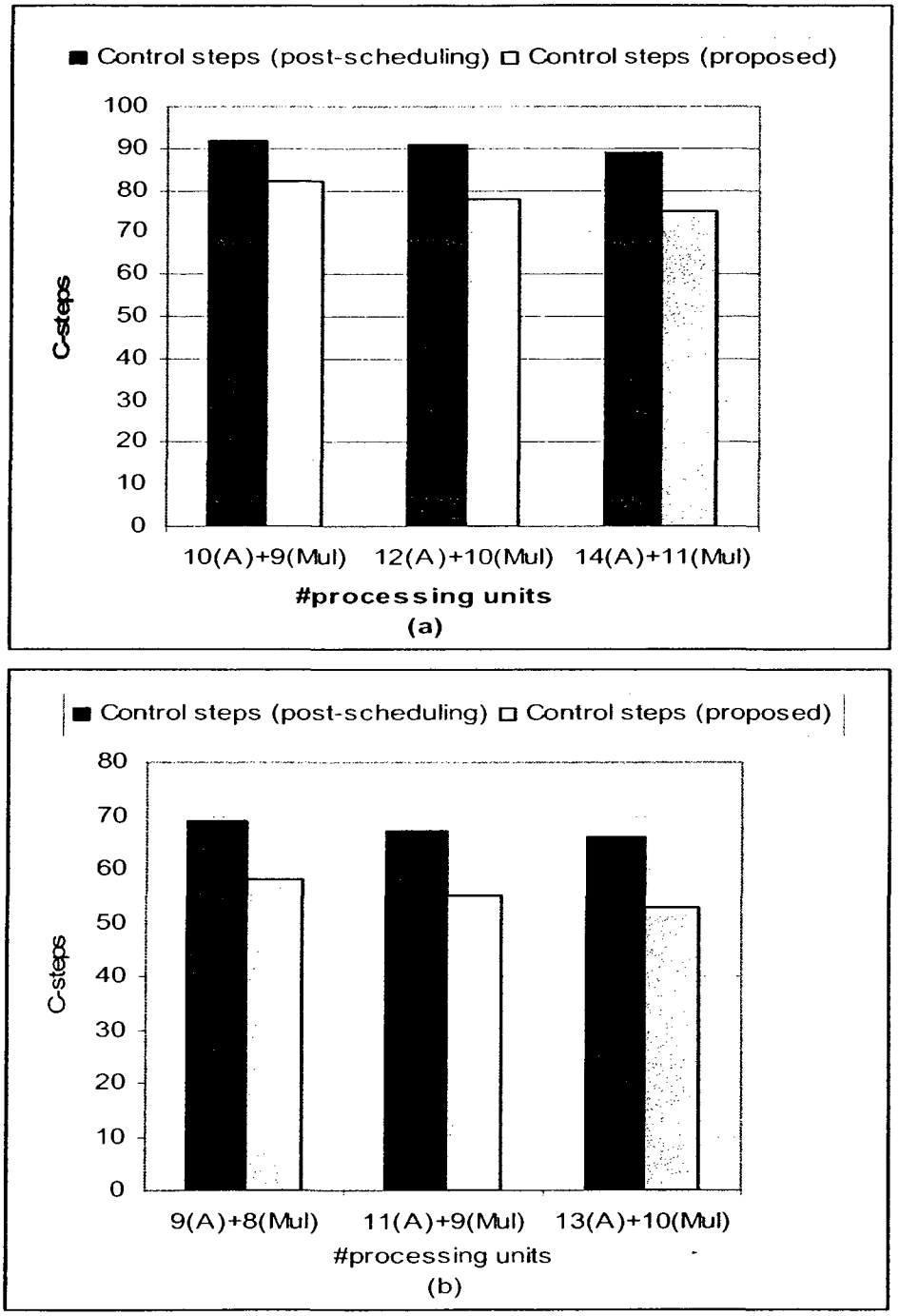


Figure 2. 12: The number of control steps obtained by using the proposed and post-scheduling schemes for different numbers of processing units when applied to (a) DCT-feig benchmark (b) DCT-chem benchmark.

2.9 Summary

The tasks of high-level synthesis, namely, the scheduling and resource allocation should use a realistic model of the parallel processing architecture. When a high-level synthesis does not consider a realistic model of the target parallel processing architecture, the resulting schedule may lead to an inefficient implementation. A realistic model should support inter-processor communication delays and structural pipelining of functional units.

A technique for the synthesis of DSP cyclic data flow graphs onto heterogeneous distributed-register based multiprocessing architectures employing a graph theoretic approach has been devised. The main focus has been on developing a new high-level synthesis framework by considering a realistic model for the multiprocessor architecture with a distributed-register configuration. The proposed technique starts by modifying the original DFG representing a DSP algorithm by inserting communication (dummy) nodes to represent the ICDs between the nodes of different types. The modified DFG is then used to build iteratively a time schedule based on the mobility of each node. An algorithm has been proposed to identify each critical or near-critical loop in the modified DFG. Next, by employing the initial time schedule and by using the loop identification algorithm, the task of an initial processor allocation is carried out. Since, the initial time schedule and processor allocation does not take into account the ICDs of the nodes of the same type, the initial time and processor schedules may not be valid. Hence, the initial time and processor schedule have been modified to take into account the ICDs between a pair of nodes of the same type assigned to two different processing units in order to find the final time and processor schedule. This modification has been carried out by inserting

additional cycles into the time schedule in order to ensure on the validity of the ICDs between a pair of nodes of the same type. In order to assess the proposed synthesis technique, it has been applied to the synthesis of different DSP digital filters and has been compared with various other commonly used synthesis techniques. Reasonable computation times are obtained for all of the benchmark problems considered, i.e., less than two seconds in case of the DCT-feig (547 nodes). It has been shown that the proposed synthesis technique outperforms these techniques in terms of the iteration period and the numbers of processing units of the synthesized architecture.

Chapter 3

Simultaneous Scheduling, Allocation and Placement taking into Consideration Inter-processor Communication Delay

3.1 Introduction

In Chapter 2, due to the importance of considering the effect of physical design on high-level synthesis, a technique for the high level synthesis with the objective of minimizing interconnect delay of data communication between processing units has been developed. In the proposed technique for scheduling and processor allocation, the interprocessor communication delay has been assumed to be taken from feedback placement information or from an estimated value of the interprocessor communication delay. In this chapter, a technique in which the placement process is integrated into the high level synthesis in order to determine the physical position of the processing units in the placement space during the building of the time schedule and processor allocation, which

provides with a more accurate information about the interconnect delay between the functional units, is presented. Furthermore, the technique of chapter 2 and most of the other techniques for high level synthesis uses only operation-specific functional units, i.e., adders or multipliers, in the allocation process. In this chapter [44-46], the proposed technique provides the designer with a greater flexibility to explore the design space by using a hybrid arithmetic functional unit library composed of both fixed operation-specific units and reconfigurable functional units capable of executing multiple operations. Moreover, by using these reconfigurable units, the data transfers can be more localized so that interprocessor communication delays are reduced.

A technique for simultaneous scheduling and allocation and placement using hybrid library of functional units composed of both operation-specific and reconfigurable multiple-operation functional units, is proposed. In order to build the time schedule and processor allocation simultaneously with placement process, the information about the positions of the functional units that already placed in placement space and about the candidate positions for placing a new functional unit must be available or predictable. Hence, a systematic process must be employed for the placement. The concept of triangular mesh is commonly used [48] to partition the interior region occupied by number of objects into nicely shaped triangles by adding vertices in the center of the objects and connecting them by edges. In our scheme, triangular meshes can be also employed to connect the centers of functional units in the placement space. In this regard, in order to find the suitable positions at where to place, one by one, the functional units, a technique is needed to, iteratively, generate the triangular mesh. We use a Delaunay triangular mesh [48] in the proposed scheme since this method of triangulation makes

candidate positions well-distributed. Moreover, Delaunay triangulation maximizes the minimum angles of the mesh. Hence, adjacent edges connected with a narrow angle are avoided which in turn allows finding, quickly, the suitable gaps to place the remaining functional units in the placement space. We have no theoretical justification that Delaunay triangulation is the best method for our purpose. We would like to implement other triangulation methods for the purpose of rectangle packing and compare them with Delaunay triangulation.

The Chapter is organized as follows. A review for the related research for the high level synthesis and placement, and that for the reconfigurable computing is given in Section 3.2. The functional structures and characteristics of the dynamically reconfigurable functional units incorporated in our scheme are described in Section 3.3. The proposed scheme for the simultaneous scheduling, allocation, and placement is presented is Section 3.4. In Section 3.5, the proposed scheme is applied to the some well-know benchmark problems. Section 3.6 summarizes the work presented in this chapter and highlights some of the salient features of the proposed scheme.

3.2 Related Research

3.2.1 Related research in high level synthesis and placement

Different approaches can be found in the literature addressing both the high level synthesis and placement. In [49] operation binding, placement, and scheduling are performed sequentially. The placement was driven by the clock slack time information obtained from the binding results. The work of [50] proposed a layout estimation technique for binding, and used it to select the most effective binding. Further, the

technique of [51] formulated the simultaneous binding and placement problem into a Mixed Integer Linear Programming (MILP) model. However, the applications of both in [50] and [51] are confined to only a 1D placed target architecture. In [52], an estimation of the layout cost for high level synthesis using a simulated annealing based floorplanner has been proposed. In [53], a technique for the integration of resource sharing and placement into an efficient linear programming formulation has been proposed. It is well-know that simulating annealing used in [52] and ILP used in [53] are not practical for intensive application due to their high time complexity.

3.2.2 Related research in reconfigurable architectures

It has been shown in [54] that reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfiguration at the various levels of the computational hierarchy gives many trade-offs in terms of flexibility, reconfiguration time, performance, area, and power/energy consumption. A fine-grained reconfigurable device (gate level) is extremely flexible; it can implement any application. However, the flexibility comes at a cost. The routing architecture must allow a connection from any part of the chip to any other part of the chip. Switch boxes are used to enable this sort of flexibility. The switchboxes are composed of many transistors to enable a flexible routing. Compared to a direct connection, it is apparent that switch boxes add much overhead to the area, communication delay (performance), and power consumption.

A significant number of reconfigurable architectures have already been proposed, varying mostly on the granularity's degree. An overview of the most popular

reconfigurable architectures can be found in [55]. Fine-grained architectures [56], [57], [58], such as classical FPGAs, suffer from high reconfiguration delays and power consumption. Coarse-grained architectures [59], [60] eliminate the disadvantages of fine-grained ones and preserve universality and flexibility at most cases, but operates only on word-length data formats. Recently, hybrid architectures [61], [62] have been proposed which try to combine the benefits of the two above approaches. All these solutions propose new architectures to enable dynamic hardware reconfiguration. The Morphosys reconfigurable system is a complete reconfigurable SoC implemented at the layout level [63]. It incorporates a 32-bit RISC processor and a 8x8 array of coarse-grained reconfigurable cells for efficient mapping of DSP applications. The basic reconfigurable cell is universal consisting of an ALU and a MAC unit. The SoC also incorporates a DMA-controller and a Frame buffer for fast data transfers between the memory and the reconfigurable array module. In [62], a coarse grain reconfigurable architecture is proposed which targets DSP applications, enabling efficient template-based operation chaining. Every node of the applications' DFG is mapped on a computational resource. The templates are implemented by interconnecting appropriately a number of computational cells. They perform template chaining by using a flexible inter-template interconnection network. Although, the proposed architecture seems to have performance gains in comparison with the straightforward template-based methods, the area overheads imposed by the basic template cell architecture are not negligible. Moreover, the inter-template communication delay has been not taken care.

To overcome the penalties of FPGA, small-scale reconfiguration would minimize the area and delay penalties by inserting into fixed-logic only the minimum amount of

reconfigurable logic and interconnect and by reusing part of the available logic to achieve the desired component flexibility. Therefore, arithmetic components designed with this technique have the flexibility to perform multiple operations but are ASIC-like in their efficiency. In high level synthesis, reconfiguration can be applied in the construction of the RTL architecture considering that each RTL component is not active in every control step. Partially inactive components can be merged into a reconfigurable component.

The concept of run-time (dynamically) reconfiguration is well known and can be applied on different phases of the design process, according to the granularity of the reconfigurable blocks, which may be complex functions, simple RTL components or LUTs. Dynamically reconfigurable components is presented in [64] by developing a morphable multiplier, which is an array multiplier that can be configured through multiplexers to work as either an adder or a multiplier. In [65] morphable multipliers are used for the design of a graphics processor. In [66], an implementation of a simulating annealing algorithm was presented to solve the scheduling, allocation and binding problems, assuming that the target architecture uses run-time reconfigurable functional units. Unfortunately, the inter-processor communication delay has been neglected. Commercial microprocessors have used reconfigurable functional units to support SIMD instructions [67]. These reconfigurable functional units support a single type of operation, like addition, and vary only in the number and width of the operations.

3.3 Dynamically Reconfigurable Functional units

Existing approaches for high level synthesis consider functional units as blocks of hardware that implement one or more operations, but where the setup of the desired function has no cost. In other words, all the operators exist, in the functional unit, and the

selection of the chosen function does not imply any time delay. Unfortunately, this assumption is not valid since using such functional units needs a certain amount of time to reconfigure its logic. When one operation is assigned to a functional unit that was last, used for a different function, it is necessary to spend a certain number of clock cycles in reconfiguration, prior to the execution of the operation.

One way to provide reconfigurable functional unit resources is to specify a concise set of operations desired in a functional unit, and to design such a multi-mode functional unit for very high speed. When a reconfigurable functional units is designed, the similarities between the desired operations can be implemented in fixed logic, and reconfigurable logic and interconnect must be used to implement the differences. Therefore, the first step in designing a reconfigurable unit is to determine the common functions between the operations to be implemented, hence minimizing the part for the reconfigurable hardware (and its associated penalties). For example, adders and multipliers have similar hardware substructure, making them more suitable to be implemented as a reconfigurable unit, resulting in an efficient flexible implementation. Other arithmetic operation combinations may also be considered for reconfigurable implementation. Other forms of this reconfigurability could to integrated a wide bit width operation with multiple operations of narrower width; several low-precision operations could be embedded within a high-precision operation; a rarely used operation could be also integrated within a high use operation. The direct and simple way to do this is to construct an individual implementation of each operational mode in the functional unit, and to use a multiplexor to select the output based on the mode. The delay overhead for this type of reconfiguration consists of the multiplexor and latch delay, as well as the

interconnect delay to move primary inputs and outputs to and from the different operators within the reconfigurable functional unit. In order for a reconfigurable functional unit to provide area savings, its area should be smaller than the combined area of all of the operations implemented individually. Partitioning operations across time instead of space (each operation can be implemented in the same physical space) and then the required component configuration is selected at the necessary time. Such structures of reconfigurable functional units avoid the large performance, area, and power penalties associated with FPGAs and DSP processors while at the same time clearly provides hardware flexibility.

The morphable multiplier proposed in [64] is employed in the technique proposed in this chapter since it is capable of implementing both multiplication and addition (in fact, it can perform two or more data-independent additions in parallel) with the same delay as a fixed logic multiplier and with very small area overhead. Given the regularity occurrence of MAC operations in DSP algorithms, such dynamically reconfigurable functional unit provides significant benefits for them. The goal of the proposed technique is to make use of this morphable multiplier in a hybrid library of functional units composed both operation-specific and reconfigurable functional units supporting sets of different operators. In this chapter, the focus is on the morphing between a set of multiplication and addition operations in the high level synthesis. The objective is to utilize such morphable multiplier to maximize the data transfer.

The morphable functional unit has been designed based on a tree multiplier. In a tree multiplier, the partial products can be generated using an array of AND gates, or more generally, radix-k Booth's multiple generators. The partial product reduction tree

(PPRT) adds the partial products and produces a sum result in a redundant form. The redundant form is converted into a binary form by a carry propagate adder. The PPRT design technique that is used in [64] is based on an approach in which a globally optimal way of interconnecting low-level compressor stages is identified. This method exploits the fact that the inputs and outputs of a compressor do not equally contribute to the delay of the multiplier. The critical path through a tree multiplier will almost certainly pass through compressors in the PPRT. In fact, not all the compressors in the PPRT are on the critical path. Compressors that are not on the critical path have timing slack. The timing slack for each non-critical compressor is equal to the minimum delay that can be added to it in order to make that compressor critical. Utilizing a compressor in more than one operation (e.g. add and multiply) requires some number of multiplexors to modify the connections of this compressor. A compressor with sufficient slack to allow the incorporation of multiplexors on its inputs is called a reusable compressor. Fig. 3.1 gives an example for a Morphable $6-bit \times 6-bit$ Multiplier taken from [64] using re-used adder cells. Gray wires show the PPRT in multiplier mode. In this figure the black adders are re-used in a 7-bit ripple carry adder since they have a slack time. In fact, a 32 bit array multiplier can be made to work as a multiplier in mode 1 and 8 32 bit adders in mode 2 with only 20% extra area. For fewer adders, the morphable multiplier uses a very little area overhead and has the same delay with a single mode multiplier component. For our application to HLS, we have chosen a morphable array multiplier that works as a multiplier in mode 1 and 2 adders in mode 2. If during the time scheduling the morphable multiplier has been reconfigured for two different type of nodes, then the two configuration must be separated by one cycle delay to allow the reconfiguration between

the two nodes. The incorporation of this morphable multiplier is given in the following section. There are some important related issues which have to be addressed in the proposed technique, for instance, how to allocate operational nodes to these reconfigurable multipliers, the area and delay trade off, the reconfiguration times, the way used for the placement of such processing units.

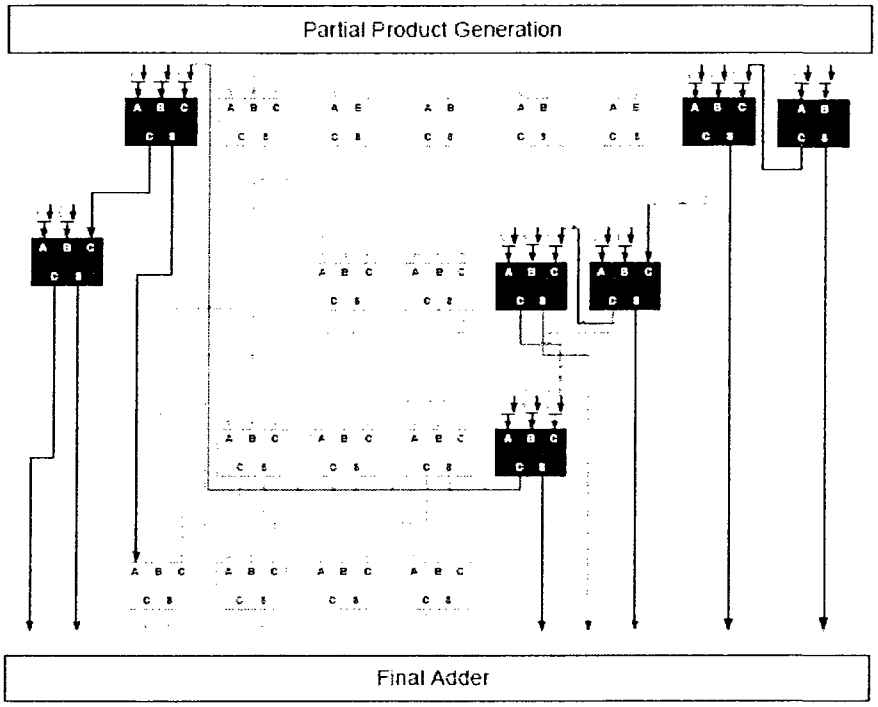


Figure 3. 1: A Morphable 6× 6 Multiplier [64] using re-used adder cells. Gray wires show the PPRT in multiplier mode. The black adders are re-used in a 7-bit ripple carry adder.

3.4 Simultaneous Scheduling, Allocation and Placement

This section presents the proposed technique for simultaneous scheduling, allocation, and placement. The problem is to find a rectangular space of minimum size into which all functional units are placed while taking into consideration the interprocessor communication delay (the distance between the functional units) to satisfy time constraints. Initially assumption is that all the operations in the DFG can be allocated to

distinct functional units. However, during the procedure of the proposed technique, two functional units with the same operation type are allowed to be combined into one if their assigned operations are not executed concurrently at the same control step. To support the functional unit sharing between operational nodes, we should allow the possibility of overlapping between functional units of the same operation type in placement space. The overlapping between units of different time is allowed in some cases in which hybrid library of fixed-specific operation functional unit and reconfigurable functional units is supported while taking into consideration not only the ICD but also the reconfiguration time.

The formulation of the problem must satisfy the following two constraints leads to a feasible scheduling, allocation, and placement solution:

1. Disjoint constraint: Two functional units should not overlap if they are of different types. However, they do if there are of the same type or a hybrid library of functional units is available.

2. The precedence relations between operational nodes are not violated: a valid way to schedule the operation while taking into consideration the interprocessor communication delay and the reconfiguration time of the reconfigurable functional units. In other words, the delay of interconnect between functional modules should not cause any node mobility violation.

An iterative procedure based on the node's mobility is employed. The earliest and the latest firing times (EFT and LFT) at which each node can be scheduled to fire are iteratively calculated.. These earliest and latest firing times are found relative to a reference node taking into consideration the interprocessor communication delay (i.e., the

positions of the functional units). In other words, the actual position of the functional units in the placement space is taken into consideration in these calculations.

Let us assume that each node in a given DFG can be executed by a functional unit of a rectangular shape. The proposed packing technique positions all of them onto an xy-plane while it tries to minimize the layout area. Let us also assume that all edges of the rectangles are parallel to the x-axis or y-axis. Under these assumptions, the algorithm schedule the nodes one by one by allocating them to a specific functional unit while specifying its position in the placement space so that the precedence relation between the nodes are not violated. In each iteration of the process, one node is chosen according to its urgency to be scheduled, which specified while taking into consideration, precedent relation, the previously place units and the candidate position for new placements, and then it is allocated to a functional unit that already placed. If there is no functional unit available to allocate the node while the precedence relation are not violated, a new functional is placed in the placement area so that ICD is taken into consideration and precedence relation are satisfied. If there is no candidate position in the placement area satisfies the ICD and precedence relations, this blocking situation is solved by inserting new cycles (control steps) into the time schedule so that treat the violation in the precedence relations is treated. Recall that the technique places the rectangles representing the functional units at candidate positions so that the area of the placement space and the iteration period are minimized.

In order to reduce the time overhead due to the proposed simultaneous approach, the proposed algorithm favours accelerating the rectangle packing process rather than perfectly minimizing the layout space. Therefore optimization schemes are not applied to

find the configuration of the rectangles (processing units) in the layout, but a heuristic is used to quickly find gaps and place the remaining rectangles representing the functional units in the gaps. Moreover, the number of candidate position available to place a new processing unit is restricted to speed up the proposed technique.

The heuristic uses a Delaunay triangular mesh [48] to connect the centers of the placed rectangles, as shown in Fig. 3.2. Let us denote a mesh as $M(U,E,T)$ consisting of vertices $U = \{u_1,\ldots,u_{n+4}\}$, edges $E = \{e_1,\ldots,e_l\}$, and triangles $Tr = \{tr_1,\ldots,tr_m\}$. The algorithm picks up first reference node from the given DFG and allocates it to a functional unit of the same type and then positions the rectangle r_1 representing the functional unit at the center of the layout area and generates a rectangular space that entirely encloses the positioned rectangle. Let the space be S, its four corner vertices be u_1 to u_4, and the center of r_i be u_{i+4}. We initially define the size of S as twice the size of r_1. The algorithm then generates four triangles tr_1 to tr_4, which connect the five vertices u_1 to u_5, as shown in Fig. 3.2. The next node to schedule is then selected according to some rule and a corresponding functional unit is then allocated to the functional unit placed previously in the layout area if it is capable of executing the current node to be schedule or other wise it is allocated to a new functional unit that need to be allocated in the layout are while taking the ICD into account. The ICD is assumed to proportional to the distance between the centers of the functional units or the candidate position in the triangular mesh. After placing each new rectangle r_i one by one, the algorithm updates M as shown in Fig. 3.2, by connecting the new vertex u_{i+4} to several vertices and modifying several triangles as done in [48]. While deciding on a position in

which to place a functional unit, the algorithm calculates candidate positions on E and evaluates the candidate positions. Next, we describe the order of visiting elements in E and then describe the scheduling and the allocation and then the evaluation of the candidate positions so that interconnect distance and proportional delay is taken into consideration while the area is minimized. Finally, we describe the modification of the triangular mesh.

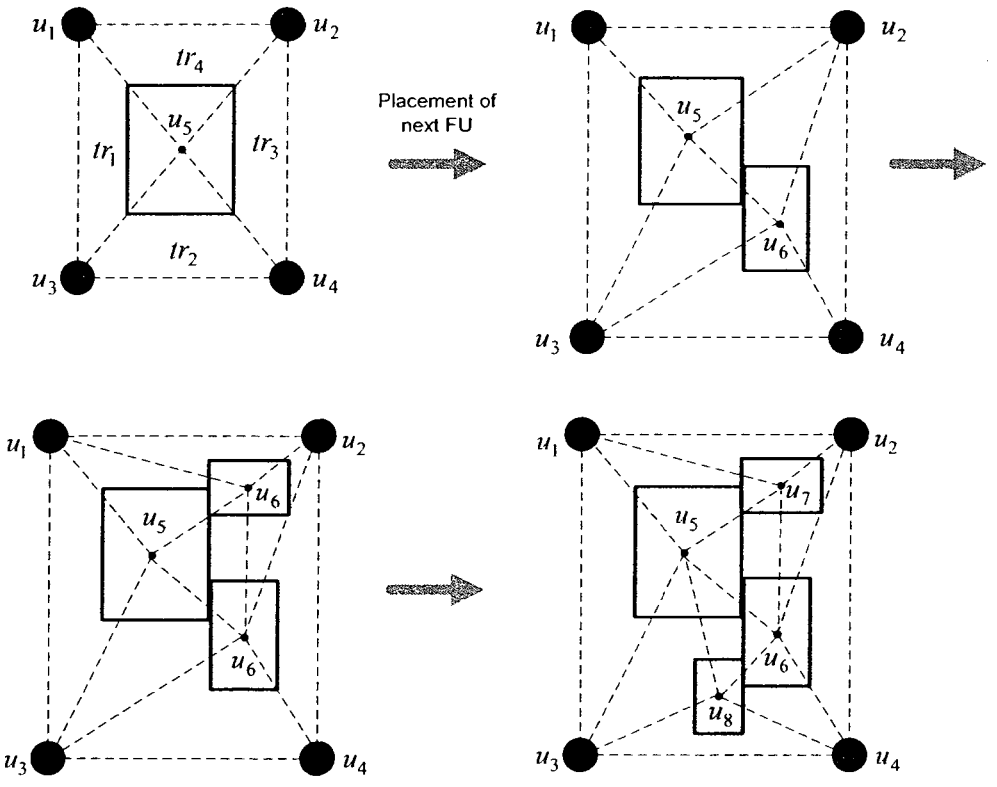


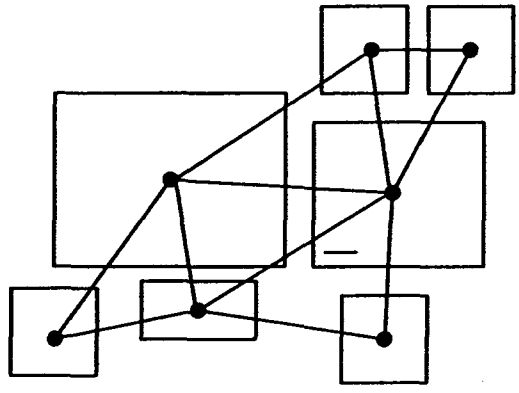
Figure 3. 2: Processing flow of functional unit placement and update of mesh M.

3.4.1 Order of referring to the mesh edges

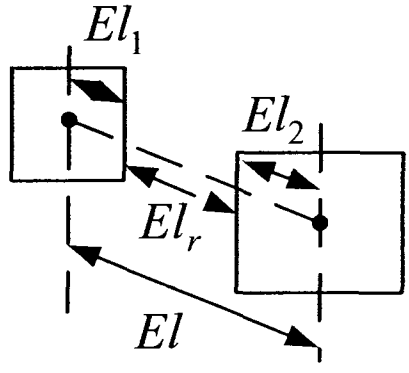
The next operational node to be scheduled is chosen from the list of remaining operations in away in which the node with the minimum mobility is chosen or at least one direct predecessor or successor operation has been scheduled. This requirement is crucial to improve the quality of the schedules that are found by the heuristic. The algorithm places

the rectangles one by one and searches for a position to place the rectangle, which satisfies the following conditions as much as possible: Condition 1: No overlap between r_k and any previously placed rectangles. Condition 2: Minimum extension of the layout area S and keep the iteration period close to its preferred minimum value i.e., close to the iteration period bound. To quickly search for positions where rectangles can be placed satisfying the above conditions, the algorithm picks suitable positions by using the following two strategies: Strategy 1: It favours selecting sparsely populated regions since it is easier to place rectangles in such places without overlapping with other rectangles. Strategy 2: It favours selecting interior positions since it is easier to place rectangles in such locations without enlarging the layout space.

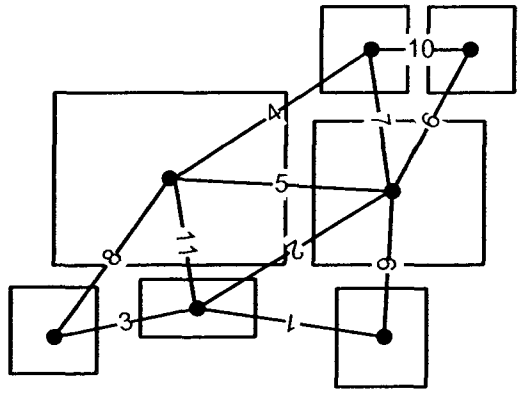
Fig. 3.3(a) shows an example of a triangular mesh and rectangles. Here, let El be the length of a mesh edge, El_1 is the length of the part of the edge that is inside the rectangle whose center places it at an end of the edge, and El_2 is the length of the other part of the edge that is inside another rectangle, as shown in Fig. 3.3(b). Our technique calculates the values of $El_r = El - (El_1 + El_2)$, the length of the remaining part of the edge lying outside the two rectangles. Here, the technique lets El_1 or El_2 take the value of zero when the ends of the edge are on $u_i, i = 1...4$. This is because it is more likely that gaps will be found around mesh edges whose El_r values are larger. The numbers in Fig. 3.3(c) denote that the edges are ordered from the largest El_r to the smallest. This is obvious since the larger the El_r the larger the interconnect delay between the two functional unit. The algorithm searches for gaps on the edges in this order so that it satisfies Strategy 1.



(a)



(b)



(c)

Figure 3. 3: (a) Triangular mesh connecting centers of previously placed rectangles. (b) Calculation of values of El_r. (c) Order of the El_r values.

A Delaunay triangular mesh is used in the proposed heuristics since Delaunay triangulation makes candidate positions well-distributed. Since the definition of Delaunay triangulation is the triangulation that maximizes the minimum angle of M, it avoids making closer candidate positions by adjacent edges connected with a narrow angle. At the same time, the algorithm counts ce_j, the number of corner vertices u_1 to u_4 touching the edge e_j. Fig. 3.4(a) shows an example of the distribution of ce_j . This figure shows that interior mesh edges have the smaller ce_j values. The algorithm then groups the mesh edges according to their ce_j values. The algorithm starts the trial placement of functional units on the edges. It first extracts edges from the $ce_j = 0$ group, then the $ce_j = 1$ group, and, finally, the $ce_j = 2$ group, so that it satisfies Strategy 2.

The algorithm extracts edges in each group in the sorted order, starting from the edge that has the largest El_r value. The algorithm calculates at most three candidate positions where r_k touches the functionals previously placed at the ends of e_j, as shown by the two dotted rectangles in Fig. 3.4(b), and tries to place r_k at each of these positions. For edges $ce_j = 1$, the technique tries to place r_k at a position at which it touches the functional unit previously placed located at one of the ends of e_j or in the center of it as shown in Fig. 3.4(c). In the case of $ce_j = 2$ edges, the technique tries to place r_k at the center of e_j as shown in Fig. 3.4(d) because there are no rectangles at the ends of $ce_j = 2$ edges.

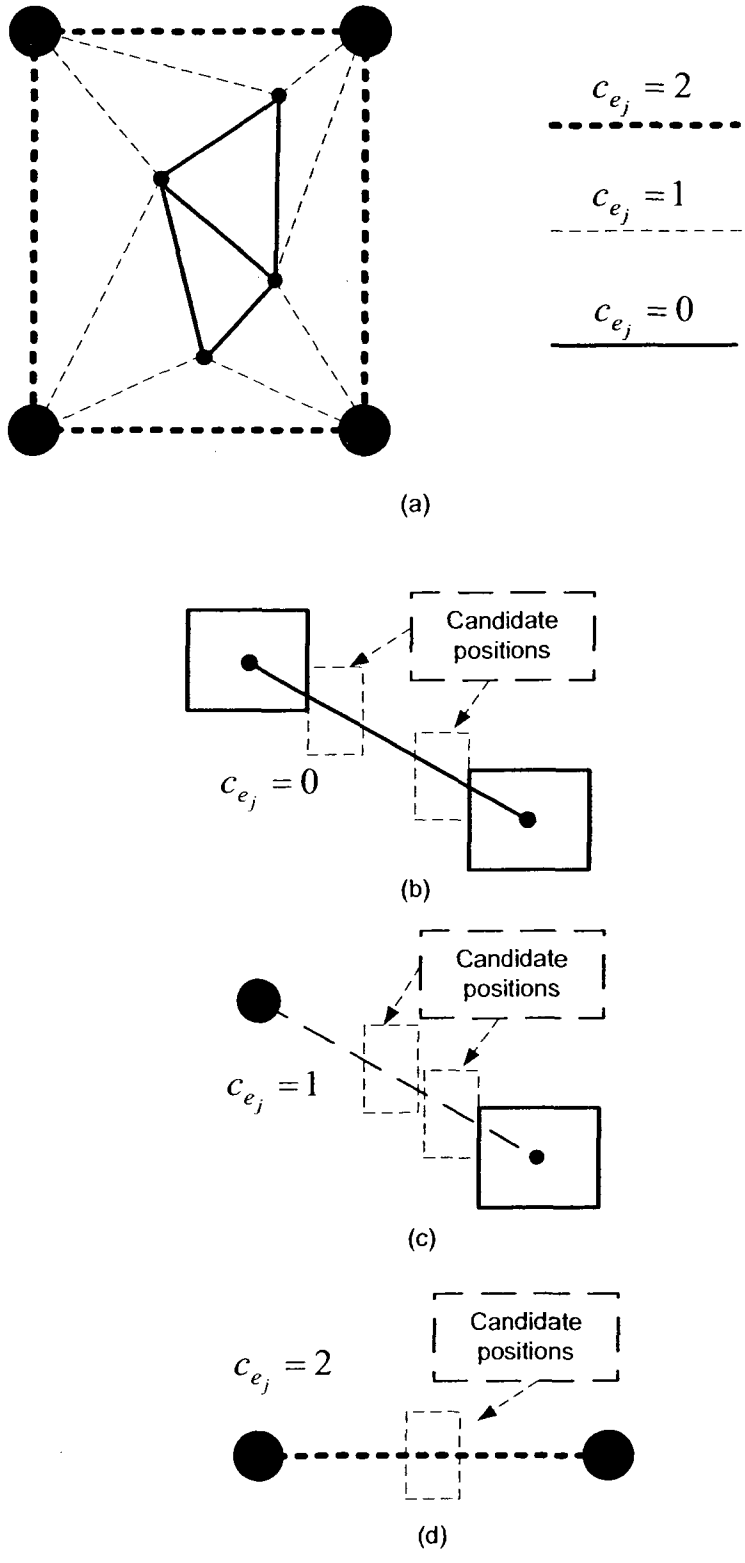


Figure 3. 4: (a) Values of ce_j for edges. (b), (c), (d) positions to try to place the current rectangle.

3.4.2 Scheme for scheduling and allocation of the nodes

The schedule is built by selecting a reference-node and by calculating the mobility of all non-scheduled nodes with respect to this reference node. All the non-scheduled nodes are put in a list. The node with the minimum mobility calculated thus far is chosen for scheduling first and then removed from the list. When choosing between equal mobility nodes, the selection is made such that individual operation concurrency is equalized with the previous step. This is to reduce the chance that a reconfigurable unit would need reconfiguration when used in that particular step. Due to the new firing time of the node, the time schedule of other non-scheduled nodes may be affected. This node is chosen to be the new reference-node and the rest of all the earliest and latest firing times for the rest of the non-scheduled nodes are calculated. The calculation of the earliest and latest firing times must include the position and distances between the functional units in the placement space. A new node is chosen for scheduling and the process is iteratively repeated until all the nodes are scheduled.

A. Valid Ranges and Extra Cycle Insertion

A valid schedule range specifies a valid way to schedule the operation for a given precedence relations, inter-processor communication and the current allocation and placement positions of the hardware.

Given a data flow graph of a DSP application, the time schedule can be built using the longest path matrix Q^f defined in Chapter 2. The scheduling heuristic calculates the valid range (mobility) of start times start times for an operational node while taking into consideration the position of the candidate functional unit in the layout space. The candidate positions to allocate and place the node are of two types: (i) the

center of the functional units that already placed or (ii) the possible points on the mesh edges to place a new functional unit. Since there is more than one candidate position and functional unit, the length of the communication delays depends on the position to which the operation is assigned. Therefore, more than one range is calculated for each node, namely, one for each candidate position in the layout space. The earliest firing time and the latest firing time for a node v_j relative to that of a reference node v_i are, respectively, given by

$$EFT\left(\left(v_j/v_i\right), v_j.cnps, v_i.crps, T_{cr}\right) = FT(v_i) + Q_{ij}^f + comm(v_i.crps., v_j.cnps) \quad (3.1)$$

$$LFT\left(\left(v_j/v_i\right), v_j.cnps, v_i.crps, T_{cr}\right) = FT(v_i) - Q_{ji}^f - comm(v_j.cnps, v_i.crps) \quad (3.2)$$

where $FT(v_i)$ is the firing time of node v_i, $cnps$ is the candidate position in placement space, $crps$ is the current position in placement space, and $comm(v_i.crps., v_j.cnps)$ is the interconnect delay between the candidate position to allocate and place node v_j and the current position to which node v_i is previously placed. The model used to calculate $comm(v_i.crps., v_j.cnps)$ will be discussed later in this section (Section 3.4.2.D). To find the earliest and the latest firing times of node v_i, the maximum earliest firing time and the minimum latest firing time of the node must be found relative to all previously scheduled nodes and taking into consideration all previously place functional units. Thus, EFT and LFT of node v_j are, respectively, given by

$$EFT(v_j) = \max_{all\, i<j} \left(EFT\left(\left(v_j/v_i\right), v_j.cnps, v_i.crps, T_{cr}\right)\right) \quad (3.3)$$

$$LFT(v_j) = \min_{all\, i<j}\left(LFT\left(\left(v_j/v_i\right), v_j.cnps, v_i.crps, T_{cr}\right)\right) \quad (3.4)$$

Thus, the mobility or the scheduling range in the schedule of any node v_j is given by.

$$M(v_j) = LFT(v_j) - EFT(v_j) \quad (3.5)$$

The ranges from the earliest firing time to the latest firing time are constantly obeyed when operations are scheduled; it is possible that when a new operation is scheduled to fire at a specific functional unit or to a specific candidate position , the earliest firing time is larger than the latest firing time. In other words, the scheduling range can be empty (the mobility $M(v_j)$ is negative). This can happen because the communication delays are not included in the final distance matrix Q^f. They can not be included because it would require that all operations are allocated and placed a position in the rectangular placement area beforehand. Hence, during the scheduling heuristic used, every time it schedules a node, it can happen that given a partial schedule, a still unscheduled operation can not be scheduled. The first reason is that the operation distance matrix does not include communication delays. It is then possible that given a partial schedule, the inter-processor communication delays for a node can not be satisfied, which is the case when all the ranges calculated for given node on all the available functional units are empty. A second reason is that the scheduling method tries to allocate the node in question to the available functional units. It can happen that given a partial schedule, a node can not be scheduled because the functional unit it needs is not available. The treatment is to insert cycles into the time schedule to solve the problem which is similar to the one presented in Section 2.6. Every time a new node is schedule,

cycles may be required to be inserted to the time schedule. These inserted cycles are in general due to the following two sources (a) an empty scheduling range (b) a time required for the reconfiguration of a reconfigurable functional unit.

B. Inserting the New Cycles

Inserting an extra cycle in the schedule creates for every resource a new free time unit in the schedule. Therefore it can be used when there was not a resource available for the operation. Furthermore, inserting extra cycles in the schedule can increase the time gap between two nodes. So, inserting extra cycles in the schedule can also be used when communication delays are not yet satisfied. Notice that a side effect is an increase of the iteration period T. When new cycles are inserted in the schedule, the number of extra cycles and the column in the time schedule table or matrix where to insert these cycles have to be specified. The cycles are inserted immediately before the preferred firing time. Because the scheduling is non-preemptive, after a cycle is inserted in the schedule every operation should still execute uninterruptedly and be allocated to a continuous series of cycles in the schedule. We proposed to move operations such that every node still starts in the same control step in the time schedule as it did before the cycles were inserted. In fact, we use similar strategy to the one proposed in Chapter 2 to determine the number of cycles to be inserted and where in the time schedule they must be inserted.

The process of cycle insertion also determines how the node can be scheduled so that:

(i) The operation fits in the time schedule.

(ii) The precedence relations or minimum interprocessor communication delays for other nodes still not violated.

The process of cycle insertion returns an offset *shift* that indicates how many cycles the node is shifted to the right with respect to the targeted firing time that was required. It also returns the number of cycles N that has to be inserted in the time schedule in front of the column corresponding to the preferred firing time. Note that the iteration period is increased by N. To illustrate how the process works, 3 examples are shown in Fig. 3.5.

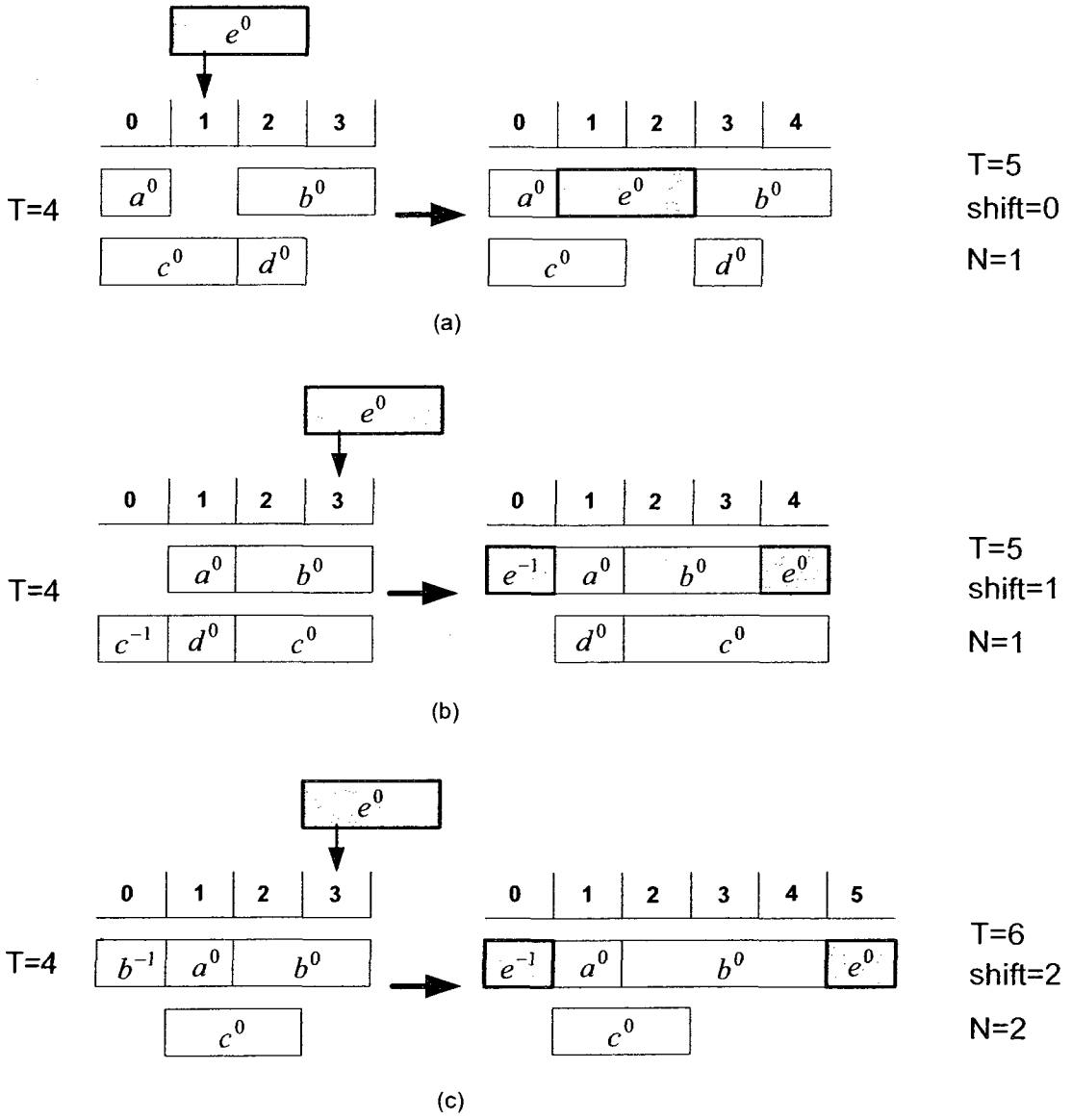


Figure 3. 5: Three examples of how a node can be inserted in the time schedule.

C. Choosing the firing time within the valid scheduling range

The level of a control step is defined to be the number of nodes which will eventually occupy this control step. This level determines the number of functional units required during this control step in the time schedule. The chosen node is scheduled to fire at a control step that would results in a minimum number of functional units required given that precedence relations are satisfied. In presence of the hybrid functional units, the level of a control step being the summation of sub-levels of the different types of operations and given by $level = level_{type1} + level_{type2} + \ldots$. The choosing of the best firing time is done by examining all the control steps within its mobility such that total number of operations per cycle is minimized, rather than individual operator concurrency. More specifically, find the control step having the minimum total level as a primary key or the minimum sublevel ($level_{type}$) as a secondary key. This is very significant to reduce the number of functional units needed and to increase the utilization of the dynamically reconfigurable functional units, thus the total area is reduced and local data transfer are maximized compared to an architecture that uses only operation-specific functional units.

The following points summarize the above discussion regarding each possible scheduling, allocation, and placement of a non-scheduled node to the placement space:

- The node is assigned to fire at previously placed functional unit or to new functional unit placed at the position in placement area results in the largest valid mobility (scheduling range).
- If there is no valid mobility for each possible candidate potion, the node is scheduled to fire at the candidate position that leads to the minimum number of inserted cycles.

- In the proposed algorithm, a new functional unit is added in the placement space only if this will lead to less number of inserted cycles.
- If adding a new unit will result in the same number of inserted cycles equal to that if the node is assigned to current functional units, then the node in question is assigned to one of current functional units so that the placement area is not increased.
- Every time a node of type *(a)* is assigned to a reconfigurable unit running in a mode of type *(m)*, a control step must be blocked on the corresponding functional unit in order to represent the reconfiguration time $RT_{a \to m}$ of the morphable multiplier.
- The position of the reconfiguration time slots in the time schedule should be tested and modified (moved) every time a new node is scheduled. Fig. 3.6 shows examples of such a slot movement.

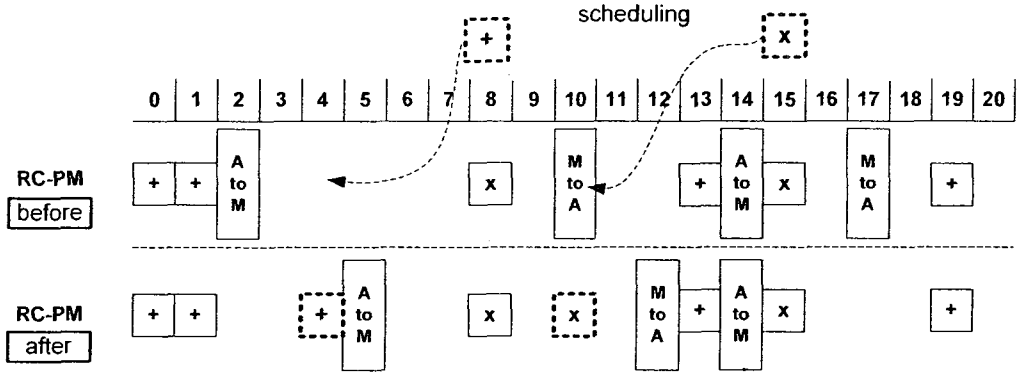


Figure 3. 6: Moving the reconfiguration time slots

D. Delay model for the wires

In fact, different delay models can be used to calculate the delay of the interconnect wires with respect to their length in the placement space, i.e., the distance between the two

functional units in question. One possible model is Elmore model that is commonly used is $d = 0.5RC$, where $C=cl$ $R=rl$, l being the length of wire, r being the resistance of wire per unit length, and c being the capacitance of wire per unit length. With the advances in the fabrication technologies more accurate delay models can be employed to consider for example the inductance of the wires.

3.4.3 Evaluation of candidate positions

Given a candidate position, the algorithm checks if the point satisfies the two conditions described in Section 3.4 and the valid mobility for the node. Starting from the $ce_j = 0$ edges, the algorithm refers to edges in the sorted order and calculates candidate positions on the edges. The algorithm then attempts to place r_k at the candidate positions. It checks overlaps between r_k and previously placed rectangles and calculates enlargement of S. If the algorithm finds that one of the candidate positions satisfies both conditions and mobility, it decides to place r_k there and selects the next rectangle. Otherwise, the algorithm selects the next edge to check to see if it satisfies both conditions. If no $ce_j = 0$ edge satisfies both conditions, the algorithm continues with the $ce_j = 1$ edges and, finally, the $ce_j = 2$ edges.

Even if the candidate position satisfies only Condition 1, the algorithm can place r_k after enlarging S. In this case, the algorithm evaluates the point. In fact, the points or the candidate position are evaluated if multiple positions satisfy the conditions and mobility. The evaluation function for the candidate positions uses a combination of the placement area and iteration period of the corresponding time schedule. It calculates the

value $\alpha A + \tau T$, where α and τ are user-defined positive values. A is calculated as follows:

$$A = \frac{A_{after}}{A_{before}}$$

where A_{after} is the area of S that would be if r_k is placed at the candidate position under evaluation and A_{before} is the area of S before the placement. A_{after} can be calculated after the enlargement of S described in the following section. T is calculated as follows:

$$T = \frac{T_{after}}{T_{before}}$$

where T_{after} is the iteration period of the time schedule after the placement of r_k and the possible insertion of cycles in the time schedule and T_{before} is that before the placement. Again, T can be calculated after the enlargement of S. We define the value of α and τ as $\alpha = \tau = 1$, if equal preference is given to the placement area and the iteration period. However, it depends on the requirements of designer: If the minimization of the layout spaces is important, α should be larger than τ. If $\alpha A + \tau T$ calculated on the candidate position is smaller than the smallest in the values of previously processed candidate positions, the algorithm saves the candidate position with this $\alpha A + \tau T$ value. The algorithm places r_k at the most recently saved candidate position because this was evaluated as the best position.

3.4.4 Local modification of the triangular mesh after the placement of a new unit

If it is decided to place the rectangle r_i at a candidate position that does not satisfy Condition 2, the algorithm enlarges S by moving some of the $u_i, i = 1...4$, as shown in

Fig. 3.7. Here, let the positions of $u_i, i = 1...4$ are (x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2). Also, we position the four corners of r_i at are (x_a, y_a), (x_b, y_a), (x_b, y_b), (x_a, y_b). The algorithm enlarges S by recalculating the position of $v_i, i = 1...4$ as follows:

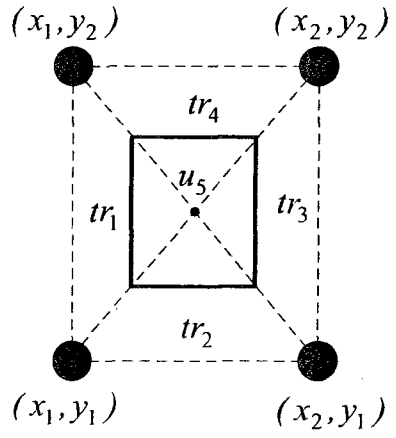
$$if\ x_a < x_a \text{ then } x_1 = x_a - ENGL$$

$$if\ y_a < y_1 \text{ then } y_1 = y_a - ENGL$$

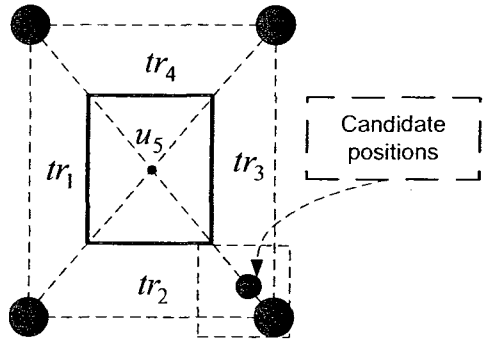
$$if\ x_b < x_2 \text{ then } x_2 = x_b + ENGL$$

$$if\ y_b < y_2 \text{ then } y_2 = y_b + ENGL$$

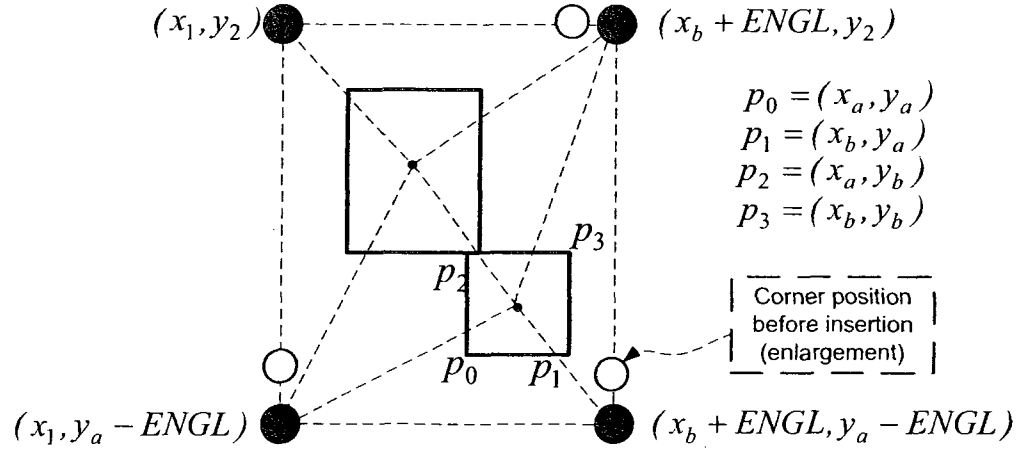
where ENGL is a constant positive value. Our implementation applies ENGL=0.1 S_w if $S_w > S_h$; otherwise, ENGL=0.1 S_h, where S_w and S_h are, respectively, the width of S and the height of S. After the algorithm places r_i by using the above steps, it updates the triangular mesh by adding the center of the placed rectangle u_{i+4} to the mesh. This process first connects u_{i+4} to the two other vertices of the triangles that share the edge e_j and divides each of the triangles into two new triangles. Fig 3.8 explains the mesh modification process in case of the reconfigurable functional units compared to that of operation-specific function unit. In Fig. 3.8(a), 5 candidate position are considered. Fig. 3.8(b) shows the enlargement of the space if a new functional unit is placed. Fig. 3.8(c) given the situation in which S has to be enlarged due to changing replacing a multiplier to a morphable one. The process then locally modifies the mesh, starting from the triangles that share the newly added edges. It selects an adjacent triangle to modify and swaps their shared edge to improve the triangles. The modification is recursively repeated between the modified triangles and their adjacent triangles until no triangles to be modified. The detailed algorithm of the mesh modification is described in [48].



(a)



(b)



(c)

Figure 3. 7: (a) A rectangle and triangular mesh. (b) One more rectangle is placed on a mesh edge. (c) Corners of S are moved when the placement of the current rectangle requires enlarging the layout region S.

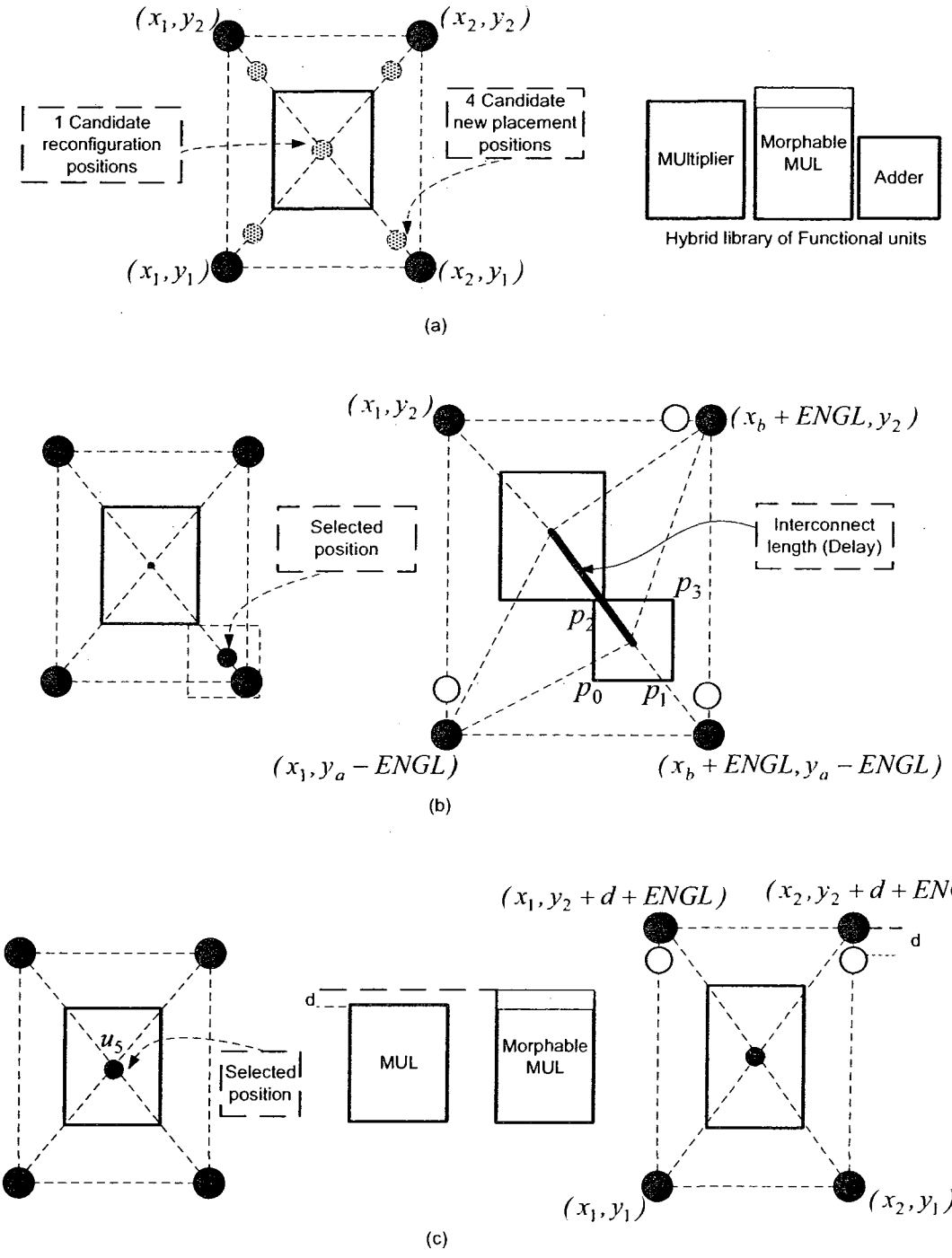


Figure 3. 8: (a) A mesh with five candidate postions. (b) One more rectangle is placed on a mesh edge. (c) Enlarging of the layout space S due to placing of a morphable multiplier.

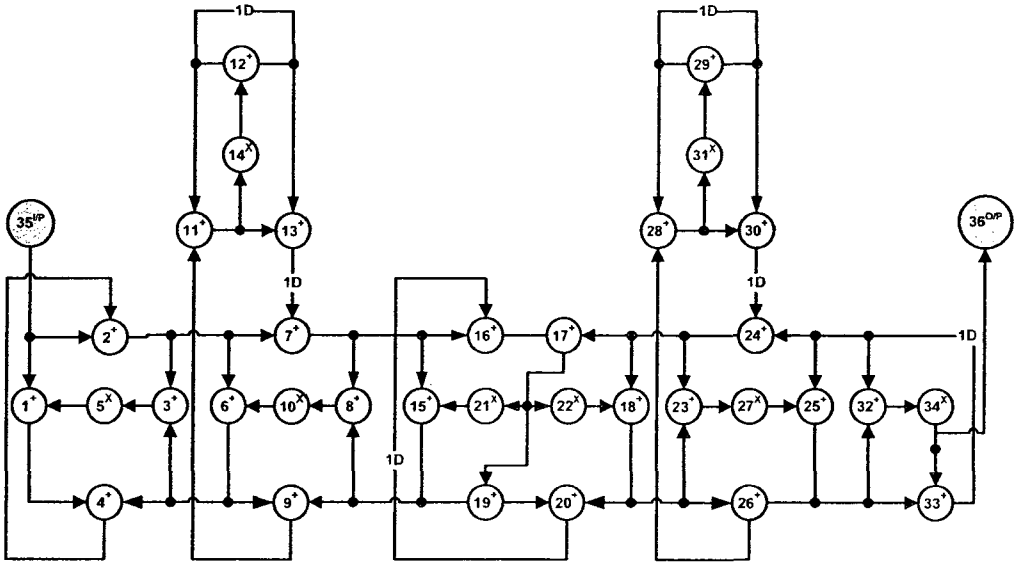


Figure 3. 9: The DFG of fifth-order elliptic wave filter

3.5 Experimental Results

In this section, some well-known benchmark examples of synthesizing intensive DSP applications using the technique presented in this chapter are considered. The proposed technique is first assessed in terms of the minimum iteration period obtained for some well-known DSP applications by using only operation-specific functional units in one case or by using hybrid library of functional units in the other case. Next, in order to study the impact of integrating the placement into the scheduling and allocation tasks, the proposed scheme is applied to the synthesis of the some benchmark problems. In this study, the results are obtained under three different scenarios for the candidate positions to place the functional units in the placement space. Finally, experiments are carried out to determine the placement area and iteration period for some problems by using a library contains only operation-specific functional units in one case or hybrid library of functional units in other case. The following are the three situations for evaluating the

candidate positions in the placement space: (i) The evaluation function gives a preference to the area over the iteration period (ii) The evaluation function gives a preference to the iteration period over the area (iii) The evaluation function gives equal preference to the area and the iteration period.

The delay and area of the functional units including that of the reconfigurable ones are taken from [64]. It is shown in [64] that an extra area is required to be added in order to introduce mode 2 (reconfiguration to adder) to a fixed multiplier. For example, an additional area overhead of 1.5% is required to implement one 32-bit Adder in a reconfigurable 16x16-bit morphable multiplier (RC-PM^{1A}). If mode2 has two adders (RC-PM^{2A}), the area overhead is pushed up significantly to be 11.4%. Normalized to the area of 32-bit Adder X: The area of the fixed 16x16-bit pipelined multiplier is 4.77X, and of the reconfigurable pipelined multipliers RC-PM^{1A} and RC-PM^{2A}, are 4.84X and 5.32X, respectively.

3.5.1 Obtaining the minimum iteration period

We first consider an example of a DSP filter, a fifth-order elliptic wave filter, in order to demonstrate the ability of proposed technique to find the minimum iteration period. We select the user parameters in the placement evaluation function so that it is formulated to give a preference only to the iteration period. The DFG of this filter is shown in Figure 3.9. The time schedules obtained for this filter in the two cases: (a) only operation-specific functional units (b) hybrid library of functional units are shown in Figure 3.10(a) and (b). It is seen that the iteration period obtained for the time schedule using reconfigurable functional units compared to that obtained using only operation-specific functional units since the in the case of reconfigurable functional units the local data

transfers are maximized. The proposed scheme is also applied to obtain a minimum iteration period for five DSP applications using the same two cases of functional units. Table 3.1, summarizes the synthesis results obtained for the five DSP applications. It is seen that, the use of reconfigurable functional units in synthesis process with the ICD provide an iteration period less than that using operation specific configuration.

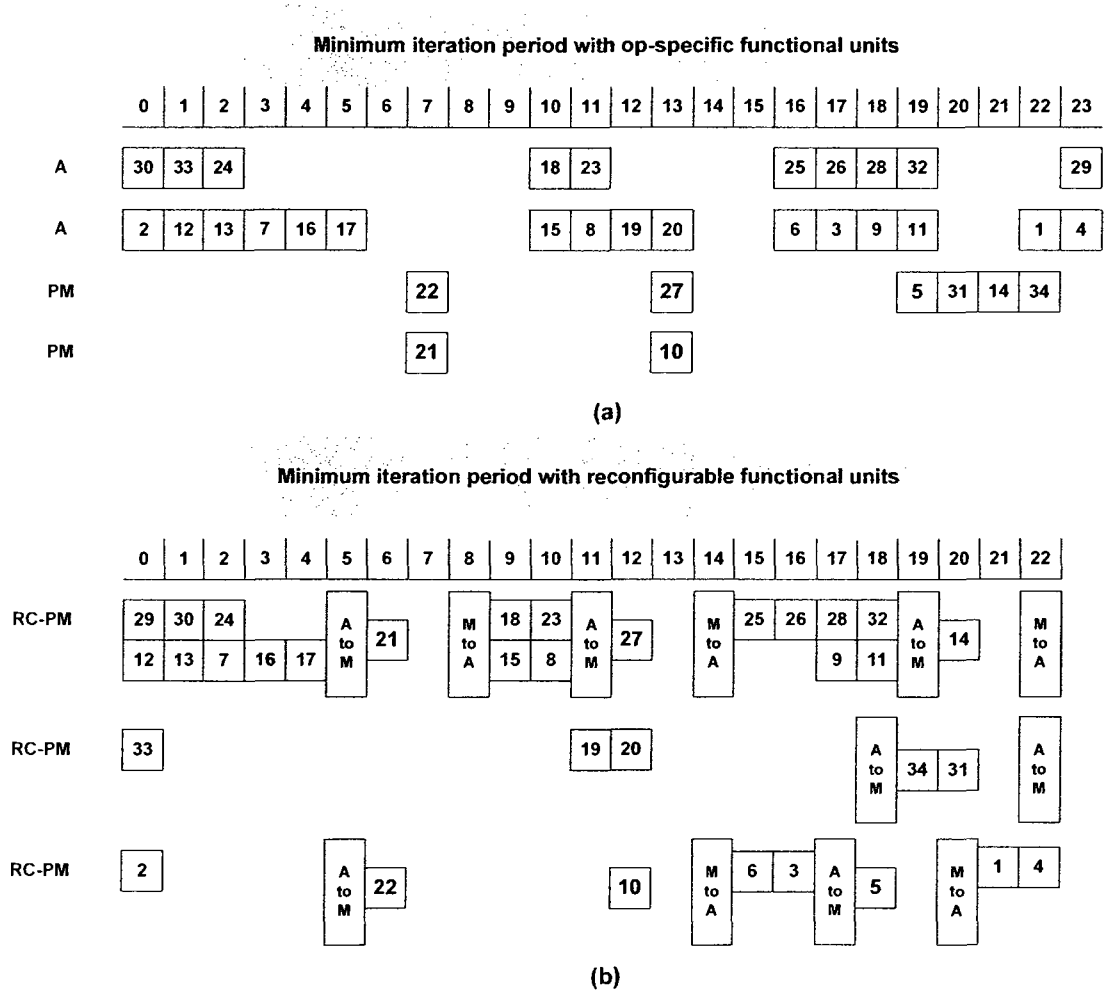


Figure 3. 10: Time and processor schedules for the DFG given in Fig. 3.9 using, (a) fixed operation-specific FUs (b) hybrid (reconfigurable) FUs

Table 3. 1: Minimum iteration periods of some DSP benchmark problems

Benchmark	Number of nodes	Type of FU	# FU	Minimum iteration period (T)
Fourth-order all-pole lattice filter	15	op-specific	2 PM, 3A	19
		Hybrid	1 RC-PM^{2,4}, 1 RC-PM^{1,4}, 1 PM	16
Fifth-order elliptic filter	34	op-specific	2 PM, 2A	24
		Hybrid	2 RC-PM^{1,4}, 1 RC-PM^{2,4}	23
8-point DCT	40	op-specific	3 PM, 3A	15
		Hybrid	1 RC-PM^{2,4}, 3 RC-PM^{1,4}, 1 PM	13
Raised cosine FIR	79	op-specific	5 PM, 5A	22
		Hybrid	3 RC-PM^{1,4}, 1 RC-PM^{1,4}, 1 PM	18

3.5.2 Placement area and iteration period results for three scenarios of candidate positions.

In order to assess the gain that can be obtained from the integration of the placement process into the high level synthesis rather than just provide to the high level synthesis a fixed information about the placement space, we assume that the proposed placement process is carried out under three different scenarios for the candidate positions. Two scenarios impose a restriction of one candidate position that can be evaluated for each mesh edge. The other flexible scenario is the proposed one in which more than one candidate position can be evaluated for each mesh edge according to its. The proposed

scheme for the three different scenarios is applied to various intensive benchmark DSP algorithms, namely, *DCT-dir, DCT-chem, DCT-feig*. The three scenarios as follows:

(1) The only candidate position of scenario 1 is: the corner of edges of $ce_j = 0$ group, the corner of edges of $ce_j = 1$ group, and finally, the center of edges of $ce_j = 2$ group.

(2) The only candidate position of scenario 2 is: the center of edges of $ce_j = 0$ group, the center of edges of $ce_j = 1$ group, and finally, the center of edges of $ce_j = 2$ group.

(3) The candidate positions of scenario 3 are the proposed one which is shown in figure 3.4.

The synthesis results in terms of the placement area and the iteration period for the three intensive benchmarks by using the various scenarios of candidate positions are shown, respectively, in Figs. 3.11 and 3.12. It is seen from the two figures that the proposed scheme produces better synthesis results in terms of both the area and iteration period when the proposed scenario of the candidate positions (that is scenario 3) is used during the placing of the functional units in the placement space than that when the other two restricted scenarios of candidate positions (scenarios 1 and 2) are used. Hence, we conclude that the more the restrictions and constraints are imposed by the placement process to the high level synthesis tasks the less the flexibility to provide better synthesis results. The situation became even worse when the information from a fixed placement is provided to the high level synthesis. The results shown in Figs 3.11 and 3.12 necessitate

the proposed approach in which the placement is solved simultaneously with the scheduling and allocation tasks. It is to be noted that, in case of scenarios 1 or 2, we can not prefer one over the other for the area or the iteration period. For example, scenario 1 gives better results than scenario 2 in terms of the placement area for the two benchmarks, *DCT-dir* and *DCT-chem* but the latter scenario is better in terms of the iteration period for the same two benchmarks. However, the situation in terms of placement area and iteration period is totally opposite for the two scenarios in case of the third benchmark, i.e., *DCT-feig*. In fact, the proposed scenario which use a hybrid of the two other scenarios provide more flexibility and, hence, better synthesis results for the three benchmarks in terms of the placement area and iteration period

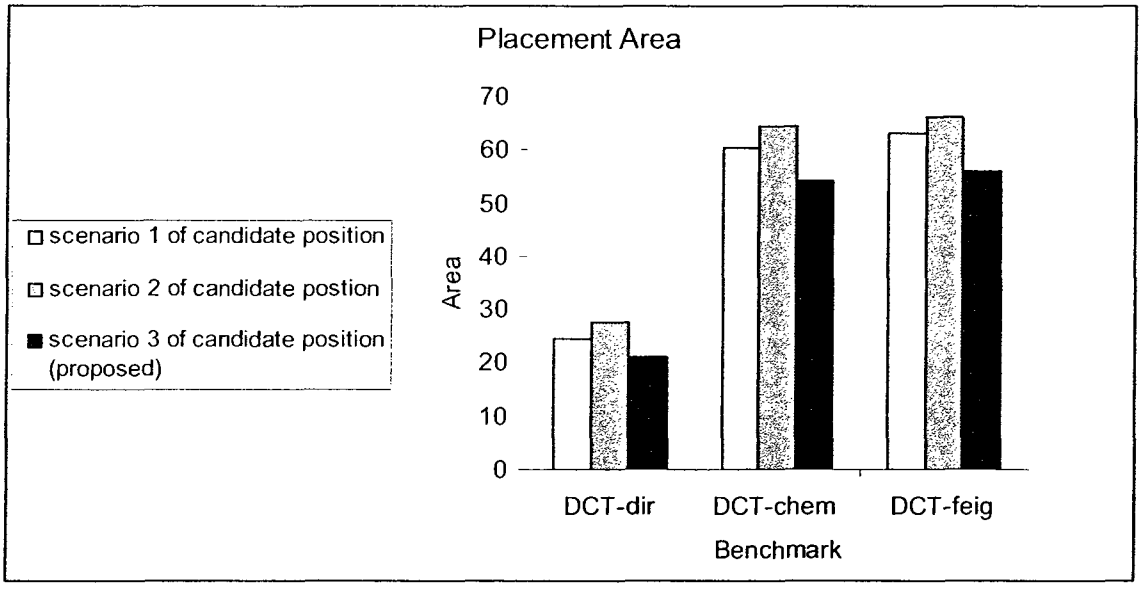


Figure 3. 11: Placement area obtained by applying the proposed technique three intensive DSP Benchmark problems with three different scenarios of candidate positions.

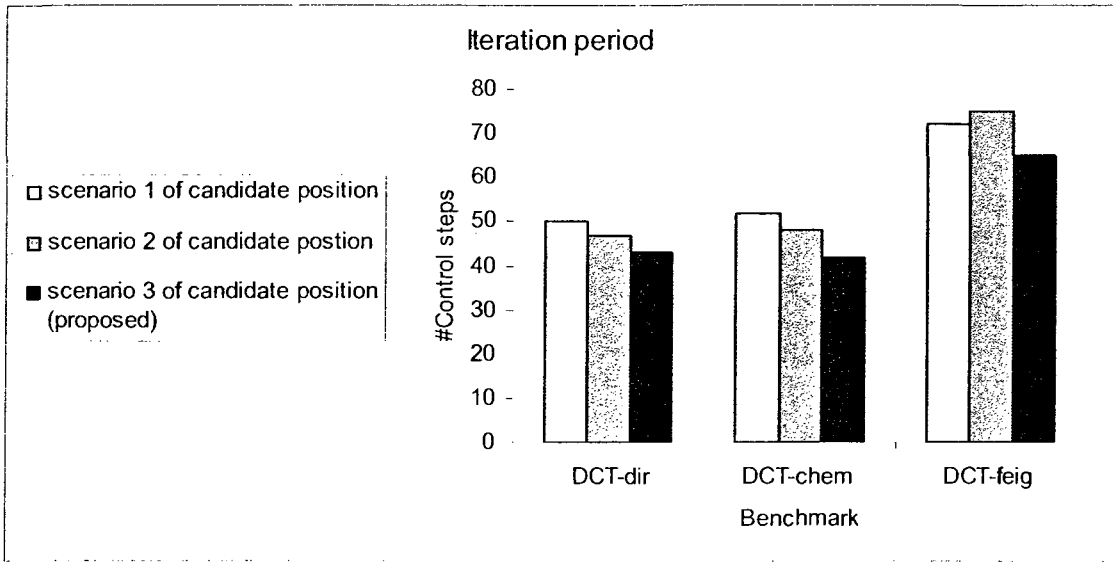


Figure 3. 12: Iteration period obtained by applying the proposed technique three intensive DSP Benchmark problems with three different scenarios of candidate positions.

The improvement in the synthesis results in terms of the placement area and the iteration period obtained by using scenario 3 of candidate positions is not associated with an overhead in terms of the computation time compared to that in case of the other two scenarios for all of the benchmark problems considered. For example, in case of the DCT-feig (547 nodes), the computation times in seconds are 2.34, 2.37, and 2.56 for scenario 1, 2, and 3, respectively. The computation times are reasonable and compare well with that obtained for the DCT-feig in Chapter 2 when the placement was not incorporated into the high level synthesis.

3.5.3 Placement area and iteration period results for three cases of the evaluation function.

In order to assess the effect of the choosing the two parameters α and τ in the evaluation function of the candidate positions given in Section 3.4.3 during the placement process, we select the parameters such that the area of the placement space has more priority to be minimized than the iterations period in one situation, the iteration period

has more priority to be minimized than the area of the placement space in the second situation, and the area and iteration period have equal preference in the third situation. The proposed technique is applied to a set of DSP benchmark problems to determine the placement area and iteration period for these problems by using a library contains only operation-specific functional units in one case or hybrid library of functional units in other case given. The parameters α and τ in three situations for evaluating the candidate positions in the placement space are selected as follows: (i) $\alpha > \tau$ when the evaluation function gives a preference to the area over the iteration period (ii) $\alpha < \tau$ when evaluation function gives a preference to the iteration period over the area (iii) $\alpha = \tau$ when the evaluation function gives equal preference to the area and the iteration period.

Table 3.2, Table 3.3, and Table 3.4 give the synthesis results in terms of placement area and iteration period for three setup situations of the evaluation function for the candidate position. It is seen from the three tables that the proposed technique when it is applied in presence of a hybrid library of functional units offers a substantial gain in terms of reducing both the placement area as well as the iteration period for all the benchmark problems compared to that in presence of only specific-operation functional units in the three situation of candidate positions.

In situation 1, in which a preference is given to the area over the iteration period, an average reduction of 17.77% is achieved in the placement area for the intensive DSP benchmarks considered with the minimum reduction being 3.16% for *DCT-chem* and the maximum 28.1% for *mcm*. On the other hand, an average reduction of 14.88% is achieved in the iteration period for the intensive DSP benchmarks considered with the minimum reduction being 6.81% for *mcm* and the maximum 19.64% for *DCT-chem*.

In situation 2, in which a preference is given to the iteration period over the area, an average reduction of 17.96% is achieved in the placement area for the intensive DSP benchmarks considered with the minimum reduction being 7.56% for *DCT-chem* and the maximum 22.25% for *DCT-planar* . On the other hand, an average reduction of 15.37% is achieved in the iteration period for the intensive DSP benchmarks considered with the minimum reduction being 8.82% for *DCT-feig* and the maximum 20.83% for *DCT-chem.*

In situation 3, in which an equal preference is given to the iteration period and the area, an average reduction of 16.55% is achieved in the placement area for the intensive DSP benchmarks considered with the minimum reduction being 8.37% for *DCT-chem* and the maximum 20.41% for *mcm.* On the other hand, an average reduction of 15.55% is achieved in the iteration period for the intensive DSP benchmarks considered with the minimum reduction being 10.95% for *DCT-feig* and the maximum 22.22% for *DCT-chem.*

Furthermore, by comparing the results shown in the three tables, it can be concluded that the proposed evaluation function when it sets $\alpha = \tau$ brings about a trade off between the iteration period and the placement area.

Table 3. 2: Placement area and iteration period obtained when the evaluation function gives the Area more preference than iteration period

Benchmark	Placement area		Reduction	Iteration period		Reduction
	Op-Specific	Hybrid		Op-Specific	Hybrid	
DCT-planar	30.17	24.65	18.29	36	30	16.66
mcm	19.57	14.07	28.10	44	41	06.81
IDCT	17.48	14.24	18.53	22	18	18.18
DCT-dir	25.42	19.58	22.97	55	46	16.36
DCT-chem	52.38	50.72	03.16	56	45	19.64
DCT-feig	61.71	52.07	15.62	77	68	11.68

Table 3. 3: Placement area and iteration period obtained when the evaluation function gives the iteration period more preference than area

Benchmark	Placement area		Reduction %	Iteration period		Reduction %
	Op-Specific	Hybrid		Op-Specific	Hybrid	
DCT-planar	37.52	29.17	22.25	31	26	16.12
mcm	23.82	18.76	21.24	39	34	12.82
IDCT	22.92	18.74	18.23	18	15	16.66
DCT-dir	31.94	24.19	24.26	47	39	17.02
DCT-chem	64.54	59.66	07.56	48	38	20.83
DCT-feig	71.39	61.22	14.24	68	62	08.82

Table 3. 4: Placement area and iteration period obtained when the evaluation function gives equal preference to area and iteration period

Benchmark	Placement area		Reduction %	Iteration period		Reduction %
	Op-Specific	Hybrid		Op-Specific	Hybrid	
DCT-planar	33.65	27.22	19.10	33	28	15.15
mcm	21.31	16.96	20.41	42	37	11.90
IDCT	20.74	16.92	18.41	19	16	15.78
DCT-dir	26.08	21.11	19.05	52	43	17.30
DCT-chem	59.12	54.17	08.37	54	42	22.22
DCT-feig	65.20	56.07	14.00	73	65	10.95

3.6 Summary

Most of the approaches for high level synthesis have not addressed the interaction with physical design such as the interconnect delays which led to unpredictable synthesis results and, hence, it significantly decreases the quality of the resulting implementation especially with the shrinking device features in sub-micron technologies. In this work, we have addressed the problem of integrating the placement of the functional units into the high-level architectural synthesis of DSP applications so that accurate information about

the interconnect delays needed for data communication between the processing units has been taken into consideration. In order to provide efficient modeling of the interconnect timing, a systematic and predictable process has been employed for the placement by using a Delaunay triangular mesh in the proposed scheme. Since this method of triangulation makes candidate positions well-distributed and maximizes the minimum angles of the mesh. Hence, we have avoided making closer candidate positions to place the functional units on adjacent edges connected with a narrow angle which in turn allows us to, quickly, find the suitable gaps to place the remaining functional units in the placement space. Furthermore, in order to maximize the local data transfers, a hybrid library of functional unit includes dynamically reconfigurable multiple-operation functional units and operation-specific functional units have been incorporated in the proposed approach. The incorporation of the hybrid library has been seen that it provides the designer of DSP applications with a greater flexibility to explore the design space. The proposed technique has been applied to well-known benchmark problems of DSP applications. The proposed scheme for the interaction between the high level synthesis and the fully flexible placement process provided a substantial gain in terms of reducing both the placement area as well as the iteration period for all the benchmark problems considered compared to the interaction with a restricted placement process. In overall, experimental results demonstrate the benefit and effectiveness of incorporating interconnect aware simultaneous placement, scheduling and allocation.

Chapter 4

Interconnect-Aware Register Binding for High-level Synthesis

4.1 Introduction

In Chapters 2 and 3, scheduling and allocation techniques have been proposed taking into account the interconnect delay of data transfers between the processing units. It is well-known that register binding is a crucial sub-task in a high level architectural synthesis of digital systems. Since the register binding affects the data transfer between the RTL components in the targeted architecture, the approaches to be used for carrying out the process of registers binding also have a great impact on the complexity and the performance of the interconnect paths used to communicate and transfer data from one module to the other in the RTL structure. In a behavioural description of a digital system,

variables are used for storing values. During the task of scheduling in the high level synthesis, temporary variables may be introduced to preserve values across control steps. A variable is said to live during a period starting the control step when it is produced and the one when it is consumed. Register sharing allows variables with non-overlapping lifetimes to reside in the same register. Without register sharing, each variable in the behavioural representation of a digital system is stored in a separate register leading to a large number of registers in the resulting architecture. An optimal solution to the register sharing problem yields an architecture with a minimum number of registers in the resulting architecture.

The register sharing is performed during the task of register binding in the high level synthesis. Moreover, the registers have an impact on the design attributes such as the delay of the RTL structures [68-70]. The way the process of registers binding is performed also has an impact on the complexity of the network of the interconnect paths required to transfer data among the RTL components [71]. Recently, it has been shown by several researchers that even with the most optimistic values of metal resistively and dielectric constant used in the interconnect technology, the signal delay time for global wires will continue to increase with the technology scaling down into a deep submicron region primarily due to the increasing length and resistance of the wires [72]-[74].

There have been several studies [75]-[77] in which increasing importance has been placed on the interconnects in deep submicron technology. These studies were focused mainly on the lower levels of the synthesis process such as during the task of placement and routing. However, the techniques that automate the design process with the use of high level synthesis, can no longer afford to perform synthesis tasks without

taking into account the effect of their design decisions on the wiring performance of resulting designs. The technique of [78] shows a significant reduction in overall power by taking into account interconnects in the high level synthesis. It is also important to develop register binding schemes that can take into consideration their effect on the complexity of the associated interconnect network. The complexity of such an interconnect network can be measured in terms of the complexity of the multiplexer network used in the RTL structure [79].

The previous approaches used to solve the register binding problem in the high level synthesis can be categorized into two major groups. The first group performs the register binding simultaneously with the time scheduling and processor allocation in the high level synthesis. The problem of obtaining a simultaneous optimal schedule and processor allocation has been proven to be NP-hard, that is, it is a problem which is not solvable by deterministic algorithms in a polynomial time [80]. Hence, the approaches used for solving the various tasks of the high level synthesis simultaneously must use efficient heuristics to be practical only for large size problems. Examples of such simultaneous approaches include simulated annealing [81], simulated evolution [82], and integer linear programming (ILP) [83], [84].

In the second group, the task of register binding is solved separately from scheduling and processor allocation [85]-[88]. However, by using this approach of decomposing of the synthesis task into sub-tasks, the results of the subtask performed first become constraints for the succeeding subtask. Hence, these techniques, at best, produce a register binding solution with the number of registers that is constrained by the lower bound of the registers provided by the scheduled data flow graph. Moreover, the

optimal Left Edge technique [85] solve the task of register binding without taking into consideration its impact on the complexity of the interconnect network.

On the other hand, the techniques of register binding in [86]-[88] do take into consideration the interconnect minimization. Although [86] and [87] provide better results in terms of interconnects, there is an overhead of additional registers that result from the register binding. The technique proposed in [88] is applicable only for an FPGA implementation having embedded memory blocks.

In this chapter, the problem of register binding [89][90] in a high-level architectural synthesis of DSP algorithms is studied. A technique for binding the tokens produced by the nodes of a scheduled DFG is proposed while aiming at minimizing the number of interconnects. First, a segmentation scheme in which the lifetime of a token is appropriately divided into multiple segments is developed. Then, the register binding problem is formulated as a min-cost flow problem so that the tokens having the same source and/or destination are bound into the same register and results in a reduced numbers of registers and interconnects.

The chapter is organized as follows. In Section 4.2, a technique for binding the variables produced by the nodes of a scheduled DFG is presented by developing a segmentation scheme in which a single-segment lifetime is appropriately partitioned to form multiple-segment lifetimes. Then, a flow network [91] is constructed in order to assign the segments to registers taking into consideration the interconnect complexity. In Section 4.3, the proposed technique is applied to some intensive benchmark DSP

problems and compared with other techniques in the literature in terms of the numbers of registers and interconnects. Section 4.3 summarizes the work presented in this chapter.

4.2 Proposed Technique for Register Binding

In this section, a technique for binding to registers the variables produced by the nodes of a scheduled DFG is presented. A segmentation scheme is developed in which a single-segment lifetime is appropriately partitioned to form multiple-segment lifetimes giving more freedom in binding the variables to registers. The binding task can be efficiently performed by using a flow network, which takes into consideration the complexity of the interconnects. Storage units, such as registers, are required for the processing of a DSP application in order for them to store the tokens produced by the execution of the nodes of the DFG representing the DSP application.

Before presenting our technique for optimizing the number of registers, we will briefly discuss the need for a register binding and the factors that affect the number of registers required for a proper implementation of a DSP application. During the execution of a DSP application, the filters coefficient and the token data produced by a node should be stored in a storage unit as long as it is still needed by some other nodes. During each cycle of the execution of the nodes, the number of storage units required depends on the maximum number of tokens that are concurrently produced in a single control step and, therefore, need to be stored. The iterative execution of a DSP application implies that a node produces a token during each iteration of the execution. In our scheme, registers are used to store the token data in view of their short access time.

The register binding could be either overlapped or non-overlapped depending on whether or not more than one token can share the same register. In the proposed register binding technique, we use the overlapped binding whenever it is possible for more than one token to share the same register during the same iteration period in order to reduce the total number of registers. In the proposed technique, the memory consistency is ensured by the register binding, since all the tokens associated with a single register have disjoint lifetime periods.

Let us now define certain terms that are used in this section for node regeneration. Assume that a node v produces a token Y_v at time t_{Y_v}. This token is later consumed by a set of n nodes w_i that are scheduled to fire at scheduling time t_i. Assume that each of these nodes is connected to the node v via a set of n edges $e_i = (v, w_i)$ each associated with N_i ideal delays. A lifetime L_v of a token data produced by a node v is defined as the difference between the time a token data is produced and the latest time when it is consumed, and it is given by

$$L_v = \max_{i=1,\ldots,n} [(N_i \cdot T + t_i) - t_{Y_v}] \tag{4.1}$$

where T is the iteration period, and $t_{Y_v} = t_v + d_v$, t_v and d_v being, respectively, the firing time and the computational delay of the node v. If $L_v = 0$, the token produced need not to be stored in a register; instead, it can be directly sent to the consumer.

We now perform the lifetime segmentation scheme by partitioning the single-segment lifetime to form multiple-segments. In the cyclic data flow graph, the life time of a token Y_v produced by the node v may span over one or more iteration periods

depending on the control step $C_{Y_v} = t_{Y_v}$ modulo T at which the token is produced and the duration of the life time of the token. In such a case, the duration of the life time has to be partitioned into sub-periods such that each sub-period S_i consists of only a part of the life time duration and appears in only one iteration period. For a given token Y_v, the number of sup-periods m into which the duration of the lifetime has to be partitioned is given by

$$m = \left\lceil \frac{L_{Y_v} + C_{Y_v}}{T} \right\rceil \tag{4.2}$$

The length of the sub-period S_i is given by

$$S_i = [min(((L_v - \sum_{j=0}^{i-1} S_j) + X_i), T)] - X_i , \qquad i = 0, 1, ..., (m-1) \tag{4.3}$$

where X_i is the control step at which the sub-period S_i is produced, and it is given by

$$X_i = \begin{Bmatrix} C_{Y_v} & i = 0 \\ 0 & i = 1, ..., (m-1) \end{Bmatrix} \tag{4.4}$$

Based on (4.3) and (4.4), for each token Y_v, we can now construct a life time set $\beta_{Y_v} = \{(X_0, S_0), (X_1, S_1), ..., (X_{m-1}, S_{m-1}\}$.

The number of registers NR_{Y_v} required for allocating a token Y_v is equal to the number of sub-periods of the lifetime, that is, $NR_{Y_v} = m$. It is to be noted that by employing the reusability concept, one register can be used for allocating the two sub-

periods, S_0 and S_{m-1}, if and only if $S_0 + S_{m-1} \leq T$. Hence, the number of registers required for allocating the token Y_v can be reduced by one.

For a cyclic scheduled DFG, the lifetimes of tokens can not be represented by intervals on a straight line, since it may span over one or more iteration periods. In this situation the lifetimes can be conveniently represented by arcs around a circle representing the iteration period. Since all the control steps together form the iteration period, the circle representing the iteration period gets divided equally into a number of arcs equal to the number of control steps.

Fig. 4.1 shows an example of circular lifetime chart of tokens. In this example, four tokens with different lifetimes are given. The lifetime token Y_{v_4} is the only token in this example whose lifetime lies in two iteration periods. The width Wid_{c_i} of lifetime chart *CLC* for control step c_i is the number of lifetimes overlapping associated with control step c_i. The maximum width of a circular lifetime chart $Wid_{\max}$ is the maximum width Wid_{c_i} over all c_i in *CLG*, i.e., $Wid_{\max} = \max_{\forall c_i}(Wid_{c_i})$.

A directed graph is called the compatibility graph *CG(N, A)*, if it is constructed according to the following rules (see the circular lifetime chart *CLC* shown in Fig. 4.1): (i) corresponding to each token Y_v in *CLC* there is a node $n_{Y_v} \in N$ in the compatibility graph *CG(N, A)*, (ii) there is a directed edge $(n_{Y_u}, n_{Y_v}) \in A$ a pair of nodes n_{Y_u} and n_{Y_v} in *CG(N, A)* if and only if the lifetimes L_{Y_u} and L_{Y_v} do not overlap at any control step, and

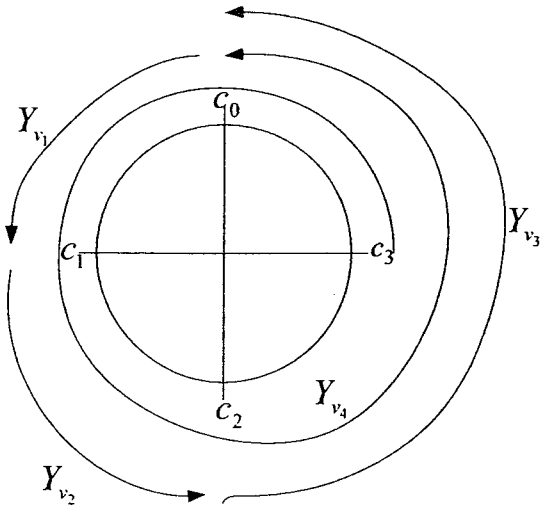


Figure 4. 1: Circular lifetime chart

(iii) the death time (*dtime*) of the token Y_u (the control step at which Y_u is consumed, i.e., $dtime\,(Y_u) = C_{Y_u} + L_{Y_u}$) is less than the birth time (*btime*) of the token Y_v (the control step at which Y_v is produced, i.e., $btime\,(Y_v) = C_{Y_v}$). Thus, $dtime(Y_u) < btime(Y_v)$.

It is to be noted that according to rule (i) for constructing $CG(N, A)$, a token having a lifetime that lies in more than one iteration period cannot be included in the compatibility graph, since such nodes overlap with all other nodes. In order to include such tokens in the compatibility graph, we incorporate the set β_{Y_v} (as given by (4.3) and (4.4)) of sub-periods of the lifetime of a token for the construction of the compatibility graph so that each node in the compatibility graph corresponds to a sub-period in the set β_{Y_v} of a token Y_v. This modification provides more flexibility in the register binding, since it allows the sub-periods of the tokens to share registers with other tokens. Referring again Fig. 4.1, we notice that according to (4.3) and (4.4), $\beta_{Y_{v_4}} = \{(c_3, 1), (c_0, 4)\}$, $\beta_{Y_{v_3}} = \{(c_2, 2)\}$, $\beta_{Y_{v_2}} = \{(c_1, 1)\}$, and $\beta_{Y_{v_1}} = \{(c_0, 1)\}$. Since the sub-

period $(c_0,4)$ of $\beta_{Y_{v_4}}$ (or $(c_0,4)_{\beta_{Y_{v_4}}}$) span over 4 control steps and equals to one iteration period hence this sub-period cannot be compatible with any other sub-period of the lifetime of any other token. However, the sub-peiod $(c_3,1)$ of $\beta_{Y_{v_4}}$ (or $(c_3,1)_{\beta_{Y_{v_4}}}$) can be compatible with $\beta_{Y_{v_1}}(c_0,1)$ or $\beta_{Y_{v_2}}(c_1,1)$, and therefore, can share the same register with them. Fig. 4.2(a) shows the compatibility graph obtained from the circular lifetime chart given in Fig. 4.1.

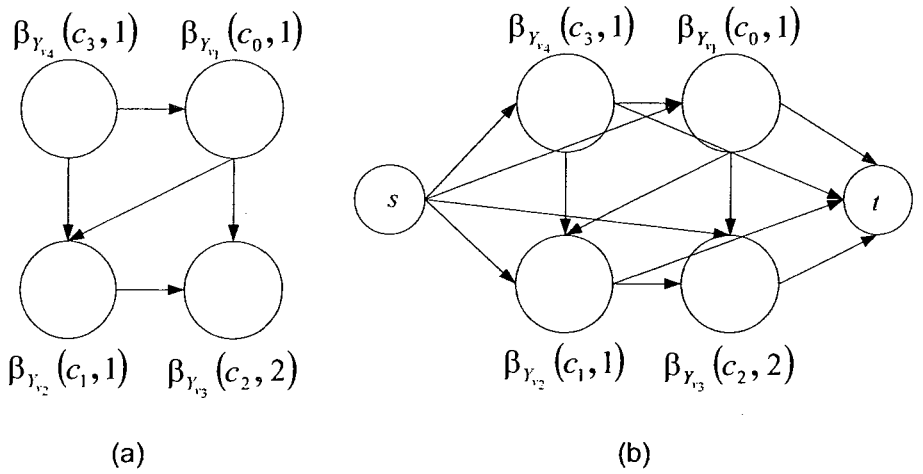


Figure 4. 2: (a) Compatibility graph obtained from the CLC of Fig. 4.1. (b) The corresponding network graph

We now construct a flow network $G(M,E)$ from the compatibility graph $CG(N,A)$ by introducing to it two additional nodes, namely, a source node s and a sink node t, such that $M = N \cup \{s,t\}$ and E is the set of edges in $G(M,E)$ that contains the set of edges A of $CG(N,A)$ plus additional edges from the node s to every node in $CG(N,A)$, and from every node in $CG(N,A)$ to the node t (see Fig. 4.2(b)).

The binding of tokens having the same source and/or destination into the same register reduces the number of interconnect. Since the processor allocation has already

been carried out, it is easy to assign a weight to an edge connecting a pair of nodes in $G(M,E)$ by using the concept of common producer and common consumer of the tokens corresponding to the pair. Such weight represents the cost of the register sharing between the pair of nodes. The weight assigned to an edge connecting a pair of nodes, which represent the sub-periods of tokens, is defined as

$$w_{(n_{Y_v},n_{Y_u})} = -L \cdot (F + \mathrm{CC}_{(n_{Y_v},n_{Y_u})}) \tag{4.5}$$

where F is a Boolean variable equal to 1 if the pair of tokens have the same producer, $\mathrm{CC}_{(n_{Y_v},n_{Y_u})}$ is the number of consumers common to the pair of tokens, and L is given as $L = \max_{\forall(n_{Y_v},n_{Y_u})} (F + \mathrm{CC}_{(n_{Y_v},n_{Y_u})})$.

In the flow network $G(M,E)$, a capacity of one is assigned to each edge $e \in E$ representing the maximum flow possible for an edge. The maximum flow in the flow network $G(M,E)$, i.e., the maximum flow from the terminal node, is set as $FL = Wid_{max}$.

Assume that $Wid_{max} = k$ for a given *CLC*, then *FL* in $G(M,E)$ is set to *k*. The register binding problem is solved by sending *k* units of flow from the source node to the sink of $G(M,E)$ such that each node is visited exactly once, all the nodes are covered, and the overall cost is minimized. Since the capacity of each of the edges is one, it is guaranteed that each flow will follow a different edge-disjoint path, $P_1,\ldots,P_k$, in $G(M,E)$. However, the paths may not be node disjoint, if they are not so in the original compatibility graph $CG(N,A)$. To ensure the generation of node disjoint paths as well, a

node separation technique [92] can be used. In this technique, each node is duplicated and then the pair is connected by an edge. All the edges outgoing from the original node after the duplication are made to be outgoing from duplicate node as seen from the example of Fig. 4.3. The node and its duplicate are connected by an edge with capacity of 1 so that only a single flow can pass through a node because of the unit capacity assigned to the edge connecting the node and its duplicate node. A cost of $-C$ where $C = \left| \sum_{\forall (n_{Y_v}, n_{Y_u})} w_{(n_{Y_v}, n_{Y_u})} \right| + 1$ is also assigned to the edge. This choice of cost, as to be shown later, ensures the coverage of all the nodes.

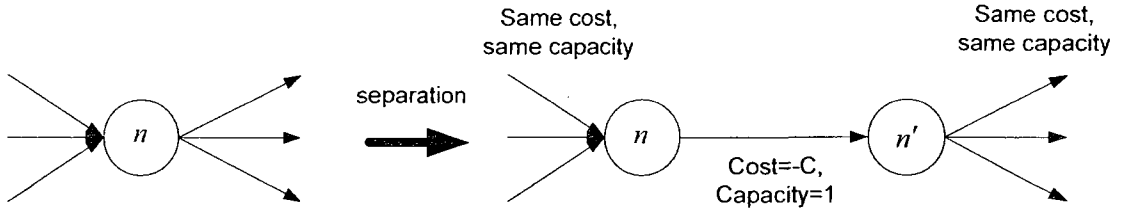


Figure 4. 3: An example of node separation

After performing the node separation, we apply min-cost flow technique to obtain a register binding with minimum number of interconnects. The min-cost flow [91] in the network finds k paths each corresponding to one clique such that the total cost is minimized. The set of nodes in the k generated paths form k cliques each being bound into a single register. The total cost on each individual path is the sum of the cost of all the individual edges in that path. Since the costs of the edges in $G(M, E)$ are negative, the more the nodes in the path the less its cost. Thus, the min cost flow that guarantees the minimization of the total cost also guarantees the coverage of all nodes, since otherwise,

the coverage of the nodes not already covered will reduce the total cost even further thus implying that min cost flow has not provided a minimum cost solution.

4.3 Experimental Results

In this section, some well-known benchmark examples of synthesizing intensive DSP applications using the technique presented in this chapter are considered. Starting from the scheduled DFG corresponding to a given DSP algorithm, the process of synthesis is carried out to obtain register binding. In our experiments, both centrally shared and distributed-register based architectures are targeted. The proposed register binding technique is first assessed in terms of the number of registers and the number of interconnects required when the ICD is ignored for both the centrally-shared- and distributed-register based architecture and the corresponding results are compared with those obtained by using the methods proposed in [85] (left-edge method), [86], and [87]. The proposed register binding technique is also assessed in terms of number of registers required for DFGs that have scheduled with and without taking ICD into consideration for some DSP filters.

4.3.1 Number of Registers and Interconnects

In order to assess the proposed register binding technique without the incorporation of interprocerssor communication delay, the proposed register binding technique and the methods in [85], [86], and [87] are applied to various intensive benchmark DSP algorithms, namely, *ellip, fir, DCT-planar, mcm, DCT-dir, DCT-chem, DCT-feig*. The number of operational nodes in the intensive DSP benchmarks considered varies from 34 to 547. The synthesis results in terms of the number of registers and the

number of interconnects obtained by the various register binding methods for the centrally-shared or the distributed register-based architecture are shown , respectively, in Figs. 4.4 and 4.5.

It is seen from Fig. 4.4 that the register binding technique even without the incorporation of node regeneration outperforms the methods of [86] and [87], and it produces a number of registers equal to that obtained by the left-edge method [85], which produces an optimal number of registers for both centrally-shared and distributed register based architectures. On the other hand, Fig. 4.5 shows that the proposed register binding technique without the incorporation of node regeneration (Chapter 5) significantly outperforms the left-edge method in term of the number of interconnects, whereas it results in the number of interconnects that is less than that provided by most of the other methods of [85] and [20], except in the case of a very few examples where the number of interconnects are the same.

Further, it can be seen from Figs. 4.4 and 4.5 that for each of the four methods the number of registers in centrally-shared register-based architecture is less than that in the distributed architecture, whereas the number of interconnects is higher in all of the DSP examples considered. This is expected in view of the fact that the sharing of registers is more in a centrally-shared register based architecture than in the corresponding distributed one, which in turn increases the complexity of the interconnect network in the architecture and, hence, the number of interconnects in the former.

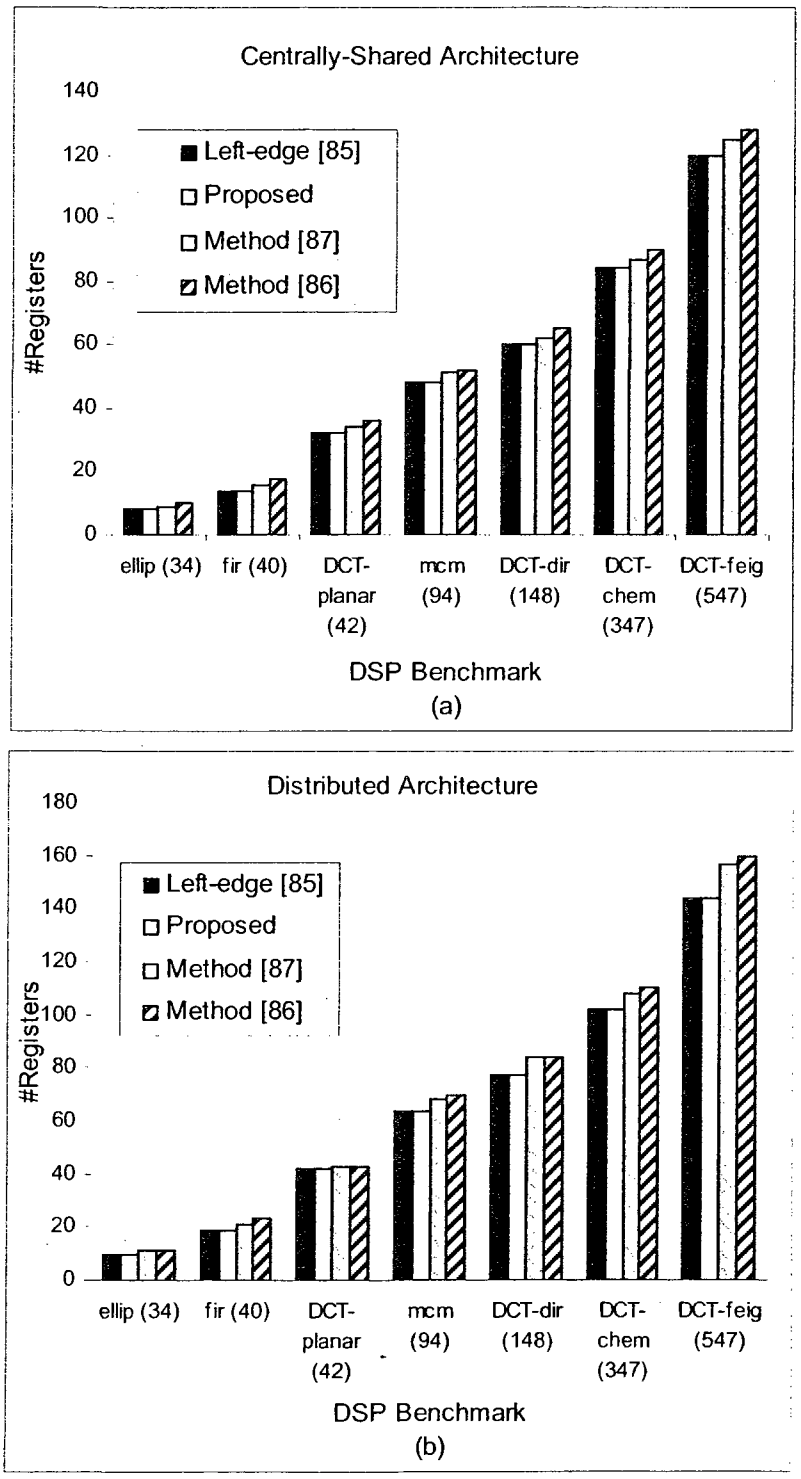


Figure 4. 4: Number of registers obtained for some intensive DSP Benchmark problems by using the proposed register binding without node regeneration, the left-edge method [85], the method of [86], and the method of [87] targeting (a) centrally-shared architectures or (b) distributed register-based architecture.

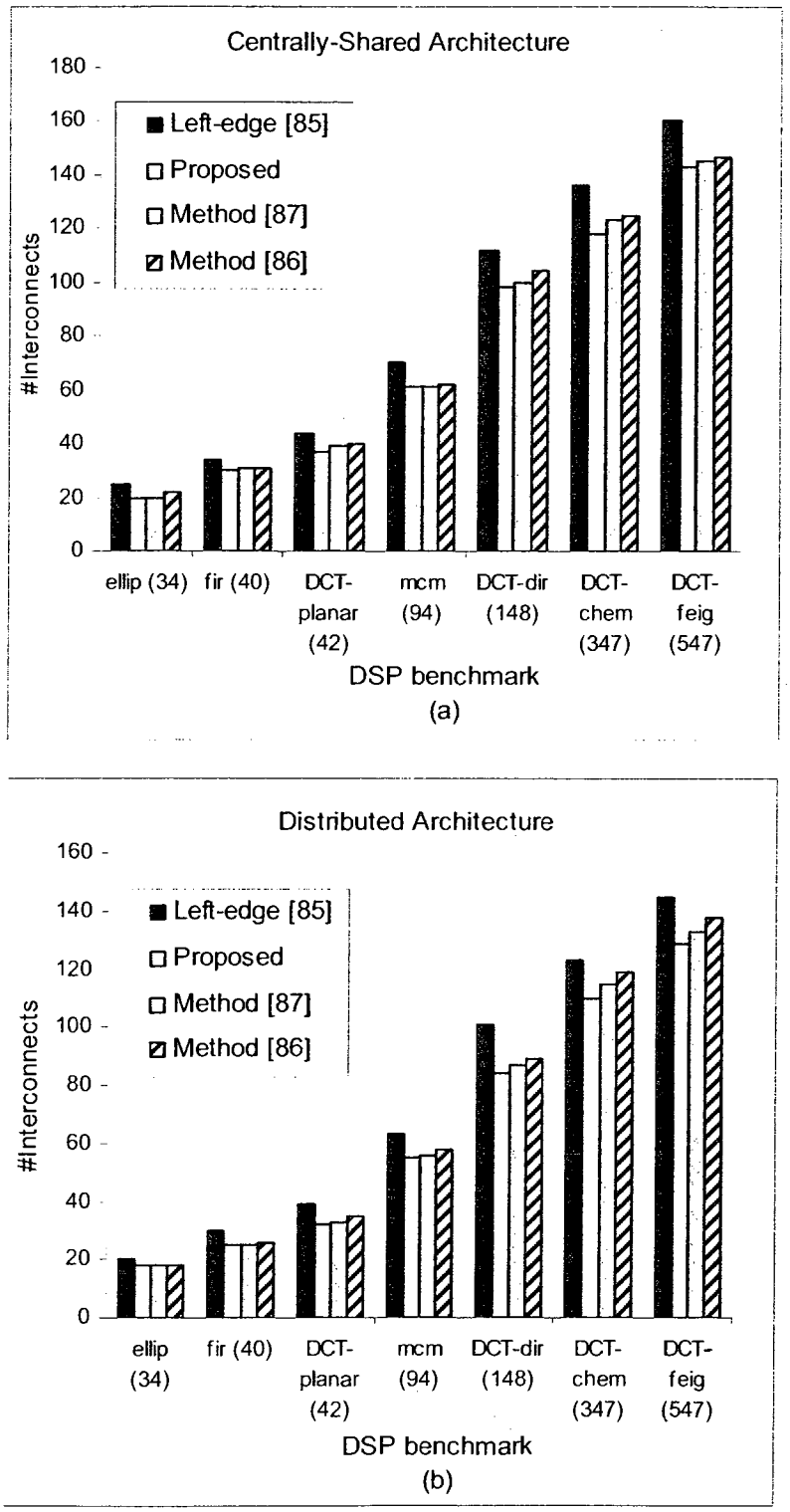


Figure 4. 5: Number of interconnects obtained for some intensive DSP Benchmark problems by using the proposed register binding without node regeneration, the left-edge method [85], the method of [86], and the method of [87] targeting (a) centrally-shared architectures or (b) distributed register-based architectures.

4.3.2 Comparison of various schemes in terms of the number of registers

Various register binding schemes have been also proposed in the synthesis techniques of FDLS [22], OSAIC [20], InSyn [21], method of [38] and MARS [22] [23], [39]. A comparison in terms of the number of registers between the proposed technique and these other techniques is carried out. Table 4.1 gives the number of registers obtained for the fifth-order elliptic wave filter using the various register binding techniques. In the case of the proposed technique, the results are provided both with and without the ICD taken into consideration, while it is neglected in the case of other techniques. In this table, NR_{p_1} and NR_{p_2} are number of registers normalized with respect to the number of registers obtained using the proposed technique, respectively, without and with the ICD is taken into consideration. It is clear from the results listed under NR_{p_1} and NR_{p_2} that proposed technique provides better synthesis results in terms of the number of registers not only when the ICD is ignored but also in all cases, with the exception of the method of [21], even when the ICD is taken into consideration. Note that NR_{p_2} values in the table would get even larger, had these other methods taken the ICD into consideration. Similar comparison results are obtained when the various synthesis techniques are applied to other benchmark DSP filters.

4.3.3 Number of registers for various DSP benchmarks resulting from the proposed register binding scheme with and without ICD.

The proposed register binding scheme is applicable to any given scheduled DFG. In order to assess the proposed scheme in terms of number of register required for the either with or without ICD, it is applied to various benchmark DSP filters. Table 4.2 gives the

iteration period and the number of registers obtained with and without the ICD by using the proposed synthesis technique and MARS technique for each DSP.

It is seen from the table that the proposed technique reduces the number of registers in both situations with and without ICD.

Table 4. 1: Number of registers required by a fifth-order elliptic filter obtained by using various synthesis techniques

Technique	No. of Registers, R	$NR_{P1} = R/R_{P1}$	$NR_{P2} = R/R_{P2}$
MARS[22][23][39] (W/O ICD)	9	1.285	1
FDLS [18] (W/O ICD)	12	1.714	1.333
OSAIC [20] (W/O ICD)	10	1.428	1.111
InSyn [21] (W/O ICD)	8	1.141	0.888
Method of [38] (W/O ICD)	10	1.428	1.333
Proposed (W/O ICD)	7 (R_{P1})	1	0.777
Proposed (with ICD)	9 (R_{P2})	1.285	1

Table 4. 2: Number of registers using proposed technique with and without ICD compared to that obtained by the MARS technique

DSP filters	(Non negligible ICD)			(Negligible ICD)		
	Iteration period	Number of registers		Iteration period	Number of registers	
		MARS	proposed		MARS	proposed
Fifth-order elliptic wave filter	24	10	9	16	8	7
Fourth-order Jaumann filter	21	7	6	16	6	5
Fourth-order All-pole filter	19	6	5	14	5	4
Second-order filter	5	3	2	3	3	2

4.4 Summary

Register binding is one of the main tasks in a high level architectural synthesis of digital systems. The function of register binding is to assign the tokens produced by the processing units to registers in the resulting RTL structure and it is done in a way so as to minimize their number. Since the register binding affects the data transfer among the RTL components, the way the process of registers binding is performed also has an impact on the complexity of the network of the interconnect paths used to transfer data. With the technology scaling down into a deep submicron region, the register binding can no longer afford to be performed without taking into account its effect on the wiring complexity of the resulting architectures. In this chapter, the problem of register binding in a high-level architectural synthesis of DSP algorithms has been studied. A technique for binding the tokens produced by the nodes of a scheduled DFG has been proposed while aiming at minimizing the number of interconnects. First, a segmentation scheme in which the lifetime of a token is appropriately divided into multiple segments is developed. Then, the register binding problem is formulated as a min-cost flow problem so that the tokens having the same source and/or destination are bound into the same register and results in a reduced numbers of registers and interconnects. In order to assess the proposed technique of register binding, it has been applied to the synthesis targeting centrally-shared and distributed register based architectures for different intensive DSP algorithms and has been compared with various other commonly used synthesis methods for register binding. The results of these experiments have shown that the proposed register binding technique produces the number of registers equal to the optimal solution

provided by using the left-edge method [85] and it outperforms other methods not only in terms of the number of registers but also in terms of the number of interconnects.

Chapter 5

Interconnect Aware Node Regeneration Scheme for Register Minimization

5.1 Introduction

The lower bound on the number of registers resulting from any register binding technique gets fixed once the DFG has been scheduled. For scheduled DFG, the processing units could be idle for one or more control steps. In the context of register binding problem, this idle state of a processing unit if utilized could minimize the lifetimes of the variables. Generally, in order to reduce the lifetime of a variable in a schedule, the node producing that variable must be scheduled as close as possible to the consuming node. However, this is not always possible due to the precedence relation constraints between nodes in a given behavioral description. Instead of storing a variable, according to its lifetime, in a

register for a long period, it is possible under some conditions to reproduce the variable at a time closer to the firing time of each of its consumers. The idle processing units can then be utilized to perform the task of reproducing the variable by regenerating a copy of the original node that produces it. Thus, if the hidden flexibility of node regeneration for a scheduled DFG could be exposed, then this flexibility can be utilized to minimize the lifetime of the variables. This flexibility can be also used to minimize the interconnect requirements by appropriately assigning the regenerated copies of the original nodes to the idle processing units. Since register binding in a decomposed high level synthesis is constrained by the lower bound on the number of registers imposed by a given scheduled DFG. On the other hand, it should be possible to decrease this lower bound by exposing of the hidden flexibility for node regeneration in a given scheduled DFG without rescheduling it.

In this chapter [89], an interconnect-aware register minimization technique is presented by proposing a node regeneration scheme that generates multiple copies of the original nodes with the resulting variables having lifetimes shorter than those of the variables produced by the corresponding original nodes. The freedom provided by having multiple copies of nodes is then further exploited to assign each copy to a processing unit that results in minimizing the complexity of the interconnect network thus obtained.

The chapter is organized as follows. In Section 5.2, a theoretical formulation of the proposed scheme for node regeneration is presented and the conditions under which such node regeneration is possible are described. A technique is then developed in Section 5.3 in which a flow network [91] is constructed in order to assign the regenerated nodes to idle processing units. In Section 5.4, the proposed technique is applied to some

intensive benchmark DSP problems and compared with other techniques in the literature in terms of the numbers of registers and interconnects. Further, in order to show the impact of the incorporation of node regeneration scheme in the proposed register binding technique on the total interconnects length of the resulting RTL structure, some well-known benchmark problems are also synthesized in this section. Section 5.5 summarizes the work presented in this chapter and highlights some of the salient features of the proposed technique.

5.2 Process of Node Regeneration for Register Minimization

Since the number of registers in register binding techniques are constrained by a lower bound imposed by the given scheduled DFG, in this section, we present a scheme for carrying out register minimization while taking into the consideration interconnect requirements of the underlying architecture by making appropriate modification in the firing times of some of the nodes in a given scheduled DFG. The modification in firing times of the nodes is performed by applying a new scheme referred to as node regeneration method.

In a given scheduled DFG, the processing units could be idle for one or more control steps. In the context of register binding problem, this idle state of a processing unit if utilized could minimize the lifetimes of the variables. Generally, in order to reduce the lifetime of a variable in a schedule, the node producing that variable must be scheduled as close as possible to the consuming node. However, this is not always possible due to the precedence relation constraints between the nodes of a given behavioral description. Instead of storing a variable in a register for a long period according to its lifetime, it is possible under certain conditions to reproduce have

flexibility in reproducing the variable at a time closer to the firing time of each of its consumers. Then, an idle processing unit can be utilized to perform the task of reproducing the variable by generating a copy of the original node that produced the variable. Thus, if such a flexibility of node regeneration for a scheduled DFG exists, then it can be utilized to minimize the lifetime of the variables. This flexibility can also be used to minimize the interconnect requirements by appropriately assigning the regenerated nodes to the idle processing units.

We now present a scheme for node regeneration and the conditions under which such a node regeneration is possible.

The problem of register minimization using node regeneration can be formulated as follows: Given the time schedule and processor allocation for a given data flow graph, calculate the lifetimes for all the tokens in the given schedule, then apply node regeneration method to minimize the number of registers without decreasing the throughput or increasing the number of processors.

Assume that in a given DFG, a node v_i is connected to r destination nodes v_j via r edges e_{ij}. Let $s < r$ of the r nodes v_j be scheduled to fire far from node v_i. A node v_j is said to fire far from node v_i if the lifetime of the token produced by v_i is not equal to this token's minimum lifetime imposed by the precedent relation between the two nodes. If an idle processor of the same type as that of the node v_i is available, then a new copy of this node is regenerated on this processor in order for it to be fired at a time close to the originally scheduled firing times of the s nodes instead of saving the token produced by v_i for a long time in a register for later consumption by the s successor nodes. This node

regeneration is possible if the operand(s) to be consumed by the node v_i are still alive after the original firing of the node v_i.

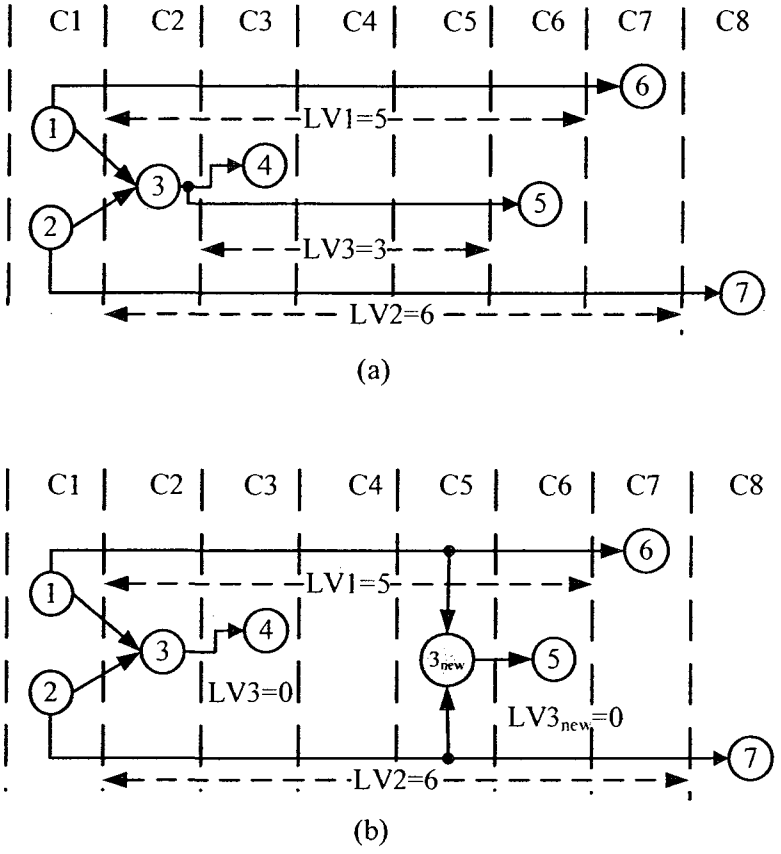


Figure 5. 1: (a) Lifetimes before node regeneration (b) Lifetimes after regeneration of node 3

The proposed node regeneration scheme is illustrated in Fig. 5.1. Fig. 5.1(a) shows a scheduled DFG in which the nodes v_1, v_2, and v_3 produce three tokens with overlapped lifetimes *LV1, LV2* and *LV3* respectively. Therefore, three registers are required for allocating these overlapped tokens. Fig. 5.1(b) shows how the node v_3 is regenerated as v_{3new} at the control step c_5 by using the long lifetimes *LV1* and *LV2* of the tokens produced by the nodes v_1 and v_2, the predecessor nodes of v_3. As a result of this simple regeneration, the token produced by the node v_3 need not be saved in a separate

register unlike in Fig. 5.1(a), where it was required to be saved until the node v_5 became ready to be fired. The lifetime *LV3* of the token produced by v_3 is zero in the new scheduled DFG of Fig. 5.1(b). Hence, only two registers are required as compared to three in Fig. 5.1(a). The following two theorems establish that by using this node regeneration method, it is possible to reduce the lifetime of the token produced by the regenerated node from its current value, if such a node is scheduled to fire at its latest firing time with respect to its successor nodes that constitute a subset of the successor nodes to the original node from which the regenerated node was created.

THEOREM 1 *Let v be a node in a DFG with the firing time t_v and computational delay d_v such that all of its predecessor nodes u_i and successor nodes w_i have fixed firing times t_{u_i} and t_{w_i}, respectively. Then, the node v can be regenerated as node v_{new} to fire at a time $t_{v_{new}}$, such that* $t_v + d_v \le t_{v_{new}} \le min\left(\begin{array}{l} \min\limits_{\forall u_i \in predecessors\ of\ v} (t_{Y_{u_i}} + L_{u_i}), \\ \min\limits_{\forall w_i \in successors\ of\ v_{new}} (t_{w_i} - \max\limits_{all\ P_{v_{new}w_i}} len[P_{v_{new}w_i}]) \end{array}\right)$, *if there exists an idle processor at time $t_{v_{new}}$.*

Proof. The firing rule of a node implies that this node can be fired on an idle processor if the node's operand(s) are available and the precedent relations are satisfied. The availability of the operand(s) of the node v_{new} is ensured by $t_{v_{new}} \le \min\limits_{\forall u_i \in predecessors\ of\ v} (t_{Y_{u_i}} + L_{u_i})$, which means that the operand(s) of v_{new} are still alive at or after $t_{v_{new}}$. Moreover, since the node v depends on all of the predecessor nodes u_i,

$t_v \geq \max_{\forall u_i \in \text{predecessors of } v} \left(t_{u_i} + \max_{\text{all } P_{u_i v}} \text{len}[P_{u_i v}[\right)$. But $\max_{\text{all } P_{u_i v_{new}}} \text{len}[P_{u_i v_{new}}[= \max_{\text{all } P_{u_i v}} \text{len}[P_{u_i v}[$

and $t_v < t_{v_{new}}$. Hence,

$t_{v_{new}} \geq \max_{\forall u_i \in \text{predecessors of } v} \left(t_{u_i} + \max_{\text{all } P_{u_i v_{new}}} \text{len}[P_{u_i v_{new}}[\right)$, which satisfies the backward precedent relations between v_{new} and the predecessors u_i. Further, since, $t_{v_{new}} \leq \min_{\forall w_i \in \text{successors of } v_{new}} (t_{w_i} - \max_{\text{all } P_{v_{new} w_i}} \text{len}[P_{v_{new} w_i}[ )$, then the forward precedent relations between v_{new} and the nodes w_i are satisfied. Thus, $t_{v_{new}}$ in the specified range satisfies the backward and forward relations and if their exists an idle processor at this time, it must be possible to regenerate v as v_{new} to fire at $t_{v_{new}}$.

THEOREM 2 *Let v be a node regenerated as v_{new} with firing times of t_v and $t_{v_{new}}$, respectively. Let L_v and L_v' be, respectively, the lifetimes of the token produced by v before and after the node regeneration. Then, $L_v - (L_v' + L_{v_{new}}) \geq d_v$, where d_v is the computational delay of v (and therefore, of v_{new}).*

Proof. Let that the node v have k successor nodes w_i. Before applying node regeneration, the node v must provide its token to all of its k successor nodes such that its lifetime L_v is given as

$$L_v = \max_{i=1,\ldots,k} (n_i T + t_{w_i}) - (t_v + d_v) \tag{5.1}$$

However, after the node regeneration, the node v provides the tokens only to those successor nodes w_i for which $(t_{w_i} + n_i T) \leq t_{v_{new}}$. On the other hand, the node v_{new}

provides the tokens to all remaining the successor nodes w_i for which the relation $(t_{w_i} + n_i T) > t_{v_{new}}$ satisfied. Without the loss of generality, assume that the relation $(t_{w_i} + n_i T) \le t_{v_{new}}$ is satisfied for the successor nodes w_1 to w_j where $j < k$. Then,

$$L_{v_{new}} = \max_{i=j+1,\ldots,k} (n_i T + t_{w_i}) - (t_{v_{new}} + d_{v_{new}}) \tag{5.2}$$

$$L'_v = \max_{i=1,\ldots,j} (n_i T + t_{w_i}) - (t_v + d_v) \tag{5.3}$$

Further, using (5.2) and (5.3),

$$L'_v + L_{v_{new}} = \max_{i=1,..j} (n_i T + t_{w_i}) - (t_v + d_v) + \max_{i=j+1,..k} (n_i T + t_{w_i}) - (t_{v_{new}} + d_{v_{new}})$$

However, for $i \le j$, $\max_{i=1,\ldots,j} (n_i T + t_{w_i}) \le t_{v_{new}}$, thus

$$\begin{aligned} L'_v + L_{v_{new}} &\le t_{v_{new}} - (t_v + d_v) + \max_{i=j+1,..k} (n_i T + t_{w_i}) - (t_{v_{new}} + d_{v_{new}}) \\ &\le \max_{i=j+1,..k} (n_i T + t_{w_i}) - (t_v + d_v) - d_{v_{new}} \end{aligned} \tag{5.4}$$

Equation (5.1) can be written as

$$L_v = \max \begin{pmatrix} \max_{i=1,\ldots,j} (n_i T + t_{w_i}) - (t_v + d_v), \\ \max_{i=j+1,\ldots,k} (n_i T + t_{w_i}) - (t_v + d_v) \end{pmatrix} \tag{5.5}$$

Since the first item of the bracketed terms of (5.5) is smaller than the second one, the above equation can be rewritten as $L_v = \max_{i=j+1,\ldots,k} (n_i T + t_{w_i}) - (t_v + d_v)$. Therefore, (5.4) can be rewritten as

$$L'_v + L_{v_{new}} \le L_v - d_{v_{new}} \text{ or } L_v - (L'_v + L_{v_{new}}) \ge d_{v_{new}}.$$

Hence the theorem. □

Based on the above two theorems, it is possible to minimize the lifetime of the token produced by the original node from which the regenerated node was created, which in turn reduces the total number of required registers.

5.3 Interconnect Aware Assignment of Regenerated Nodes to Control Steps and Processing Units

We now provide the proposed methods for interconnect aware scheduling and assignment of the regenerated nodes to, respectively, the control steps and idle processing units in the given scheduled DFG.

5.3.1 Assignment of regenerated nodes to control steps

In the previous sub-section, we determined the candidate nodes to be regenerated as v_{new} and specified the range, given by Theorem 1, of a possible control step as

$$R(v_{new}) = \left[(t_v + d_v),\ min \left(\begin{array}{l} \min\limits_{\forall u_i \in predecessors\ of\ v} (t_{Y_{u_i}} + L_{u_i}), \\ \min\limits_{\forall w_i \in successors\ of\ v_{new}} (t_{w_i} - \max\limits_{all\ P_{v_{new} w_i}} len[P_{v_{new} w_i}[\,) \end{array} \right) \right]$$

We should now perform two tasks in order to include the regenerated nodes into the scheduled DFG:

(i) the selection of a specific control step within its range to fire the regenerated node, and (ii) the assignment of a regenerated node to an idle processing unit. These two tasks can be efficiently performed by using bipartite graphs [93] while taking into consideration the complexity of the interconnects.

In order to perform the first task, a bipartite graph is constructed for each type of nodes. We denote the bipartite graph of type t by BG_t (V_t, C, E), where V_t is the set of

the candidate nodes of type t, C is the set of nodes representing the control steps, and E is the edge set. There is an edge $(v, c) \in E$ if and only if the control step $c \in R(v)$. Fig. 5.2 shows an example of a bipartite graph for a given type of nodes. In this figure, the numbers in the brackets represent the range $R(v)$. Once a bipartite graph is built for each type of regenerated nodes, the scheduling process of the regenerated nodes can be transformed into the one of mapping of the nodes of V_t to the control steps of C. The problem of finding a mapping in the bipartite graph is solved by converting the graph into a flow network and then by using the flow algorithm of [91] to find the maximum flow in the constructed flow network. The flow network is constructed by adding a source node S, a sink node T, and two sets of edges, namely, $\{(S, v_j) | v_j \in V_t\}$ and $\{(c_k, T) | c_k \in C\}$ to the bipartite graph. The advantage of using the flow network is the convenience that it provides in imposing the constraint on the availability of idle processing units, which decides the maximal number of regenerated nodes that can be executed in parallel in each control step. Imposing such a constraint can be easily carried out by setting the capacity of the incoming edges of node T in the flow network according to the number of idle processing units available. The capacity of all other edges in the flow network is set to be 1. Obviously, the maximum flows found under these edge capacities correspond to schedules that always satisfy the resource constraint (i.e., the constraint on the number of idle processing units in a given control step), since the number of regenerated nodes mapped to each control step cannot exceed the capacity of the incoming edges of T. It is to be noted that even if there is an overlap between the ranges of possible control steps $R(v_{new})$, the mapping of the regenerated nodes to control steps would not be affected, since according to Theorem 1, precedent relation between nodes is still satisfied. The

method proposed in [91] to implement the max-flow algorithm is used in this study, which has a complexity of $O(MN \log(N^2/M + 2))$, where M and N are the numbers of edges and vertices, respectively, in the flow network.

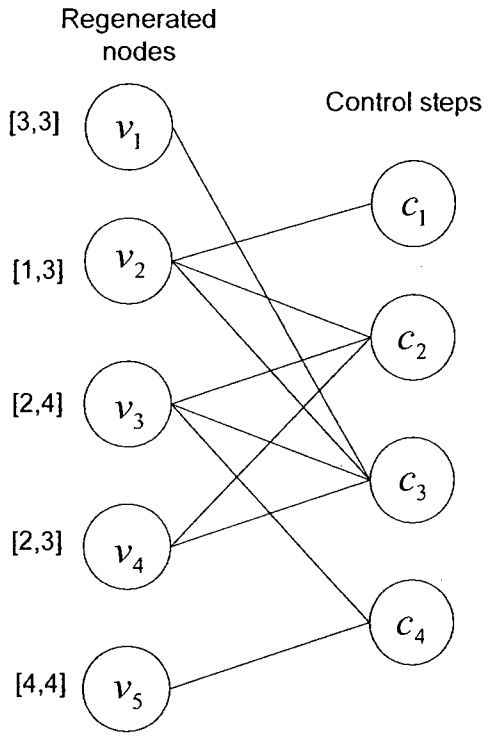


Figure 5. 2: An example of a bipartite graph

Fig. 5.3 shows a flow network constructed from the bipartite graph given in Fig. 5.2. In this example, there are 2 idle processing units available to each of the control steps c_2 and c_3, implying that maximally two regenerated nodes can be executed in parallel in each of the two control steps. On the other hand, there is only one idle processing unit available to each of the control steps c_1 and c_4. Thus, the capacity of each of the edges (c_2, T) and (c_3, T) is set to 2. The capacity of other edges in the flow network is set to 1.

An example of one possible mapping from the regenerated nodes to the control steps found by using the maximum-flow algorithm is shown by using thick edges in Fig. 5.3.

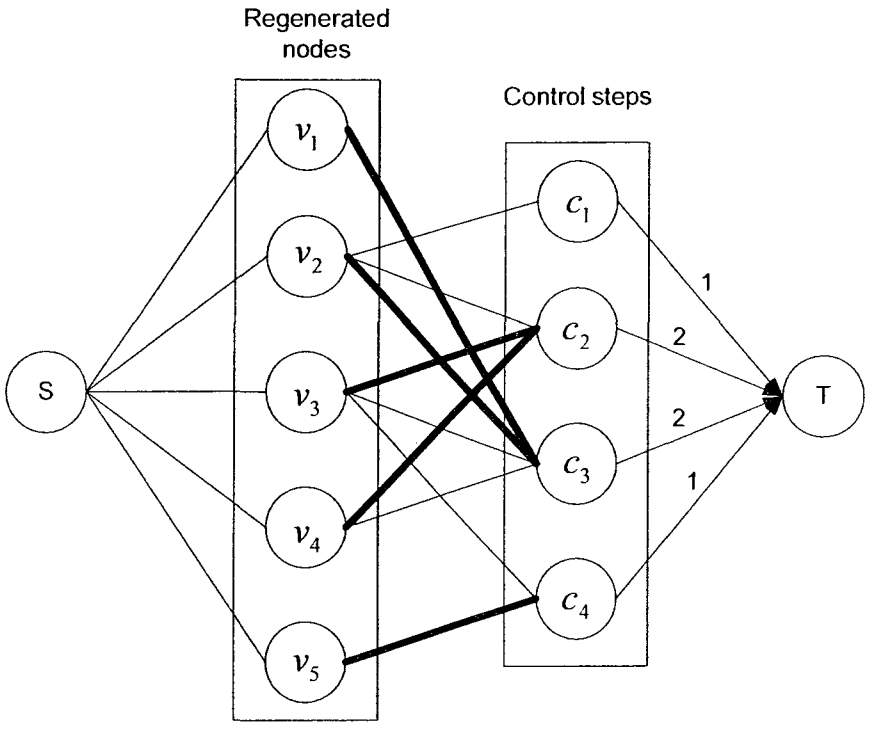


Figure 5. 3: The flow network for the bipartite graph of Fig. 5.2, and a corresponding maximum flow mapping satisfying the constraint on the number of idle processing units.

In the flow network, the maximum flow decides the maximal number of regenerated nodes that can be scheduled so that the available idle processing units are maximally utilized. Since under the constraint of the available number of idle processing units, there exist multiple maximum flows, each corresponding to a feasible schedule in which the idle processing units are maximally utilized, the mapping process itself for obtaining a max-flow could be guided by the considerations that affect the interconnect complexity of the resulting architecture. It is obvious that the higher the number of nodes having common input and outputs assigned to a single processing unit, the less the number of interconnects between the processing unit and the other modules of the architecture. Hence, it is preferred to assign a regenerated node to the idle processing unit

to which maximum number of its predecessor or successor nodes having common inputs or outputs are assigned. In order to take this fact into consideration while scheduling the regenerated node to a control step, a preference is given to the control step that has more number of nodes having input/output common with that of the regenerated node in question assigned to idle processing unit available during the control step. Accordingly, a scheduling-cost *SC* reflecting this preference is assigned to each edge connecting the regenerated node to a control step in the flow network. Thus, the higher the input/output commonality, the less the scheduling cost. The search for a max-flow is carried out by incorporating this scheduling-cost to the mapping process. To this end, two steps have to be performed: 1) a scheduling-cost is formulated to evaluate the preference of scheduling a regenerated node to a particular control step, and 2) the minimum-cost max-flow algorithm [21] is used to carry out this scheduling process. In the proposed method, the scheduling-cost is assigned to the edges of the part of flow network that corresponds to the bipartite graph from which the network was constructed. The costs assigned to the edges connecting the node S to a regenerated node and that assigned to the edge connecting a control step to the terminal node T are set to 0. Consequently, our problem is to find a solution to the minimum-cost maximum-flow problem for the flow network. The cost function to be minimized is given by $\sum_{\forall e_{(v,c)} \in E} SC_{e_{(v,c)}}$, where E is the set of edges connecting the regenerated nodes to the control steps to which the nodes are scheduled.

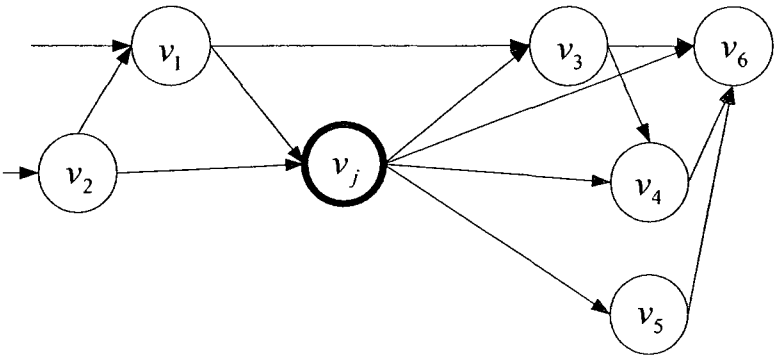
The following is method used to determine the scheduling-cost assigned to each edge connecting a regenerated node to a control step in the flow network.

Calculation of Scheduling-Cost in Flow Network

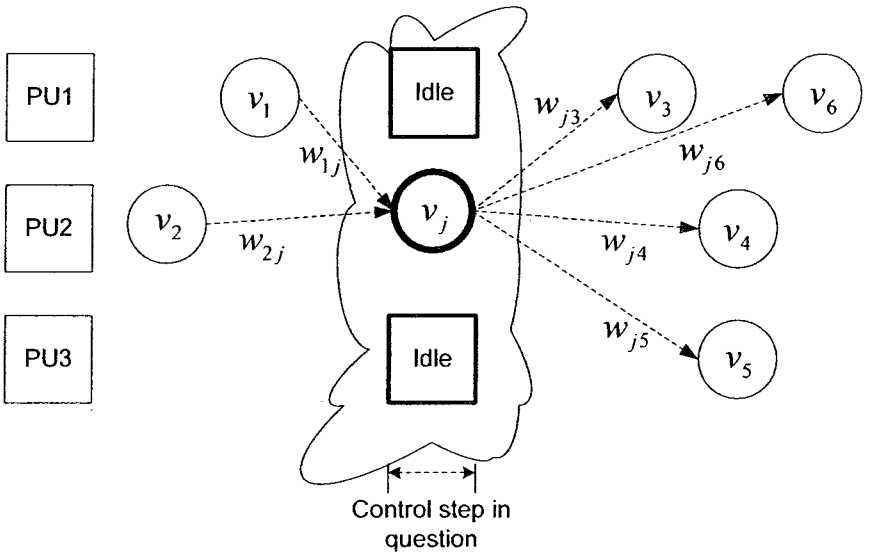
As discussed above, in addition to the capacity associated to each edge connecting a regenerated node to a control step in the flow network, this edge is also assigned a cost representing the scheduling-cost of a possible mapping of the regenerated node to a certain control step. The formulation of such a scheduling-cost to an edge is done as follows. The regenerated node in question is inserted into the scheduled DFG at the targeted control step by introducing an edge connecting the regenerated node to each of its predecessor and successor nodes. The set of predecessor nodes of the regenerated node are the same as those of the original node from which the regenerated node is created. On the other hand, the set of its successor nodes are that subset of the successor nodes of the original node that are scheduled to fire at a control step greater than the targeted control step as discussed in the previous sub-section. It is to be noted that inserting the regenerated node in question into the scheduled DFG is performed without specifying the idle processing unit that will execute it. After performing this insertion process, we associate weights to each of the edges connecting the regenerated node to its successor or predecessor nodes. These weights are formulated as follows. For each edge e connecting the regenerated node to a successor or a predecessor node, we calculated CIO_e as

$$CIO_e = CI + CO + 1 \tag{5.6}$$

where CI and CO are, respectively, the numbers of common inputs and outputs between the regenerated node and the other node (a successor or a predecessor node) of the edge e. Assume that *IDLE* represents the set of idle processing units available during the target control step. Let the regenerated node v_j has successor nodes v_i and predecessor nodes v_k.



(a)



(b)

Figure 5. 4: (a) A DFG containing a regenerated node v_j. (b) The weighted edges connecting v_j with its predecessor and successor nodes.

Then, we calculate the weight associated with the edge in the scheduled DFG connecting the inserted regenerated node v_j to a predecessor node v_i or to a successor node v_k as

$$w_{e_{ij}} = \begin{cases} CIO_{e_{ij}} + \alpha & \text{if } PU(v_i) \cap IDLE \neq 0 \\ CIO_{e_{ij}} & \text{otherwise} \end{cases} \quad (5.7)$$

$$w_{e_{jk}} = \begin{cases} CIO_{e_{jk}} + \beta & \text{if } PU(v_k) \cap IDLE \neq 0 \\ CIO_{e_{jk}} & \text{otherwise} \end{cases} \quad (5.8)$$

where $PU(v)$ is the processing units to which the node v is assigned, and α and β are constants used to increase weight of a particular edge if the edge connects the regenerated node to a predecessor or a successor node assigned to a processing unit that is idle during the targeted control step. We set the values α and β based on the type of the organization of the targeted architecture. Fig. 5.4 gives an example of the calculation of weights associated with the edges in a scheduled DFG connecting an inserted regenerated node v_j with its successor or predecessor nodes. Fig 5.4(a) shows the nodes and edges of the DFG including those resulting from the insertion of the regenerated node at the targeted control step. Fig 5.4(b) shows that there are three processing units in the scheduled DFG, two of them being Idle during the targeted control step. The weights of edges connecting the regenerated node v_j to the other nodes are also shown in Fig. 5.4(b). The weight of each edge is calculated by using (5.6)-(5.8). For example, the edge from v_j to v_3 is associated with the weight $w_{e_{j3}}$. Since v_j and v_3 have 2 common output nodes, namely, v_4 and v_6, CO=2. They also have 1 common input node, namely, v_1, implying that CI =1. Then, according to (13), $CIO_e = 2+1+1=4$. As v_3 is a successor node, we use (5.8)

to find $w_{e_{j3}} = 4 + \beta$. The constant β is added to CIO_e in this particular example, since the processing unit to which the node v_3 is assigned is idle at the targeted control step.

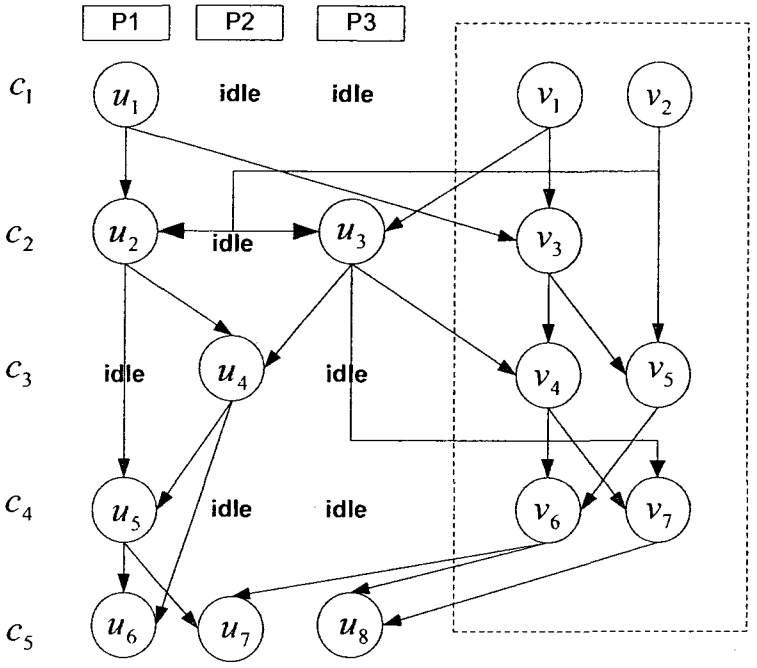
We now determine the scheduling-cost of the edge connecting the regenerated node v_j and a particular control step c in the flow network. After specifying in the scheduled DFG the weight of each edge connecting the regenerated node in question to its predecessor and successor nodes, we now define the quantity U_{v_j} as follows.

$$U_{v_j} = \sum_{\forall e_{jk}} w_{e_{jk}} + \sum_{\forall e_{ij}} w_{e_{ij}} \tag{5.9}$$

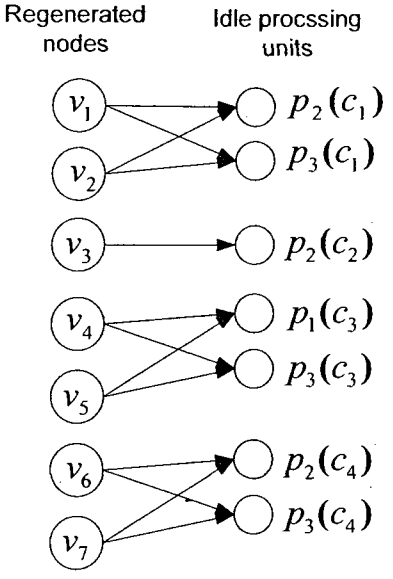
By using (5.9), we determine the weights $SC_{e_{(v_j c)}}$ in the flow network as follows

$$SC_{e_{(v_j c)}} = L - U_{v_j}, \tag{5.10}$$

where $L = \max_{\forall v_j}\left(U_{v_j}\right) + 1$. Finally, in order to schedule the regenerated nodes to specific control steps, min-cost max-flow algorithm [21] is applied to the flow network.



(a)



(b)

Figure 5. 5: (a) A scheduled DFG contains 7 regenerated nodes. (b) The bipartite graph constructed from the scheduled DFG.

5.3.2 Assignment of regenerated nodes to idle processing units

Once each regenerated node has been assigned to a specific control step by applying min-cost max-flow algorithm on the flow network, we next perform the second task, i.e., the assignment of the regenerated nodes to the idle processing units. A bipartite graph, showing all possible assignments of each regenerated node to idle processing units, is constructed. We denote the bipartite graph of type t by BG_t (V_t, P, E), where V_t is the set of regenerated nodes of type t (for example, addition or multiplication), P is the set of idle processing units, and E is the edge set. There is an edge $(v_i, p_j(c_k)) \in E$, if and only if the node v_i is scheduled at c_k and the processing unit p_j is idle during the control step c_k. Fig. 5.5 gives an example for the construction of the bipartite graph from a scheduled DFG containing the regenerated nodes. Fig. 5.5(a) shows a part of a scheduled DFG containing 7 regenerated nodes denoted as v_i, $i = 1, 2, \ldots, 7$. There are three processing units used in the scheduled DFG. In addition to the nodes u_i originally assigned to the three processing units, each regenerated node v_i has to be assigned to one idle processing unit available during the control step at which the regenerated node is scheduled. The shaded part of the scheduled DFG shown in Fig. 5.5(a) contains the regenerated nodes to be assigned to idle processing units as well as the data flow edges connecting each regenerated node with all other nodes in the DFG. The assignment of a regenerated node is carried out taking into consideration the impact that this assignment would have on the interconnect complexity of the targeted architecture. In order to explore this impact, the assignment of other regenerated nodes having flow dependency with the regenerated node should also be taken into consideration. For this purpose, the concept of node

compatibility is incorporated. The resulting bipartite graph is then used to solve the problem of assignment of the regenerated nodes to the idle processing units. Before proceeding further in presenting the proposed assignment scheme, let us define some terms that are required in the proposed scheme. We say that a pair of regenerated nodes are compatible if only if: (a) they are scheduled at two different control steps, (b) there is a single idle processing unit capable of performing both of them, i.e., a processing unit idle during the two control steps at which they are scheduled, and (c) they have data flow dependency. Nodes v_1 and v_3 given in Fig. 5.5 are compatible, since they satisfy the three compatibility conditions: they are scheduled to two different control steps c_1 and c_3, there is a processing unit, namely P_2, idle during c_1 and c_3, and they have data flow dependency, i.e., node v_3 consumes the output of v_1. Other pairs of compatible nodes in Fig 5.5(a) are $(v_2 \rightarrow v_5)$, $(v_4 \rightarrow v_6)$, $(v_4 \rightarrow v_7)$, $(v_5 \rightarrow v_7)$. We also define the compatible path as a link-list set of compatible nodes $(v_1 \rightarrow v_2 \rightarrow \cdots v_{i-1} \rightarrow v_i \rightarrow v_{i+1})$ such that v_{i-1} is compatible with v_i and v_i is compatible with v_{i+1}. The set of nodes in a compatible path are ordered according to the control steps at which they are scheduled, i.e., $c(v_{i-1}) < c(v_i)$, where $c(v_i)$ represents the control step at which the node v_i is scheduled.

We now proceed in presenting the proposed scheme for the assignment of regenerated nodes to the idle processing units. Since multiplexers connects multiple inputs to a processing unit or to a register in the targeted architecture, the multiplexer count is an efficient indicator of the interconnect complexity [79]. In this regard, the more the nodes having data flow dependency assigned to a single processing unit, the less the number of different inputs or outputs to or from the processing units required, and hence,

the less the number of multiplexers. Accordingly, since compatible nodes have flow dependency and they can be executed on a single processing unit, we modify the bipartite graph by introducing in it a mapping that represents a possible assignment of a set of link listed compatible nodes to a processing unit. In the bipartite graph, each edge connecting a regenerated node to a processing unit is replaced, whenever possible, by a path connecting the link list of compatible nodes. Such path is called an assignment compatible path. An assignment compatible path is denoted as $v_1 \rightarrow v_2 \rightarrow \cdots v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow p_j(c(v_1), c(v_2), \ldots c(v_{i-1}), c(v_i), c(v_{i+1}))$.

The process of generating the assignment compatible paths for the bipartite graph is carried out as follows. Assume that v_i is a regenerated node that can be assigned to an idle processing unit p_j during the control step $c(v_i)$. This implies that there is an edge $v_i \rightarrow p_j(c(v_i))$ in the bipartite graph. Further, assume that v_i is compatible with v_{i+1}, which in turn is compatible with v_{i+2}, such that $c(v_i) < c(v_{i+1}) < c(v_{i+2})$. Then, the edge $v_i \rightarrow p_j(c(v_i))$ in the bipartite graph is replaced by the path $v_i \rightarrow v_{i+1} \rightarrow p_j(c(v_i), c(v_{i+1}))$ to include the regenerated node v_{i+1}, and then once again replaced by $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} \rightarrow p_j(c(v_i), c(v_{i+1}), c(v_{i+2}))$ to include v_{i+2} resulting in an assignment compatible path. After introducing an assignment compatible path, we remove any redundant edge from the resulting bipartite graph. We say an edge $v_i \rightarrow p_j(c(v_i))$ is redundant, if and only if it is contained in the assignment compatible path in question. For example, if we have $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} \rightarrow p_j(c(v_i), c(v_{i+1}), c(v_{i+2}))$ as an assignment compatible path, the original edge $v_i \rightarrow p_j(c(v_i))$ of the bipartite graph is removed since

it is contained in the assignment compatible path. Note that $p_j(c(v_i)) \cup p_j(c(v_{i+1})) \cup p_j(c(v_{i+2})) = p_j(c(v_i), c(v_{i+1}), c(v_{i+2}))$. The resulting bipartite graph, in which all possible assignment compatible paths have been introduced and any redundant edge removed is called the modified bipartite graph. Note that the modified bipartite graph may still contain some original edges of the bipartite graph when a regenerated node is not compatible to any other regenerated node with respect to a given processing unit. Fig. 5.6(a) depicts all possible assignment compatible paths (indicated by thick edges) and all the redundant edges (indicated by crossed thin edges), corresponding to the bipartite graph of Fig. 5.5(b). It is seen from this figure that 3 new assignment compatible paths are introduced. The path, for example, $v_1 \rightarrow v_3 \rightarrow p_2(c_1, c_2)$ is introduced in the bipartite graph, since the nodes v_1 and v_3 are compatible. The corresponding edge $v_3 \rightarrow p_2(c_2)$ is redundant. The removal of the redundant edges from the graph of Fig. 5.6(a) gives rise to the modified bipartite graph as shown in Fig. 5.6(b).

We use the modified bipartite graph to perform the assignment of the regenerated nodes to processing units. It is to be noted that one or more than one node can be found in more than one assignment compatible path. The assignment compatible paths differ from one another in terms of the impact that a path would make on the interconnect complexity depending on the set of compatible nodes that belong to a path. The more the number of inputs and outputs that is common to the nodes belonging to an assignment compatible path or common to these nodes and the nodes previously assigned to the processing unit in question, the less the number of interconnects in the resulting architecture. We assign a weight to each edge in an assignment compatible path as follows. A weight is assigned to

each edge, except the last one, of a path as the sum of the numbers of inputs and outputs common to the two nodes of all the pairs of nodes, where each pair is formed by the initial node of the edge and a node from the link list of the regenerated nodes that follow the initial node of the edge in the assignment compatible path. For the last edge in an assignment compatible path, a weight is assigned as the number of edges in the path plus the sum of numbers of inputs and outputs common to the two nodes of a pair of nodes, where the set of pairs of nodes consist of the pairs in which one node is taken from the set of regenerated nodes in the path and the other node of the pair is taken from the nodes assigned to the processing unit of the path in question, other than those belonging to the path. The formulation of the assignment of weights to the edges of the assignment compatible paths just describe is carried out as follows

Assume that

$$v_1 \rightarrow v_2 \rightarrow \ldots v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots v_n \rightarrow p_j(c(v_1), c(v_2), \ldots, c(v_{i-1}), c(v_i), c(v_{i+1}) \ldots c(v_n))$$

is an assignment compatible path in a modified bipartite graph. The weight assigned to an edge $v_i \rightarrow v_{i+1}$ is given as

$$w_{(v_i, v_{i+1})} = \sum_{j=i+1}^{n} (CO_{(v_i, v_j)} + CI_{(v_i, v_j)}) \tag{5.11}$$

where $CI_{(v_i, v_j)}$ and $CO_{(v_i, v_j)}$ are, respectively, the number of inputs and outputs common to nodes v_i and v_j. Assume that there is a set of nodes v'_k, $k = 1, 2, \ldots, m$, previously assigned to the processing unit p_j. The weight assigned to the last edge of the assignment compatible path, i.e., the edge $v_n \rightarrow p_j(c(v_1), c(v_2), \ldots, c(v_{i-1}), c(v_i), c(v_{i+1}) \ldots c(v_n))$, is given as

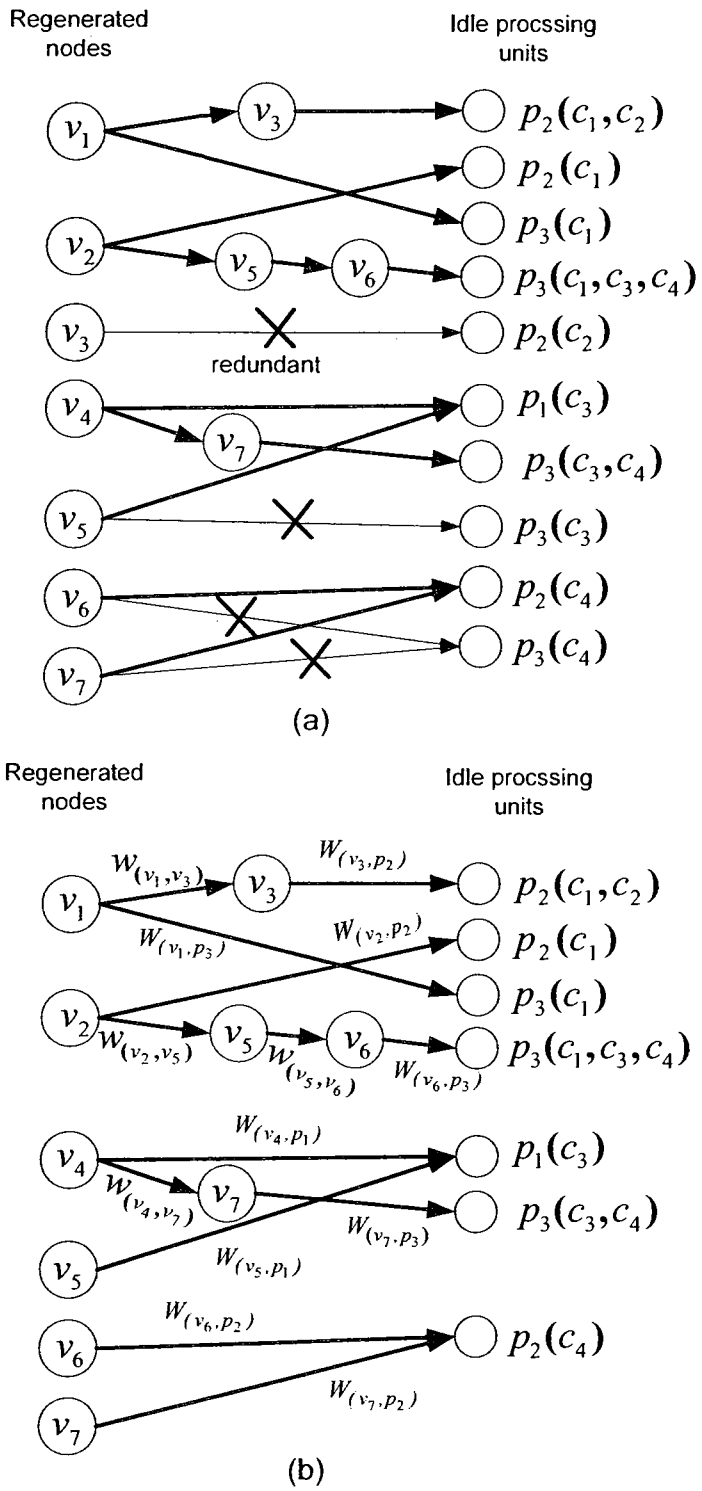


Figure 5. 6: (a) Assignment compatible paths introduced into the bipartite graph given in Fig 5.5b. (b) Weighted paths used during the assignment process.

$$W_{(v_n,p_j)} = \left[\sum_{i=1}^{n}\sum_{k=1}^{m}(CI_{(v_i,v'_k)} + CO_{(v_i,v'_k)})\right] + (n-1) \qquad (5.12)$$

where, as mentioned earlier, node v_i belongs to the set of regenerated nodes in the assignment compatible path, and v'_k belongs to the set of nodes assigned to the processing unit of the path in question. In Fig 5.6(b) the weights assigned to the edges of the modified bipartite graph are also indicated, where each of the weights is determined using (5.11) or (5.12).

After assigning weights to each edge in the modified bipartite graph, we determine the length of each assignment compatible path as the sum of the weights of all the edges in the path. The longest path is chosen first for the assignment and its entire link list of nodes are assigned to the processing unit terminating the path, i.e., the link list of nodes ($v_1, v_2, \ldots v_{i-1}, v_i, v_{i+1}, \ldots, v_n$) is assigned to

$p_j(c(v_1), c(v_2), \ldots, c(v_{i-1}), c(v_i), c(v_{i+1}) \ldots c(v_n))$ if the path $v_1 \rightarrow v_2 \rightarrow \ldots v_{i-1} \rightarrow v_i \rightarrow$

$v_{i+1} \rightarrow \ldots v_n \rightarrow p_j(c(v_1), c(v_2), \ldots, c(v_{i-1}), c(v_i), c(v_{i+1}) \ldots c(v_n))$ has the longest length. Assuming that this is the longest one, and therefore, the nodes $v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n$ are assigned to p_j, these nodes and their outgoing edges are removed from the modified bipartite graph.

Assigning the link list of nodes $v_1, v_2, \ldots v_{i-1}, v_i, v_{i+1}, \ldots, v_n$ to the processing unit p_j at the control steps $c(v_1), c(v_2), \ldots, c(v_{i-1}), c(v_i), c(v_{i+1}) \ldots c(v_n)$, respectively, implies that this processing unit is no longer available for assignment of other nodes at these specific control steps. Consequently, in the modified bipartite graph, if there is another

node representing the same processing unit p_j with a set of control steps that is a subset of the set of control steps $c(v_1), c(v_2),\ldots,c(v_{i-1}), c(v_i), c(v_{i+1})\ldots c(v_n)$, then this node is also no longer available at its set of control steps. We call such node representing the processing unit p_j at the associated control steps as fully occupied. On the other hand, if there is a node in the modified bipartite graph representing the processing unit p_j with a set of control steps such that only a subset of this set is also a subset of the control steps $c(v_1), c(v_2),\ldots,c(v_{i-1}), c(v_i), c(v_{i+1})\ldots c(v_n)$, then the processing unit p_j is no longer available at the control steps of this subset, but it is still available at the remaining control steps of the set of control steps associated with this node of p_j. We, therefore, call such a node representing the processing unit p_j and the associated set of control steps as partially occupied. In the modified bipartite graph, the entire nodes and edges of an assignment compatible path terminated by to a fully occupied node are removed except those nodes and edges that are shared between the path in question and some other assignment compatible paths. On the other hand, the entire nodes and edges of an assignment compatible path terminated by a partially occupied node are first removed except those nodes and edges that are shared between the path in question and some other assignment compatible paths, and then the set of nodes in the removed path, excluding the nodes scheduled at control steps responsible for making the terminating mode of the path partially occupied, are used for constructing, as necessary, one or more new assignment compatible paths in the modifies bipartite graph using the remaining link list of the removed path. As an example, assume that $v_1' \rightarrow v_2' \rightarrow v_3' \rightarrow v_4' \rightarrow v_5' \rightarrow v_6' \rightarrow v_7' \rightarrow \ldots v_m' \rightarrow$

$p_j(c(v_1'), c(v_2'), c(v_3'), c(v_4'), c(v_5'), c(v_6'), c(v_7'), \ldots, c(v_m'))$ is an assignment compatible path in which the node p_j is partially occupied such that it is no longer available at the control steps $c(v_3')$ and $c(v_6')$. Consequently, this assignment compatible path is removed, and 3 new assignments compatible paths $v_1' \rightarrow v_2' \rightarrow p_j(c(v_1'), c(v_2'))$, $v_4' \rightarrow v_5' \rightarrow p_j(c(v_4'), c(v_5'))$, and $v_7' \rightarrow \ldots v_m' \rightarrow p_j(c(v_7'), \ldots, c(v_m'))$ are added in the modified bipartite graph.

The removal of the longest assignment compatible path, the removal of other nodes and edges resulting from the removal of the longest assignment compatible path and the addition of new assignment compatible paths constitute one iteration of regenerated nodes assignment and results in a reduced modified bipartite graph. The weights of the edges in the reduced modified bipartite graph are updated using (5.11) and (5.12). The process is repeated using the reduced modified bipartite graph until all regenerated nodes are assigned.

5.4 Experimental Results

In our experiments, both centrally shared and distributed-register based architectures are targeted. The proposed scheme is first assessed in terms of the number of registers and the number of interconnects required when the node regeneration is incorporated in the register binding technique proposed in the previous chapter to carry out the synthesis of the some benchmark examples targeting both the centrally-shared- and distributed-register based architecture and the corresponding results are compared with those obtained by using the proposed register binding technique without incorporation of the

node regeneration scheme Finally, experiments are carried out to determine the total interconnect lengths in the RTL structures obtained through the synthesis of some DSP problems with and without the incorporation of the node regeneration scheme in the proposed register binding technique.

5.4.1 Number of registers and interconnects with node regeneration

In order to assess the effect of incorporating the node regeneration scheme into the proposed register binding technique, the technique is applied to the same set of DSP benchmark problems as in Section 4.3. Fig. 5.7 gives the number of registers obtained with and without the incorporation of node regeneration scheme to the proposed register binding technique for targeting centrally-shared and distributed register-based architectures. It is seen from this figure that the node regeneration reduces the number of registers in both types of architectures. The proposed register binding technique offers a substantial gain in terms of reducing the number of registers when it incorporates the node regeneration. In centrally-shared register-based architecture, an average reduction of 16.13% is achieved for the intensive DSP benchmarks considered with the minimum reduction being 12.7% for *DCT-chem* and the maximum 21.87% for *mcm*. In the distributed architecture, the average reduction is 13% with the minimum and the maximum reduction being 6.25% and 15.47% for the benchmarks of *DCT-planar* and *DCT-chem,* respectively.

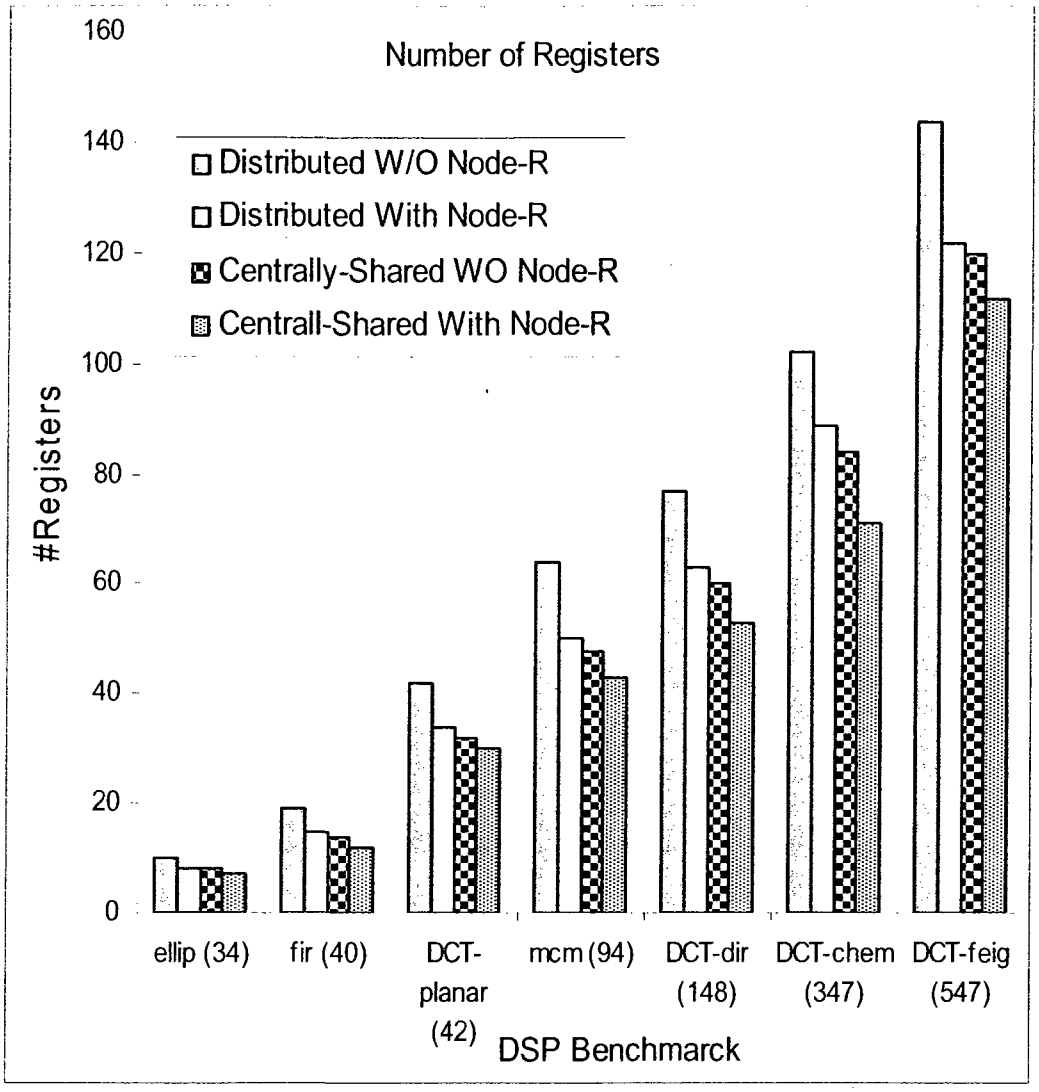


Figure 5. 7: Number of registers obtained by applying the proposed technique for the register binding of some intensive DSP Benchmark problems with and without node regeneration targeting both the centrally-shared and the distributed register-based architectures.

Fig. 5.8 gives the corresponding results in terms of the number of interconnects. Again we see that the incorporation of the node regeneration scheme to the proposed register binding technique reduces the number of interconnects for both types of architectures. In centrally-shared register-based architecture, an average reduction of 8.89% is achieved

for the intensive DSP benchmarks considered with the minimum reduction being 5% for *ellip* and the maximum 13.3% for *fir*. In distributed one, the average reduction is 19.56% with the minimum and the maximum reduction being 16.36% and 24% for the benchmarks of *mcm* and *fir,* respectively.

5.4.2 Overall length of interconnects for the RTL structures of some DSP benchmarks.

Interconnect lengths have become a dominant factor in the design of integrated circuits. The parasitics associated with length of interconnects account for a significant part of the noise, delay and power associated with a signal [77]. In order to show as to how reducing the number of interconnects obtained by the incorporation of the node regeneration scheme into the proposed register binding results in reducing the total wire length, we generate the layouts of the RTL structures of some of the DSP benchmark problems using Cadence's Silicon Ensemble and measure the total length of the interconnects. The DSP benchmarks chosen are 8-tap FIR filter, 8-tap IIR filter, and 1-point 8X8 DCT filter. For each benchmark, two RTL structures, one targeting the centrally-shared register based architecture and the other the distributed registered based architecture, are obtained.

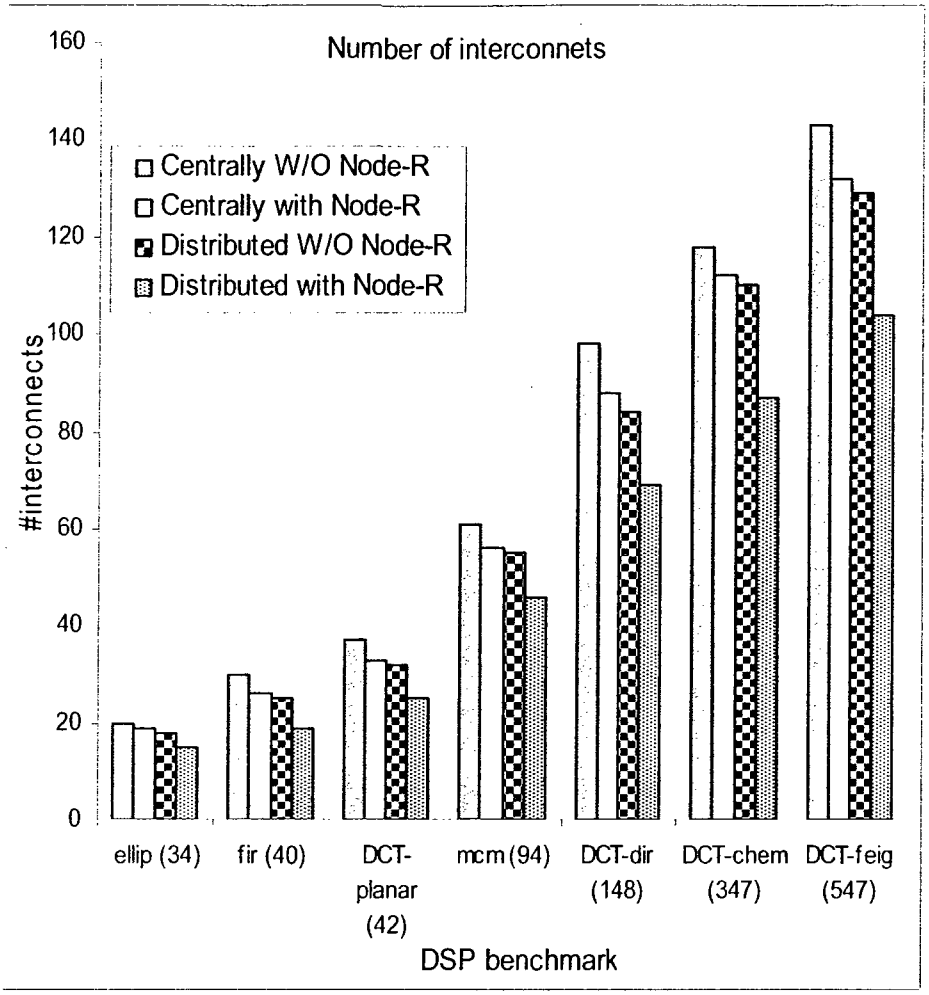


Figure 5. 8: Number of Interconnects obtained by applying the proposed technique for the register binding of some intensive DSP Benchmark problems with and without node regeneration targeting both the centrally-shared and distributed register-based architectures.

Table 5.1 lists the resources of the synthesized RTL structures and the resulting wire length for each of the benchmarks considered with and without the incorporation of node regeneration scheme. The RTL resources are specified in terms of the number of the multipliers, adders, and registers. Even though the proposed register binding technique and the node regeneration schemes are not concerned with the total number of interconnects in all the tasks of the high level synthesis globally, it is obvious from this

table that they help in reducing the total wire length in the RTL structures. For the benchmark problems considered, the average reduction in the total wire length is 10.83% for the centrally-shared register based architecture and it is 11.18% for the distributed one.

Table 5. 1: Comparison of the total wire length with and without the incorporation of node regeneration scheme

Benchmark	Type of architecture	RTL resources		Total wire-length (μm)		Reduction
		Without node regeneration	With node regeneration	Without node regeneration	With node regeneration	
FIR	Centrally-shared	2M, 2A, 14R	2M, 2A, 11R	139287	122771	10.80 %
	Distributed	2M, 2A, 21R	2M, 2A, 15R	111020	98953	11.85 %
IIR	Centrally-shared	4M, 2A, 13R	4M, 2A, 10R	173271	151480	12.50 %
	Distributed	4M, 2A, 17R	4M, 2A, 14R	160661	146392	14.20 %
DCT	Centrally-shared	4M, 4A, 27R	4M, 4A, 22R	509463	462420	9.20%
	Distributed	4M, 4A, 36R	4M, 4A, 28R	480860	444563	7.50%

5.5 Summary

The lower bound on the number of registers resulting from any register binding technique gets fixed once the DFG has been scheduled. In this paper, a scheme, referred to as node regeneration, has been proposed to reduce the number of registers to a value that is even lower than this bound. This scheme by utilizing the idle processing units generates multiple copies of the nodes in the original scheduled DFG with the lifetimes of the tokens in the modified DFG to be shorter than that of tokens of the nodes in the original scheduled DFG. In essence, the scheme reduces the number of registers by having at its

disposal a modified scheduled DFG without re-scheduling the DFG or adding additional resources. The freedom provided by having multiple copies of nodes has also been exploited to the idle processing units to minimize the complexity of the interconnect network. In order to assess the proposed technique of register binding, it has been applied to the synthesis targeting centrally-shared and distributed register based architectures for different intensive DSP algorithms and has been compared with various other commonly used synthesis methods for register binding. The reductions in terms of the number of registers and the number of interconnects are even more substantial when the proposed node regeneration scheme is incorporated in the register binding technique. Finally, it has been shown that the reduction in the number of interconnect in fact results in reducing the total wire length of the layout of the RTL structures.

Chapter 6

Conclusion

6.1 Concluding Remarks

With the technology scaling down into a deep submicron region, the high level synthesis tasks can no longer afford to be performed without taking into account the impact of the interconnects on the performance of the resulting architectures. A realistic model to be used in the high level synthesis should, for instance, support interprocessor communication delays, different types of processing units, and the structure and organization of the data path. This doctoral thesis has been concerned with the problem of developing efficient interconnect aware techniques for the high-level synthesis of DSP applications leading to the implementations with parallel processing architectures. Under this common theme, the thesis has two distinct focuses. The first focus has been on developing new techniques for scheduling and processor allocation while taking into

consideration the interprocessor communication delay. To this end, two techniques have been proposed. In the first technique, the interprocessor communication delay used in the tasks of scheduling and processor allocation has been estimated or taken from an already placed architecture. While in the second technique, a placement process has been integrated into the high level synthesis to consider the impact of the positions of the processing units in the placement space on the building of the time and processor schedules. The second focus of this thesis has been on developing a technique to carry out the register binding while taking into consideration the complexity of the interconnects. Since the lower bound on the number of registers resulting from any register binding technique gets fixed once the DFG is scheduled, a node regeneration scheme has been proposed to reduce the number of registers to a value that is even lower than this bound and at the same time to lower the interconnect complexity.

A technique for the synthesis of DSP cyclic data flow graphs onto heterogeneous distributed-register based multiprocessor architectures employing a graph theoretic approach has been proposed. The interprocessor communication delay has been assumed to be taken from estimation or from feedback information from a placement. The proposed technique starts by modifying the original DFG representing a DSP algorithm by inserting dummy communication nodes to represent the ICDs between the nodes of different types. The modified DFG is then used to build iteratively a time schedule based on the mobility of each node. An algorithm has been proposed to identify each critical or near-critical loop in the modified DFG. Next, by employing the initial time schedule and by using the loop identification algorithm, the task of an initial processor allocation is carried out. Since, the initial time schedule and processor allocation does not take into

account the ICDs of the nodes of the same type, the initial time and processor schedules may not be valid. Hence, the initial time and processor schedule have been modified to take into account the ICDs between a pair of nodes of the same type assigned to two different processors in order to find the final time and processor schedule. This modification has been carried out by inserting additional cycles into the time schedule in order to ensure on the validity of the ICDs between a pair of nodes of the same type. In order to assess the proposed synthesis technique, it has been applied to the synthesis of different DSP digital filters and has been compared with various other commonly used synthesis techniques. It has been shown that the proposed synthesis technique outperforms these techniques in terms of the iteration period and the numbers of processors of the synthesized architecture.

Existing synthesis tools perform the high level synthesis tasks and the tasks of physical design such as placement independently. A technique for the integration of the placement process into the high-level architectural synthesis has been developed in away so that information about the position of the processing units in the placement space and about interconnect delays are used during the building of time schedule and the allocation of the nodes. A systematic process has been employed for the placement by using a Delaunay triangular mesh in the proposed scheme, since this method of triangulation can make candidate positions of the processing units well-distributed. This triangulation method maximizes the minimum angles of the mesh, and, hence, it is then became possible to avoid making candidate positions for placing the processing units on adjacent edges connected by a narrow angle which in turn allows to, quickly, find the suitable gaps to place the functional units in the placement space. In order to maximize the local

data transfers, a hybrid library of functional unit, which includes dynamically reconfigurable multiple-operation and operation-specific functional units, have been used in the proposed technique. Moreover, the use of hybrid library provides the designer of DSP applications with a greater flexibility to explore the design space. The proposed technique has been applied to well-known benchmark problems of DSP applications in order to assess the effectiveness of the interaction between the high level synthesis and the placement process. During the experiments, the placement has performed under two assumptions about the number of candidate position allowed to place the processing units. In one case, a restricted number of candidate positions is used. While in the other case, the number of the candidate positions has been increased in order to provide more flexibility to the placement process. It has been shown that a substantial gain in terms of reducing both the placement area as well as the iteration has obtained period for all the benchmark problems considered when the high level synthesis interacts with a flexible placement compared to a restricted placement process.

An interconnect aware register binding has been proposed. The function of register binding is to assign the tokens produced by the processing units to registers in the resulting RTL structure and it is traditionally done in a way so as to minimize their number. A technique for binding the tokens produced by the nodes of a scheduled DFG while aiming at minimizing the number of interconnects has been presented. First, a segmentation scheme in which the lifetime of a token is appropriately divided into multiple segments is developed. Then, the register binding problem is formulated as a min-cost flow problem so that the tokens having the same source and/or destination are bound into the same register and results in a reduced numbers of registers and

interconnects. In order to assess the proposed technique of register binding, it has been applied to the synthesis targeting centrally-shared and distributed register based architectures for different intensive DSP algorithms and has been compared with various other commonly used synthesis methods for register binding. The results of these experiments have shown that the proposed register binding technique produces the number of registers equal to the optimal solution provided by using the left-edge method and it outperforms other methods not only in terms of the number of registers but also in terms of the number of interconnects.

The lower bound on the number of registers resulting from any register binding technique gets fixed once the DFG has been scheduled. A scheme, referred to as node regeneration, has been proposed to reduce the number of registers to a value that is even lower than this bound. This scheme by utilizing the idle processing units generates multiple copies of the nodes in the original scheduled DFG with the lifetimes of the tokens in the modified DFG to be shorter than that of tokens of the nodes in the original scheduled DFG. In essence, the scheme reduces the number of registers by having at its disposal a modified scheduled DFG without re-scheduling the DFG or adding additional resources. The freedom provided by having multiple copies of nodes has also been exploited to the idle processing units to minimize the complexity of the interconnect network. In order to assess the proposed scheme, it is incorporated with proposed register binding technique and it has then been applied to the synthesis targeting centrally-shared and distributed register based architectures for different intensive DSP algorithms and has been compared with various other commonly used synthesis methods for register binding. It has been shown that the reductions in the number of registers and interconnects are

even more substantial when the proposed node regeneration scheme is incorporated in the register binding technique. Finally, it has been shown that the reduction in the number of interconnect by the proposed register binding technique and the node regeneration scheme, in fact, results in reducing the total wire length of the layout of the RTL structures.

6.2 Directions for Future Research

The research work undertaken in this thesis can be extended in several respects. One interesting area of investigation would be the development of high level synthesis techniques while taking into consideration other performance metrics for the interconnects such as power and area. In this thesis, a technique for the integration of the placement process into the scheduling and allocation tasks of the high level synthesis has been proposed. To the best of the auther's knowledge, there is no reported work in the literature in which the routing is incorporated into the process of high level synthesis. Due to the impact of the routing on the interconnect network of an architecture, it would worth exploring the integration of interconnect routing into the high level synthesis itself.

Another interesting area of investigation could be the development of techniques for high level synthesis targeting the newly advanced 3D integrated technologies which offer a great promise in providing improvements in the overall circuit performance. The 3D integrated circuits are fabricated by stacking multiple active device layers using wafer bonding with vertical interconnects for inter-layer communication. This recent progress in the fabrication of 3D integrated circuits has opened up the possibility of exploiting this technology to alleviate the problems related to power and interconnect delays resulting

from the deep-submicron technology. Hence, methodologies and techniques need to be developed to address the scheduling and resources allocation during high level synthesis for vertically integrated 3D systems. Also, the problem of integration of 3D-placemet into the high level synthesis must be dealt with in order to determine the location of individual processing units on a 3D placement and to identify the intra-layer and inter-layer interconnect delays.

References

[1] D.J. DeFatta, J.J. Lucas, W.S. Hadgkiss. Digital signal processing, a system design approach. John Wiley & Sons; 1988.

[2] M.C. McFarland, A.C. Parker, and R. Camposano, "The high level synthesis of digital systems," in *Proc. IEEE International Symposium on Circuits and Systems,* vol. 78, no.2, February 1990, pp. 301-318.

[3] A. Itradat, M.O. Ahmad, A. Shatnawi, "Scheduling of DSP data flow graphs with processing times characterized by fuzzy sets," in *Proc. Canadian Conference on Electrical and Computer Engineering 2004* , CCECE 2004, Niagara Falls, Canada, May 2004, pp. 1245 – 1248.

[4] G. Pierre and J.P. Knight, ``Force Directed Scheduling for the behavioural synthesis of ASICs," *IEEE Trans. on Computer Aided Design*, vol. 8, pp. 661-679, June 1989.

[5] A. Shatnawi, M.O. Ahmad, and M.N.S. Swamy, "Scheduling of DSP data flow graphs onto multiprocessors for maximum throughput," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 6, May 1999, pp. 386-389.

[6] C. Tseng and D.P. Siewiorek., "Automated Synthesis of data paths in digital systems," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems,* vol. 5, pp. 379-395, July 1986.

[7] C. Tseng and D.P. Siewoirek, ``Automated synthesis of data paths in digital systems," *IEEE Trans. on Computer Aided Design*, vol. 5, pp. 379-295, July 1986.

[8] C.H. Genotys and M.I. Elmasry, " VLSI design synthesis with testability," in *Proc of the 25th ACM/IEEE Conference on Design Automation*, New Jersey, United States, June 1988, pp.16-21.

[9] P. Marwedel, "A new synthesis algorithm for the MIMOLA software system," in *Proc. 23rd Design Automation Conference,* Las Vegas, NV, July 1986, , pp. 271-277.

[10] Z. Shao, Q. Zhuge, C. Xue, and E.H.-M. Sha, "Efficient assignment and scheduling for heterogeneous DSP systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 6, pp. 516 – 525, June 2005.

[11] A.M. Sllame and V. Drabek, "An Efficient list-based scheduling algorithm for high-level synthesis," *in Proc. IEEE EUROMICRO Symp. on Digital System Design: Architecture, Methods and Tools Conference DSD'2002*, Germany, September 2002, pp.316-323.

[12] S.M. Heemstra de Groot, S. H. Gerez, and O. E. Henmann, "Range chart-guided iterative data-flow-graph scheduling," *IEEE Trans. Circuits Syst.-I:*, vol. 39, no. 5, pp. 351-364, May 1992.

[13] B.A. Curtis and V.K. Madisetti, "Rapid prototyping on the Georgia Tech digital signal multiprocessor," *IEEE Trans. Signal Processing*, vol. 42, no. 3, pp. 649-662, March 1994.

[14] S. Tongsima, E.H.-M. Sha, and N.L. Passos, "Communication sensitive loop scheduling for DSP applications," *IEEE Trans. Signal Processing*, vol. 45, no. 5, pp. 1309-1322, May 1997.

[15] K. Ito, T. Iwata, and H. Kunieda, "An optimal scheduling method for parallel processing system of array architecture," in *Proc. Asia and South Pacific Design Automation Conference, ASPDAC '97*, 1997, pp. 447-454.

[16] S.L. Sindorf and S.H. Gerez, "An integer linear programming approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays," in *Proc. PROGRESS 2000 Workshop on Embedded Sys.*, Utrecht, The Netherlands, October 2000, pp. 1-8.

[17] H.-Y. Tyan and Y.H. Hu, "Minimum initiation interval of multi-module recurrent signal processing algorithm realization with fixed communication delay," in *Proc. IEEE Acoustics, Speech, and Signal Processing Conference, ICASSP '99,* vol. 4, 1999, pp. 2005- 2008.

[18] P.G. Paulin and J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's,'' *IEEE Trans. Computer-Aided Design,* vol. 8, no. 6, pp. 661-679, June 1989.

[19] C. T. Hwang, J. H. Lee, Y. C. Hsu, and Y. L. Lin, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. Computer- Aided Design,* vol. 10, no. 4, pp. 464-475, April. 1991.

[20] C. H. Gebotys and M. I. Elmasry, "Global optimization approach for architectural synthesis," *IEEE Trans. Computer-Aided Design,* vol. 12, no. 9, pp. 1266-1278, Sept. 1993.

[21] A. Sharma and R. Jain, "InSyn: integrated scheduling for DSP applications," *IEEE Trans Signal Processing*, vol. 43, no. 8, pp. 1966-1977, Aug. 1995.

[22] C.-Y. Wang and K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, and allocation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Syst.,* vol. 14, no. 3, pp. 274–295, March 1995.

[23] K. Ito, and K. Parhi, "A generalized technique for register counting and its application to cost-optimal DSP architecture synthesis," *Journal of VLSI Signal Processing,* vol. 16, pp. 57–72, 1997.

[24] S. H. Gerez, S. M. Heemstra de Groot, and O. E. Henmann, "A polynomial time algorithm for the computation of the iteration-period bound in recursive data flow graphs," *IEEE Trans. Circuits Syst.-I,* vol. 39, no. I, pp. 49-52, Jan. 1992.

[25] A. Itradat, M.O. Ahmad, A. Shatnawi, "High-level architectural synthesis of dsp data flow graphs with inter-processor communication delays," under review for publication in *IET Journal on Circuits, Devices.*

[26] A. Itradat, M.O. Ahmad, A. Shatnawi, "A delay-optimal static scheduling of DSP applications mapped onto multiprocessor architectures," in *Proc. IEEE*

International Symposium on Parallel Computing in Electrical Engineering 2006, PARELEC 2006., Bialystok, Poland, Sept. 2006, pp. 386 – 391.

[27] A. Itradat, M.O. Ahmad, A. Shatnawi, "A processor allocation of DSP applications onto heterogeneous multiprocessor architectures," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering 2007, CCECE* 2007, Vancouver, Canada, April 2007 pp. 944 - 947.

[28] A. Itradat, M.O. Ahmad, A. Shatnawi, "Minimization of I/O delay in the architectural synthesis of dsp data flow graphs," in *Proc. IEEE International Symposium on Circuits and Systems, ISCAS 2008*, Seattle, USA, May 2008, pp. 205 - 208.

[29] A. Itradat, M.O. Ahmad, A. Shatnawi, "Scheduling of DSP algorithms onto heterogeneous multiprocessors with inter-processor communication," in *Proc. 3rd International NEWCAS Conference,* Quebec, Canada, June 2005, pp. 95 – 98.

[30] R. W. Floyd, "Algorithm 97: shortest path," *Communications of ACM*, vol. 5, no. 6, pp. 345, 1962.

[31] J. Um and T. Kim , "Resource sharing combined with layout effects in high-level synthesis," in *Journal of VLSI Signal Process. Syst.*, Vol. 44, no. 3, pp. 231–243, 2006.

[32] H. Wei-Sheng, H. Yu-Ru, H. Juinn-Dar, H. Ya-Shih, "A multi-cycle communication architecture and synthesis flow for Global interconnect Resource Sharing," in *Proc. IEEE Asia and South Pacific Design Automation Conference, ASPDAC 2008,* March 2008, pp. 16 - 21 .

[33] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Trans Circuits Syst*, vol. 28, no. 3, pp. 196-202, 1981.

[34] D.Y. Chao and D.T. Wang, "Iteration bounds of single-rate data flow graphs for concurrent processing," *IEEE Trans Circuits Syst. -I*, vol. 40, no. 9, pp. 629-634, 1993.

[35] E.D. Lagnese, "Architectural Partitioning for Systems Level design of Integrated Circuits," *CMUCAD-89-27,* Carnegie Mellon University, Ph.D. *Thesis,* 1989.

[36] R. Gupta and G. DeMicheli, "VULCAN--A System for High-Level Partitioning of Synchronous Digital Circuits," Tech. Rept. CLS-TR-91-471, 1991.

[37] C.H. Gebotys,"An Optimal Methodology for Synthesis of DSP Multichip," *Journal of VLSI Signal Processing,* Vol. 11, pp. 9-19, 1995.

[38] S.Y Ohm, F.J. Kurdahi, and N.D. Dutt, "A unified lower bound estimation technique for high-level synthesis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 5, may 1997.

[39] K. Parhi, "Calculation of Minimum Number of Registers in Arbitrary life time Chart," *IEEE Trans. Circuits Syst. -II:*, vol. 41, no. 6, pp. 434-436, June 1994.

[40] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, J.L. van Meerbergen, and A. van der Werf, "Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 14, no. 8, pp. 945-960, Aug. 1995.

[41] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *IEEE Trans. on VLSI Systems,* vol. 3, no.1, pp. 2-19, March. 1995.

[42] K.R. Rao, P. Yip, *Discrete Cosine Transform*, Academic Press, 1990.

[43] S. Narayan and D. D. Gajski, "System clock estimation based on clock slack minimization," in *Proc. Eur. Conf. Design Automation*, Hamburg, Germany, pp. 66–71, 1992.

[44] A. Itradat, M.O. Ahmad, A. Shatnawi, "Incorporating of reconfigurable units in a simultaneous scheduling, allocation, and placement with interprocessor communication delay," To be submitted for possible publication in *IEEE Trans. on Computer Aided Design for Integrated Circuits and Systems.*

[45] A. Itradat, M.O. Ahmad, A. Shatnawi, "Architectural synthesis of DSP applications with dynamically reconfigurable functional units," in *Proc.*

International Symposium on Circuits and Systems. ISCAS 2007, New Orleans, USA, 27-30 May 2007, pp. 1037 - 1040.

[46] A. Itradat, M.O. Ahmad, A. Shatnawi, "Dynamically reconfigurable adaptable multi-module based synthesis of DSP data flow graphs," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2007*, Vancouver, Canada, April 2007, pp. 1515 – 1518.

[47] A. Itradat, M.O. Ahmad, A. Shatnawi, "Delay and sampling-rate aware architectural synthesis in presence of communication overhead," in *Proc. 3rd International IEEE-NEWCAS 2008 Conference,* Quebec, Canada, June 2008, pp. 323 – 326.

[48] S.W. Sloan, "A Fast Algorithm for Constructing Delaunay Triangulation in the Plane," Advances in Eng. Software, vol. 9, pp. 34-55, 1987.

[49] D. Kim, J. Jung, S. Lee, J. Jeon and K. Choi,, "Behavior-to-placed RTL synthesis with performance-driven placement," in *Proc. IEEE International Conference on Computer-Aided Design*, November 2001, pp. 320–325.

[50] H. Jang and B. Pangrle, "Grid-Based approach for connectivity binding with geometric costs," in *Proc. IEEE International Conference on Computer-Aided Design*, November 1993, pp. 94–99.

[51] M. Munch, N. Wehn, and M. Cleasner, "Optimum simultaneous placement and binding for bit-slice architecture," in *Proc. ACM/IEEE Design Automation Conference*, 1995, pp. 735–740.

[52] Y. Fang and D.F. Wong," Simultaneous functional-unit binding and floorplanning," in *Proc. IEEE International Conference on Computer-Aided Design*, 1994, pp. 317–321.

[53] J. Um , T. Kim," Resource sharing combined with layout effects in high-level synthesis," *Journal of VLSI Signal Processing Systems*, vol. 44 no.3, pp.231-243, September 2006

[54] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp.171-210, 2002.

[55] R. Hartenstein" A decade of reconfigurable computing: a visionary retrospective," *in Proc. ACM/IEEE Design Automation and Test in Europe Conference*, 2001, pp. 642.649.

[56] R. Tessier and W. Burleson., "Reconfigurable computing for digital signal processing: a survey," *Journal of VLSI Signal Processing*, vol. 28, no. 1, pp.7–27, 2002.

[57] Xilinx Corporation, Xilinx Spartan Series, www.xilinx.com.

[58] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera reconfigurable functional unit," in *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1997, pp. 87-96.

[59] S. Wallner, "A configurable system-on-chip architecture for embedded and real-time applications: concepts, design and realization," *Journal of Systems Architecture*, vol. 51, no. 6-7, pp. 350-367, June 2005.

[60] H. Singh, M. Lee, F. K. G. Lu, N. Bagherzadeh, and E. Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

[61] R. Kastner, S. Ogrenci-Memik, E. Bozorgzadeh, and M. Sarrafzadeh, "Instruction generation for hybrid reconfigurable systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 605-627, 2002.

[62] M. Galanis, G. Theodoridis, S. Tragoudas, and C. Goutis, "A high performance data-path for synthesizing dsp kernels," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 6, pp. 1154-1163, June 2006.

[63] M.Lee, H. Singh, G. Lu, N. Bagherzadeh, and F. Kurdahi, "Design and implementation of the morphosys reconfigurable computing processor," in *Journal of VLSI Signal Processing Systems*, vol. 24, pp. 147-164, 2000.

[64] S. M. S. A. Chiricescu, M. A. Schuette, R. Glinton, and H. schmit, "Morphable multipliers," in *Proc. 12th International Conference on Field Programmable*

Logic and Applications (FLP '02), , Montpellier, France, September 2002, pp. 647–656.

[65] K. Dale, J.W. Sheaffer, V. V. Kumar, and D. P. Luebke. Applications of small scale reconfigurability to graphics processors. Technical Report CS-2005-11, University of Virginia, 2005.

[66] J.C. Alves and J.S. Matos, "A simulated annealing approach for high-level synthesis with reconfigurable functional units," in *Proc. 38th Midwest IEEE Symposium*, Aug. 1995, pp. 314 – 317.

[67] A. Peleg, S. Wilkie, and U. Weiser, "Intel mmx for multimedia pcs," *Communications of the ACM*, vol. 40, no. 1, pp. 24–38, 1997.

[68] H. Shih-Hsu, H. Shih-Hsu, C. Chun-Hua, and N. Yow-Tyng, Y. Wei-Chieh, "Register binding for clock period minimization," in *Proc. Design Automation Conference, 2006 43rd ACM/IEEE*, 24-28 July 2006, pp. 439 – 444.

[69] P. Paulin and J.P. Knight,"Scheduling and binding algorithms for high level synthesis, " in *Proc. 26th Design Automation*, 1989, pp. 1–6.

[70] S.P. Mohanty and N. Ranganathan, "Simultaneous peak and average power minimization during datapath scheduling," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 52, no. 6, pp. 1157 – 1165, June 2005.

[71] R. Mehra, L. M. Guerra, and J. M. Rabaey, "A partitioning scheme for optimizing interconnect power," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 3, pp. 433 – 443, March 1997.

[72] K. Banerjee and A. Mehrotra, "An interconnect scaling scheme with constant on-chip inductive effects," *Analog Integrated Circuits and Signal Processing*, vol. 35, no. 2-3, pp. 97–105, May-June 2003.

[73] A. Chandrakasan, W. J. Bowhill, and F. Fox. Design of High Performance Microprocessor Circuits, IEEE Press, 2001.

[74] P. Caputa and C Svensson, "Well-behaved global on-chip interconnect," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 52, no. 2, pp. 318-323, Feb. 2005.

[75] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," in *Proc. ICCAD*, Nov. 2001, pp. 320 - 325.

[76] A. Kaplan, P. Brisk, and R. Kastner, "Data communication estimation and reduction for reconfigurable systems,' in *Proc. DAC*, June 2003, pp. 616 - 621.

[77] S. Shweta, M. Nazanin, and N. Adrian, "Pre-layout estimation of interconnect lengths for digital integrated circuits," in *Proc. the 16th IEEE Int. Conf. on Electronics, Communications and Computers,* 2006, pp. 38-43.

[78] L. Zhong and N.K. Jha, "Interconnect-aware low-power high-level synthesis," *IEEE Trans. Computer-Aided Design,* vol. 24, no. 3, pp. 336-351, March 2005.

[79] B. M. Pangrle, "On the Complexity of Connectivity Binding," *IEEE Trans. on Computer-Aided Design,* vol. 10, no. 11, pp. 1460-1465, NOV. 1991.

[80] S.H. Gerez, S.M. Heemstra de Groot, and O.E. Henmann, "A polynomial time algorithm for the computation of the iteration-period bound in recursive data flow graphs," *IEEE Trans. Circuits and Systems- I,* vol. 39, no. 1, pp. 49-52, Jan. 1992.

[81] K. Choi and S. P. Levitan, "A flexible datapath allocation method for architectural synthesis," *ACM Trans. Design Automation of Electronic Systems*, vol. 4, no. 4, pp. 376-404, 1999.

[82] T.A. Ly and J.T. Mowchenko, "Applying simulated evolution to high level synthesis," *IEEE Trans. Computer-Aided Design,* vol. 12, no. 3, pp. 389-409, March 1993.

[83] C. Gebotys and M. Elmasry, "Optimal synthesis of high-performance architectures," *IEEE J. Solid-State Circuits*, vol. 27, no. 3, pp. 389-397, March 1992.

[84] S. Shehata, B. Haroun, A. Al-Khalili, " A methodology for high level synthesis of high performance DSP structures targeting FPGAs," in Proc. *2nd International Conference on ASIC,* October 1996, pp. 89 – 92.

[85] F.J. Kurdahi and A. C. Parker, "Real: a program for register allocation," in *Proc. of DAC*, June 1987, pp. 210 - 215.

[86] C.Y. Huang, Y.S. Chen, Y.L. Lin, and Y.C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proc. 27th DAC*, 1990, pp. 499–504.

[87] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2004, pp. 68 - 73.

[88] H.A. Atat and I. *Ouaiss, "*Register binding for FPGAs with embedded memory," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines,* April 2004, pp. 165 - 175.

[89] A. Itradat, M.O. Ahmad, A. Shatnawi, "Interconnect-aware register binding with and without node regeneration for high-level synthesis," submitted for publication in *IEEE Trans. on Computer Aided Design for Integrated Circuits and Systems.*

[90] A. Itradat, M.O. Ahmad, A. Shatnawi, "Incorporation of reservation stations into the scheduling of DSP graphs onto heterogeneous multiprocessors," *in Proc. 48th Midwest Symposium on Circuits and Systems, MWSCAS 2005*, 7-10 Aug. 2005, Cincinnati, USA, Vol. I, pp. 460 - 463.

[91] R. Sedgewick, *Algorithms in C++*, Part 5 Graph Algorithms, 3rd ed., Addison Wesley, 2002.

[92] M. Sarrafiadeh and R.D. Lou, "Maximum k-coverings of weighted transitive graphs with applications," in *Proc. Int. Symposium on Circuits and System,* New Orleans, May 1990, pp. 332 - 335.

[93] A.H. Timmer and J.A.G. Jess, "Execution interval analysis under resource constraints", in *Digest of technical papers of ICCAD-93*, 1993, pp. 454-459.