A Fuzzy Logic Based Approach for High-Level Synthesis of DSP Data-Flow Graphs onto Multiprocessor Systems

Awni H. Itradat

A Thesis
in
The Department
of
Electrical and Computer Engineering

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

April 2004

Your file Votre référence
ISBN: 0-612-91050-4
Our file Notre référence
ISBN: 0-612-91050-4

ABSTRACT

A Fuzzy Logic Based Approach for High-Level Synthesis of DSP Data Flow Graphs onto Multiprocessor Systems

Awni H. Itradat

In recent years a great deal of research has been conducted in the area of synthesizing and scheduling the DSP data flow graphs onto multiprocessor systems. During the design of a DSP application, there exist many sources of uncertainty. For example, the design acceptability subject to certain constraints may be imprecise, since different designers may have different rules in determining an acceptable design. Another source of uncertainty could be that in the high level synthesis the final form of the hardware implementation of a functional unit is uncertain and imprecise, and therefore, its characteristics also uncertain. This thesis is concerned with the study of these sources of uncertainties in the module selection, allocation, and scheduling of the resources for DSP applications.

A fuzzy logic approach for module selection and process allocation of fully static DSP data flow graphs (DFG) onto multiprocessor systems is proposed. Fuzzy rule base systems are used to minimize the area and maximize the utilization of the processors within the constraint of a specified latency. The proposed technique provides the designer with more flexibility to explore the design space by using different types of processor modules for the same task. Both heterogeneous and general-purpose processor units are used during the resource allocation process. It is shown that in most cases, moving from a

fully homogenous to a fully heterogeneous architecture results in decreasing the design area. However, a hybrid multiprocessor architecture brings about a trade off between the area and the resource sharing.

Most of the static scheduling techniques assume the worst-case computational delay of the functional units used in the target architecture. This assumption is not realistic, since some of the computational time of the DSP tasks may be imprecise due to the fact that the design of the functional units may not have been completed at the layout level. Even if they are so designed, the fabrication process introduces variations in the resulting area and time. In this thesis, the impreciseness in the system components is taken into account by representing the computational time as a fuzzy set and efficiently constructing rate-optimal schedule using fuzzy arithmetic. The proposed approach can be incorporated into any technique for scheduling of cyclic or acyclic data flow graphs in order to obtain a more efficient schedule. By employing a fuzzy rule base system, the characteristics of the derived system, namely the latency and area, are then used to infer the acceptability of the design of the target architecture. The design acceptability inferred by the proposed approach is shown to be close to the one inferred by the conventional scheduling approach, when the best computational times of the functional units are assumed.

To My Loving Family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and heartfelt thanks to my thesis supervisors Drs. M.O. Ahmad and Ali Shatnawi for their interest, guidance and constructive criticism throughout this work. I am grateful for their extremely careful and thorough review of my thesis. I feel privileged for having the opportunity to work with them. Their advice and close support have been invaluable.

I am grateful to my family members 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, for her patience, encouragement and love. She provides me with the peace of mind and the determination to complete this work.

Finally, I would like to dedicate this thesis to the soul of my father who passed away only a few days prior to my thesis defense. Yabah, I promise that I will do my best to keep up to your expectations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

V	A node in a data flow graph
E	An edge in a data flow graph
$p_{v_0v_k}$	Path between the two nodes (v_0 , v_k)
T_0	Iteration period bound
D_l	Total computational delays of all the nodes in loop l
N_l	Number of ideal delay elements loop l
$ST(l)$	The slack time of a loop l
U	Universe of discourse
A	Arbitrary fuzzy set
$\mu_A(u)$	Membership value of the element u
$\eta_+(m)$	Positive contribution of module m
$\eta_-(m)$	Negative contribution of module m
$adj_f(m)$	Adjustment of utility of module m
EFT	Earliest firing time
LFT	Latest firing time
$M(v_j)$	Mobility of node v
β	Set of multiprocessor configurations

Chapter 1

Introduction

1.1 General

Digital signal processing (DSP) and image processing are among the areas demanding a very high computational power for the implementation of the underlying tasks. Due to the parallelism within the DSP applications, a multiprocessor system is a natural choice for the implementation of these applications. Furthermore, the progress in VLSI technology has resulted in an enormous increase in the hardware execution speed. This has resulted in a wide variety of signals to be processed by digital circuits. Not too long ago, the digital processing of signals was restricted to low-bandwidth signals such as speech. The DSP applications have now been extended to include real-time processing of high-quality audio and video signals.

VLSI technology has now advanced to a stage that it would be rather complex to design a digital system starting at the transistor level or logic level. In the VLSI design, high-level synthesis refers to designing a system at the register transfer level (RTL), rather than looking at the components at the logic gate level. In such a design, a behavioral description or block diagram is used. At this level, an adder, for instance, is viewed as a functional unit instead of as a series of flip-flops and NAND gates. The attributes such as latency, area and type of functional units, however, can be taken into consideration without having to deal with low-level implementations. Scheduling, allocation and module selection are the key tasks that influence these attributes in the

high-level synthesis. A general goal of the high-level synthesis is to find hardware structures that minimize these design attributes subject to certain constraints.

Generally, the design flow of digital systems can be divided into three phases: high level synthesis, logic synthesis and layout synthesis. The higher the level of synthesis, the more the degree of freedom. However, at a higher level of synthesis, less information available about the final circuit parameters to guide the design, and the quality of the final result is considerably affected by the decisions made about the circuit parameters at a higher level of synthesis. Wrong decisions made at a higher level of synthesis could be more expensive than those of lower level. For instance, if a mistake is detected after placing and routing, the whole design process may have to be restarted.

One of the most important issues in high-level synthesis of a DSP task is to obtain a good schedule so as to reduce the overall computation time and the total area. In order to implement and execute a system efficiently, one has to determine an execution order or schedule of all operations in the data flow graph of the DSP application. In order to construct an effective execution order, the knowledge of the characteristics of the functional units, namely, the area and the execution time should be available. However, during the high-level synthesis both the timing and area information of a design component remain imprecise, and ignoring the impreciseness in the synthesis may result in an overly expensive and time consuming design.

In order to achieve a better design of a DSP application, the impreciseness in the design attributes during the synthesis task needs to be properly treated. In a high-level synthesis the impreciseness refers to the lack of the exact knowledge of the relationship between the architectures of the functional units or their final implementations and the

performance of the overall design. The impreciseness during the synthesis of DSP data flow graphs may arise from several sources. It could be from the fact that there exist various choices of off-the-shelf processor modules that can be used to implement a functional unit; hence, the decision as to which architecture should be chosen is ambiguous. Moreover, the detailed characteristics of the functional units are not known at this stage. Since the exact implementation is unknown, the functional unit's characteristics remain imprecise.

Since scheduling, process allocation and module selection are the most critical tasks in the multiprocessor the synthesis of a DSP application represented by data flow graph, on a multiprocessor system, an efficient scheme should be developed to accommodate the imprecise environment. The following sections will first provide a brief review of the necessary background material for the high-level synthesis of the DSP tasks onto multiprocessors. In order to provide motivation for the research work under taken in this thesis, the weaknesses of the current research in the high-level synthesis of such applications and the shortcomings of the schemes of scheduling and allocation in the underlying multiprocessor systems are also discussed. Finally, the objectives of the research and the organization of this thesis are described.

1.2 Data Flow Graph

The operations and their precedence relations in an application constitute its specification, which is generally modeled by a vertex-weighted directed graph. The graphical format is thus the representation of the data flow mechanism implied by the system algorithm. The data flow graph (DFG) is proven to be an efficient representation

of the system specification due to its ability to explore 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.

Before discussing the properties of DFGs, some definitions used in the graph theory are first given in here: A graph *G*, such as the one shown in Figure 1, can be represented by the pair *(V, E)*, where *V* is a set of vertices or nodes, and *E* is a set of elements called edges. Each edge is associated with a pair of nodes. A graph is directed if each edge in *G* is associated with an ordered pair of nodes [4].

The symbols $v_1, v_2 \ldots v_{n-1}, v_n$ are used to represent the nodes and the symbols $e_1, e_2, \ldots$ are used to represent the edges of the 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.

A direct path $p_{v_0 v_k}$ is a finite sequence of distinct nodes $v_0 \ldots v_k$ 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.*

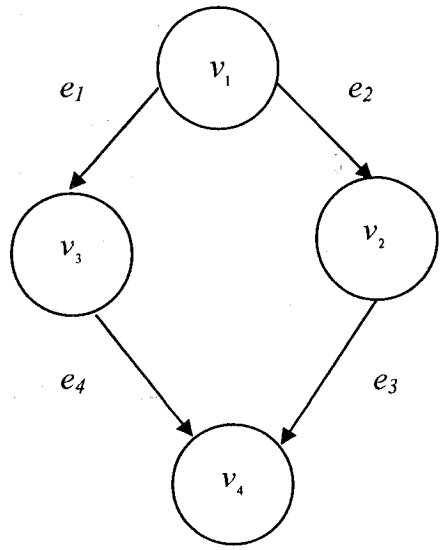

Figure 1.1: A simple data flow graph with four nodes.

1.3 High-Level Synthesis

A high-level synthesis can be described as the process of transformation of a behavioral description into a structural description 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 synthesis task starts at behavioral level and proceeds downwards to register transfer level (RTL), logic level and finally circuit level, each time adding some additional information needed at the next level of synthesis.

The synthesis task is to take 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 inputs to 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 [1].

In recent years there has been a trend toward carrying out the synthesis at higher and higher levels of the design hierarchy. High-level synthesis is gaining acceptance in industry, and there has been considerable interest shown in the synthesis at higher levels. There are a number of reasons for this [1, 2]:

a. **Shorter design cycle.** Since more of the design processes at lower levels are automated, it is possible to hit the market window by carrying out the synthesis at higher levels. Furthermore since much of the cost of the chip is in the design development, automating more of that process can lower the cost significantly.
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 different trade-offs between cost, speed, power, and so on, or to take an existing design and produce a functionally equivalent one that is faster or less expensive.

It is expected that the trend toward the 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 trend 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 high level. First, designer can concentrate on studying the design behavior rather than the detailed implementation. Second, 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 best fits their needs.

Some of the applications that need such high level synthesis are digital signal processing (DSP), communications, and image processing. These applications demand a high computational power. That is, these applications are computational intensive, and must be executed at high speed with high throughput. Furthermore, issues such as low cost, low power and small chip area are other important issues. Due to the parallelism within DSP applications (e.g. digital filters), a multiprocessor system is a natural choice for the implementation of these applications [2, 3].

An interesting technique used to implement a DSP application is known as application specific integrated circuit (ASIC). The ASICs are integrated circuits that are dedicated to perform specific tasks. They can be designed at the RTL level from multiprocessing elements, memory storage, and necessary interconnection.

1.4 The Synthesis Task

Scheduling, allocation, and model selection are the most important steps in the synthesis task. They represent the core of transforming the behavior 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 a clock cycles. Allocation consists in assigning the operations to generic hardware resources. Module selection is the process of selecting suitable functional unit subject to constraints.

1.4.1 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 of the application given certain constraints on the available hardware resources [5-7].

Scheduling is significant when 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 [7].

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 run time. In this thesis, only static scheduling is considered.

Another characteristic of a scheduling method is whether or not it allows operations to be interrupted once their execution has begun. If it is 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 operations are executed without interruptions.

When an algorithm is scheduled for execution on a multiprocessor, 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*. For resource constrained scheduling the hardware is specified and it cannot be changed by the scheduling method. In contrast, a *time-constrained scheduling* tries to use as little hardware as possible when an execution speed is given.

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 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. Schedules corresponding to subsequent iterations can have a

constant displacement in the processor space. Cyclo-static schedules differ from *fully-static schedules* in which each operation is assigned to the same processing element for all iterations.

Performance Bounds

There are some performance bounds for multiprocessor scheduling problems. These bounds give the minimum values for some of the optimization goals that can be chosen. These bounds cannot always be achieved for every multiprocessor configuration, yet they provide a means to estimate the quality of schedules found by a scheduling algorithm. Furthermore, they can be used by the scheduling method to guide its search for a good schedule. Some bounds that are commonly used [8, 9] are now briefly described.

The *iteration period bound* (IPB) gives a lower bound on the iteration period when unlimited hardware is available. When the DFG contains no loops, the iteration period can be made arbitrarily small. Otherwise IPB is calculated as

$$T_0 = IPB = \max\left\lceil \frac{D_l}{N_l} \right\rceil$$

where l represents a loop in the cyclic DFG and D_l is the sum of the computational delays of all the nodes in loop l, *and* N_l is the number of its ideal delay elements. The loop l that results in the IPB is called the *critical loop* of the DFG.

To give an example, consider the second-order filter given in Figure 1.2. Assume that an addition requires 1 TU (time unit) to execute and a multiplication 2 TU. The critical loop of this example is shown Figure 1.2. It easily follows that the iteration period bound is 3.

The slack time of a loop l is defined as the number of control steps by which the loop can be expanded before violating any inter-iteration precedence constraints, and is given by

$$ST(l) = T_0 N_l - D_l$$

where $T_0 = IPB$, the slack time of the critical loop of a schedule is usually zero. A slack time of zero implies that when one operation in the loop is scheduled, there is no freedom in choosing the firing times of the other operations in the loop. These firing times are then all fixed. When the slack time of a loop is larger than zero, there is more freedom of choice in scheduling the operations in the loop.

Another bound is the *periodic delay bound* (PDB), which gives the lower bound on the latency of the schedule when the iteration period is equal to IPB. The bound is given by

$$PDB = L_0 = \max_{P \in i/o\ path} \left(D_P - T_0 N_P\right)$$

where P is the set of paths from input to output. The path P that determines the value of PDB is called the critical path.

The third bound is the *processor bound* (PB), which gives the minimum number of processors that are necessary to have a schedule with $T_0 = IPB$.

$$PB = \left\lceil \frac{D_{DFG}}{T_0} \right\rceil_{T_0 = IPB}$$

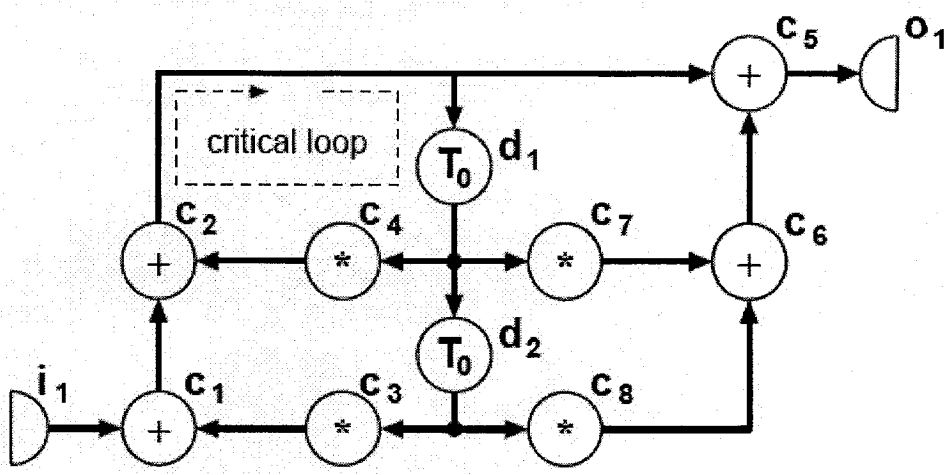

Figure 1.2: The data flow graph for second-order filter

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 that will be described later 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) and as *late as possible* (ALAP) algorithms.

ASAP Algorithm

A simple scheme to find the earliest control step for an operation to be scheduled is carried out by using the "as soon as possible" algorithm, as is done in Carnegie Mellon University's Emerald/Facet system [10] and CATREE system from University of Waterloo [11].

The ASAP algorithm starts with the highest nodes (that have no parents) in the DFG and assigns these nodes to control steps in increasing order as it proceeds downwards. It follows a simple rule, that is, a successor node can be executed only after

its parent has been executed. This algorithm clearly results in a schedule with least number of control steps but never takes into account the resource constraint. Figure 1.3b illustrates an ASAP schedule for the DFG shown in Figure 1.3a.

ALAP Algorithm

This algorithm is a refinement of the ASAP scheduling concept with conditional postponement of operation [12]. This algorithm is used to find the latest control step for an operation to be scheduled. The ALAP algorithm works exactly in the same way as the ASAP Algorithm expect that it starts at the bottom of the DFG and proceeds upwards given a certain latency. This algorithm gives the slowest possible schedule that takes the maximum number of control steps. However, this does not necessarily reduce the number of functional units used. Figure 1.3c illustrates an ALAP schedule for the DFG shown in Figure 1.3a.

1.4.2 Classification of Multiprocessor Scheduling Techniques

Over the years researchers have tried to come up with various kinds of solutions [10-21] to the scheduling problem. Several algorithms have been put forth and each one has its own advantages and disadvantages. Scheduling algorithms can be broadly classified into time-constrained scheduling and resource-constrained scheduling, based on the goal of the scheduling problem. In time-constrained scheduling, the number of functional units tends to be minimized for a fixed number of control steps. On the other hand, in resource-constrained scheduling, the number of control steps tends to be minimized for a given design cost (number of functional units and storage units).

(a) Time-Constrained Scheduling

The time constrained scheduling is also called fixed control steps approach. The time constrained scheduling is important for the designs targeted towards application in real time systems such as digital signal processing (DSP) systems, where the main objective is to minimize the cost of the hardware. An example of a time constrained scheduling algorithm is the force-directed scheduling method.

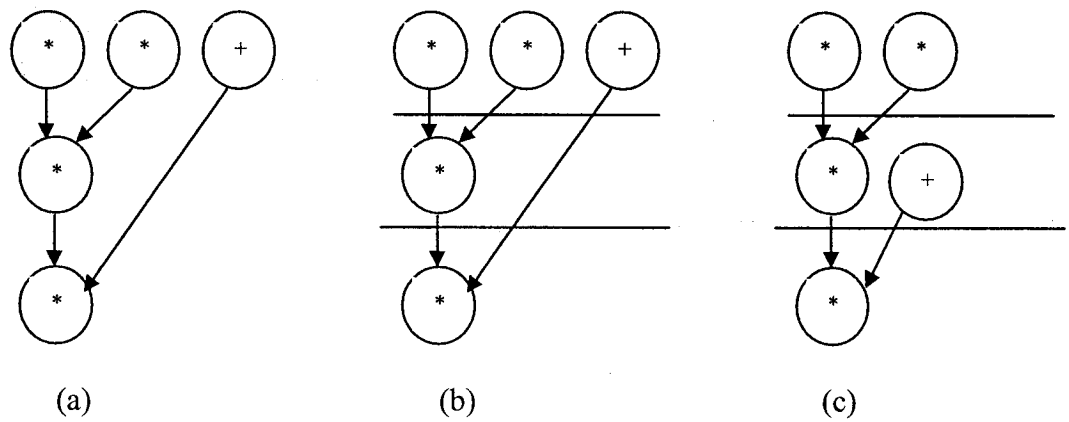

Figure 1.3: (a) A data flow graph (b) As Soon As Possible scheduling (c) As Late As Possible scheduling.

Force-Directed Scheduling

The force-directed scheduling (FDS) is a heuristic method [14, 15] that is a very popular scheduling technique for time constrained scheduling. This algorithm achieves the goal of minimizing the total number of functional units by uniformly distributing the operations of the same type over the available control steps. This algorithm is briefly explained below.

FDS uses a guiding factor called force, which is calculated for each operation. This guiding factor is used in the selection of the operation to be scheduled and in the

choice of the control step. The force between an operation and a particular control step is proportional to the number of the operations of the same type that can be scheduled for that control step. The schedule is built by giving a priority to the minimum force value for a pair of operation and control step. Then, the forces are updated and the process repeated. The advantage of this technique is that it can be used in scheduling with pipelined functional units.

The complexity of the FDS algorithm is $O(cn^2)$, where c is the number of control steps and n is the number of the operations in the DFG. The FDS algorithm does not always produces an optimal solution.

(b) Resource-Constrained Scheduling

In applications where the design is restricted by the silicon area, resource constrained scheduling algorithm is useful. The goal of these algorithms is to produce a design with the best possible performance but still meet the given resource constraints. The schedule is gradually constructed, one operation at a time, so that the resource constraints and data dependencies are not violated. The number operations scheduled in any control step should not exceed the number of functional units available. Also the algorithm ensures that all the predecessors are scheduled before that operation. The following is the description of two popular scheduling algorithms in this category.

List-Based Scheduling

The scheduling technique described in [16] belongs to list-based scheduling algorithms. List-based scheduling is a generalization of the ASAP scheduling algorithm with the inclusion of the resource constraint. A list-based scheduling maintains a priority list of ready nodes, i.e., nodes whose predecessors have already been scheduled. The priority

order for each operation is sorted with a priority function that resolves any resource contention. In any iteration, operations with higher priority are scheduled first and lower priority operations are deferred to later control steps. Scheduling an operation to a control step makes other succeeding operations ready, to be added to the priority list. A simple priority function can be inversely proportional to the mobility (the difference between ALAP and ASAP schedules), i.e., the greater the mobility the lower the priority and vice versa.

As can be seen, the success of a list scheduler depends mainly on the priority function used. Mobility is a good priority function, since smaller the mobility higher the urgency for scheduling. There are many other priority functions that have been proposed. The time and space complexity for this approach is slightly higher than the other list-based scheduling algorithms, since several lists have to be maintained dynamically.

Static List Scheduling

This approach [17] starts by creating a single large list before starting scheduling. This algorithm differs from the ordinary list-based scheduling algorithm in both the assignment of control steps as well as in maintaining the priority list. The algorithm first uses the ASAP and ALAP algorithms to obtain the earliest and the latest possible control step assignments for each operation. The algorithm then sorts all the operations in an ascending order of latest possible control step labels as the primary key and then sorts each set of operations with the same latest control step labels in a descending order of the earliest possible control step labels as the secondary key.

Once the priority list is created the operations are scheduled sequentially starting with the last operation in the priority list (i.e., the operation with the highest priority). In

each iteration when the limit for the number of resources is reached, the rest of the operations are deferred to later control steps. This scheduling technique has an advantage over the ordinary list based approach: A list is constructed once statically and not grown dynamically.

Comparison of List-Based and Static List Scheduling Algorithms

The two scheduling algorithms described above are the most popular methods for scheduling operations under resource constraints. Table 1.2 gives a comparison between the two approaches.

TABLE 1.1: A COMPARISON OF RESOURCES CONSTRAINED SCHEDULING ALGORITHMS

Properties	List-based	Static list
Computational complexity	High	Not very high
Quality of schedule	Mostly optimal	Mostly optimal
Space complexity	Very high	Not high
Input problem size	Any size	Any size
Execution speed	Slower than FDS	Faster than list-based

1.4.3 Allocation Process

Allocation is a task of determining generic resources (functional units) on which operational nodes are executed. It involves assigning operations and variables to hardware resources and specifying their usage while trying to minimize the amount of hardware resources needed. It assumed that an instance of a generic resource could only start 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 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 the minimize of the number of hardware resources required for the implementation of a digital system, the operations (nodes) of a DFG 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.

Hardware functional units may be homogenous or heterogeneous. The functional units are homogenous if all of them are of the same type and execute the same set of different types of operations. Each type of operations has a different computational delay, since such functional units usually require more hardware resources to support multiple operations. Homogenous functional units if used in the synthesis process may result in inefficient designs. Arithmetic logic units (ALUs) are suitable for homogenous design. The heterogeneous functional units, on the other hand, vary in the operations they support, and have different execution delays [15].

Data path allocation involves the mapping operations, onto the functional units, and also assigning values to registers, and providing interconnections between the operators and registers using buses and multiplexers. The optimization goal is usually to minimize some objective function, such as the total interconnect length, the total number of registers, the bus driver 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.

The techniques used to perform data path allocation can be classified into two types, the global type and the iterative/constructive type. The global techniques find simultaneous solutions to a number of assignments at a time, whereas the iterative/constructive techniques assign nodes one at a time. Exhaustive search is an extreme case of the global technique. The iterative/constructive techniques generally look for a smaller of the search space than the global techniques do, and therefore, are more efficient, but are less likely to find the optimal solutions.

1.4.4 Combined Scheduling and Allocation

Both scheduling and allocation are the required tasks in a high level synthesis. To achieve optimized synthesis results, the scheduler must have some knowledge in advance about the hardware resources available and those needed by the target system. This knowledge includes the information about the hardware units, such as the computational delay, access time, and interconnection. This information is available after the hardware allocation process has been completed. In addition, the allocation process needs to know the control step at which an operation is to be scheduled. As a result, the minimum number of hardware required and the operations assigned to them are determined. Thus, there is an interdependence between the scheduling and the process allocation.

Many researchers focus on performing the tasks of scheduling and allocation in two different stages. Since there are interactions between these tasks, another attractive approach is to solve both of these tasks simultaneously. Regardless of the approach used, using the constraints specified in advance under the user control, it is possible to start with the one that best achieves these constraints.

1.4.5 Module Selection

In the scheduling and allocation techniques discussed in sections 1.4.1-1.4.4, the resource types have been assumed to be generic. That is, all the adders (multipliers) are of the same kind, if more than one adder (multiplier) is needed. In the module selection problem, this assumption is no longer valid. For instant if two multipliers are required, these may be of different kinds, e.g. fully parallel and serial-parallel multipliers. Both the multipliers can provide the same functionality but have different characteristics, such as one is faster than the other and/or one smaller than the other. When considering module selections, multiple design attributes, such as timing, area, and power etc. are often involved. These attributes normally conflict with each other. The task of module selection is to find a mapping from the functional units to the existing modules (architectures) in a library, which leads to an optimized design depending on the design criteria.

Besides finding the schedule and the allocation, the module selection is another important phase in a high-level synthesis. They all have relationships with one another. Different selected modules may result in a different schedule/allocation and vice versa. This then results in a system with a different performance.

1.5 Design Space Exploration

Since there are various choices of the modules to be used in a synthesis process, a system can have several implementations. Each implementation has a different set of properties. The design space in this context refers to the set of all possible module bindings as well as their schedules. Given a module binding, the best schedule needs to be generated for a design, in order for it to be evaluated and compared with other designs.

Considering the tradeoffs among the conflicting criteria in evaluating the module bindings and their schedules in the design space, the design exploration, therefore, is the process of finding the most desirable design from the design space. The meaning of the term "desirable" depends on the design criteria. If our goal is to minimize both the latency and the area, an optimal design refers to the design whose latency and area is the least.

High-level synthesis provides a designer with the capability of a rapid design space exploration. A designer can examine the tradeoffs between the various designs without getting into the details of the implementation. The exploration of the design space at a lower level, say at logic level, is limited, since changing each gate level design is a time consuming task.

1.6 The Weaknesses of the Current Research in the High-Level Synthesis

In the previous sections, a review of the existing work in the high-level synthesis has been given. Table 1.2, summarizes the weaknesses in the previous research that has been conducted in the area of high-level synthesis. Scheduling, process allocation, and module selections are inter-dependent tasks in a high-level synthesis. The result of one can affect the others. Because of their complexity, some researchers have ignored the dependencies between these tasks and tackle each one as an individual phase. Leive *et al* [18] have presented a solution to the module selection problem. In this scheme, the module selection phase is carried out without taking into account the effects of the scheduling and process allocation on that selection. Jain and Parker [19] have developed a module

selection framework in high-level synthesis. However, they have assumed that the module selection phase is performed prior to the scheduling and process allocation.

TABLE 1.2: THE WEAKNESSES IN THE PREVIOUS ALGORITHMS FOR SCHEDULING, PROCESS ALLOCATION, AND MODULE SELECTION IN THE HIGH LEVEL SYNTHESIS

Algorithm	Weaknesses
Reference [18]	Does not consider simultaneous scheduling, process allocation, module selection.
Reference [19]	Does not consider simultaneous scheduling, process allocation, module selection.
Reference [20]	Uses exponential ILP approach to find the schedule, which is time consuming
Reference [21]	Impractical for large systems
Reference [22]	Uses branch and bound approach to find the best modules bindings which time consuming.
Reference [23]	Generates too many numbers of schedules to find the best module binding.
Reference [24]	Does not consider multiple design criteria in evaluating the synthesis results in the design space

In the work concerning a combined scheme of scheduling, process allocation, and module selection, several heuristics have been proposed. Some approaches use artificial intelligence methods or simulated annealing [23], whereas others use an iterative heuristic [24]. All these methods can not be applied to large systems in practical sense. Other techniques [20] have attempted to find an optimal solution via an integer linear

programming (ILP) approach. However, the complexity of the approach grows exponentially.

Regardless of the consideration of the interaction between the synthesis tasks, none of the approaches cited in Table 1.2 consider the impreciseness in the system attributes in developing the synthesis models.

1.7 Scope and Organization of the Thesis

As discussed above, none of the previous work in the scheduling, process allocation and module selection for the high-level synthesis considers the impreciseness that arises from the lack of the exact knowledge of the relationship between the architecture of the functional units and their final implementations and the performance of the overall design. The objective of this thesis is to incorporate these sources of impreciseness into a more general model of a high-level synthesis in order to find the scheduling, process allocation, and module selection that are practically more realistic in achieving the final implementation. In this thesis, the impreciseness incorporated in the synthesis model is represented by using fuzzy theory and fuzzy rule base.

The inter-dependence among the scheduling, process allocation, and module selection is included in the synthesis framework developed in this thesis. This framework tends to explore the design space more efficiently; since it allows various choices of the modules to be used during the synthesis process. Having more flexibility in the choice of modules in the high-level synthesis provides the designer the ability to find a more optimized design.

In Chapter 2, background material on fuzzy logic necessary to characterize the impreciseness in the system attributes and in developing the synthesis model is reviewed. In Chapter 3, a framework using fuzzy rule base and the theory of fuzzy logic for the module selection that allows the designer to select the best modules for the design depending on the tradeoffs between the system attributes, area and latency, is proposed. Chapter 4 presents a new allocation technique that use fuzzy rule base to find the best multiprocessor configuration for rate-optimal scheduling of a fully static data flow graph (DFG). In Chapter 5, a new algorithm for scheduling of DSP data flow graphs with processing time characterized by fuzzy sets is proposed. Chapter 6 concludes the thesis by highlighting the contribution of this research and making some suggestions for possible future work.

Chapter 2

Fuzzy Logic and Uncertainty

2.1 Introduction

Fuzzy logic was developed by Lotfi A. Zadeh in the 1960s in order to provide mathematical rules and functions which permitted natural language queries. Fuzzy logic provides a means of calculating intermediate values between absolute true (a value of 1) and absolute false (a value of 0). With fuzzy logic, it is possible to calculate the degree to which an item is a member of a set. Fuzzy logic calculates the shades of gray between black/white and true/false.

Fuzzy logic is a superset of the conventional (or Boolean) logic and contains similarities and differences with the Boolean logic. Fuzzy logic is similar to the Boolean logic, in that Boolean logic results are returned by fuzzy logic operations when all the fuzzy memberships are restricted to 0 and 1. Fuzzy logic differs from the Boolean logic in that it is permissive of natural language queries and is more like human thinking; it is based on degrees of truth.

This chapter gives an introduction to the fundamental notions and concepts of fuzzy logic and fuzzy sets. In the following section, we address different points of view about the definition of uncertainty.

2.2 Definition of the Uncertainty

In this section we review different points of view about the definition of uncertainty, its types, and causes. These points of view belong to Zimmerman [25], Klir [26] and Dubois and Prade [27].

2.2.1 Zimmerman's Point of View

Zimmerman [25] defines "certainty" as "the case when one has the appropriate quantitative and qualitative information to describe, prescribe or predict deterministically and numerically a system, its behavior or other phenomena". Anything not described by this definition shall be called "uncertain". Furthermore, Zimmerman introduces a classification of uncertainty causes based on the quality and quantity of the available information. He classifies the causes of uncertainty as follows.

• Lack of information

This cause of uncertainty may be considered as the most frequent one. For example, in decision logic, one calls "decision under uncertainty" the case in which a decision making process lacks information about the possible states of nature that will occur. This kind of information which is not available can be considered as quantitative lack of information. The counterpart of this kind of information lack is the qualitative one. In this case, the decision making process has information about the probabilities for the occurrence of various states but it is not sure which state will occur; this is called "decision making under risk". "Approximation" is another situation that can be described by the lack of information. This depends on the situation presented. For instance, one can consider that the available information is sufficient for his/her situation and he/she does

not have or does not want to gather more information to make an exact description. Transition from a situation of "uncertainty" caused by a lack of information to a situation of "certainty" can be achieved by increasing the available information or collecting information with better quality which depends on the situation.

• Abundance of information

This is due to the capability of a system to process a large amount of data at the same time. To reduce the complexity, people tend to classify the available data into understandable form by using coarser grid or rougher "granularity" or by concentrating on the most important features and neglecting the not useful information for that situation. To do so, especially in scientific activities, some kind of "scaling" is used.

• Conflicting evidence

This situation occurs when the available information describing two different behaviors of a system are conflicting. The reason for this conflict may be the erroneous available information, it may also be information of irrelevant features of the system is being used, or the model of the system which the observer has is wrong. In this situation, correcting the available information can make transition from "uncertain" to state of "certain.

• Ambiguity

Ambiguity is a situation in which certain information has a different meaning based on the situation. From the mathematical point of view, it is the situation in which we have one-to-many mapping. This type of uncertainty can be classified under lack of information because adding more information about the situation may put us in a situation closer to certainty.

• **Measurement**

Measurement means describing the physical properties of a system or objects such as weight, temperature, length, etc. The precision of the measured quantity depends on the accuracy of the used tools. The quality of measuring technology has increased with time but it has not reached the perfection. In this situation we have uncertainty about real measure and the only available information is the indicated measure.

• **Belief**

This cause of uncertainty appears when subjective information is available as a kind of belief in a certain situation. This belief is built by an observer (expert) from past subjective information about the system or by statistical data about the system.

2.2.2 Klir's Point of View

Klir [26] found that there are six definitions of the word "uncertain" in the dictionary:

• Not certainly known, questionable, problematical.

• Vague, not definite or determined.

• Doubtful, not having certain knowledge, not sure.

• Ambiguous

• Not steady or constant, varying.

• Liable to change or vary, not dependable or reliable.

When a more detailed investigation about these meanings was conducted, Klir found that uncertainty can be captured by two classes; vagueness and ambiguity. The former is related to the difficulty of making sharp or precise distinction in the world. The latter is associated with one-to-many relations, which means situations with two or more

alternatives in which the choice between them is left unspecified. In addition, Klir introduced a recent definition of uncertainty based on its connection with the information theory. The most fundamental aspect of this connection is that uncertainty included in any situation is a result of some information deficiency. Information may be incomplete, imprecise, fragmentary, not fully reliable, vague, contradictory, or deficient in some other way.

2.2.3 Dubois and Prade's Point of View

Dubois and Prade [27] state that imprecision and uncertainty can be considered as two complementary aspects of a single reality, that of imperfect information. It has been observed that much of this information often cannot be obtained as precise and definite numbers for various reasons; imperfect measuring instruments, the fact that the sole source of information is a human being and the information is imprecise, incoherent, and in any case incomplete. Doubois and Prade could clearly distinguish the concepts of imprecision and uncertainty: imprecision is associated with the content of a piece of information, while uncertainty is associated with its truth. Imprecision refers to lack of knowledge about the value of a physical parameter. The possible values of the parameter are represented by a certain interval obtained experimentally or from an expert. This interval represents the imprecision in the physical parameter. Certainty refers to the degree of truth that the value of the physical parameter belongs to a certain interval. In other words, each element belonging to this interval has a certain possibility to be the actual value of the physical parameter. This possibility is associated with a weight that is derived from the available knowledge about the physical parameter. When there are different imprecision intervals representing the value of the physical parameter, then

these intervals are used to construct a new interval without sharp boundaries. This interval is represented by a fuzzy set as it is explained later in this chapter. Each element belonging to this interval has a degree of truth of being the actual value of the physical parameter. Therefore, this new interval represents the uncertainty of the physical parameter.

Uncertainty can be judged by means of different qualifiers such as probable, possible, or necessary. Probable has two different meanings, one is related to statistical experiments, and the other is related to subjective judgment. Like probable, possible has two interpretations as well: physical (as a measure of material difficulty of performing an action), and subjective judgment. On the other hand, necessary has much stronger notion, in either the physical or the subjective sense. A piece of information will be called precise when the subset associated with its value or component cannot be subdivided. From the above overview about uncertainty and its causes, it is seen that Dubois and Prade's definition of uncertainty is more comprehensive and practical than those the others.

2.3 Fuzzy Sets and Fuzzy Logic

Fuzzy sets were introduced in 1965 by Lotfi Zadeh [28], as a means to model the uncertainty of the real world. They are used to represent imprecise, ambiguous, or vague information. Fuzzy logic which was introduced in 1973 by the same author [29], is a superset of the conventional Boolean logic that has been extended to handle the intermediate values between "completely true" and "completely false".

Boolean logic has two values often defined as true or false, on or off, black or white. However, in the real world there are many situations where events are not black or

white but some shade of gray. Fuzzy logic is a continuous form of logic that allows us to describe the shades of gray. If one is asked to describe his/her day in Boolean logic, it would be good or bad. Fuzzy logic might recognize the day as being very bad, bad, poor, average, better than average, good, very good.

Classic logical systems are based on Boolean logic, which assumes that every element is either a member or a non-member of a given set (never both). Unfortunately, this system imposes an inherent restriction on the representation of imprecise concepts. Assuming that Boolean logic is used to identify whether a room temperature is "hot" or "cold", most people would agree that 100°F is a "hot" room temperature and 25°F is a "cold" room temperature. However, if the room temperature falls to 75°F, it becomes much harder to classify the temperature as "hot" or "cold"; Boolean logic does not provide the means to identify an intermediate value.

Fuzzy logic extends Boolean logic to handle the expression of vague concepts and, as a result, solve the preceding problem. To express imprecision in a quantitative fashion, it introduces a set membership function that maps elements to real values between zero and one (inclusive); the value indicates the "degree" to which an element belongs to a set. A membership value of zero indicates that the element is entirely outside the set, whereas a one indicates that the element lies entirely inside a given set. Any value between the two extremes indicates a degree of partial membership to the set. In the example discussed in the previous paragraph, if fuzzy logic is used to represent the "hotness" of a room, 100 °F would have a membership value of one and 25 °F would have a membership value of 0. On the other hand, 75 °F would have a membership value between zero and one.

Zadeh defines the process of "fuzzification" as a methodology to generalize any specific theory from a crisp to a fuzzy form. This is achieved by applying the extension principle. Researchers have applied this principle on many areas such as control, reasoning, mathematical programming, decision making, pattern recognition, and many others.

In addition to its role in modeling and processing imprecise or ambiguous information, fuzzy logic is used to model complex systems. These systems are difficult to be described using mathematical relations. In addition, mathematical modeling becomes more difficult when there are uncertainties and ambiguities in the systems to be modeled.

Zadeh's approach was later expanded into fuzzy systems modeling by Sugeno and Yasukawa [30], Bezdek [31]]. Fuzzy modeling is a qualitative modeling scheme by which we qualitatively describe system behavior using natural language. The relation between the inputs and outputs of the system is given in the form of IF-THEN rules. There are two approaches of fuzzy systems modeling: one is *subjective* where the system behavior is established based on the knowledge of an expert and the other is *objective* where the system behavior is established from input-output data via fuzzy clustering algorithms [32].

2.4 Classical Set and Fuzzy Set

Let U be the universe of discourse which consists of all possible elements that are associated with a particular context or application. A crisp set A defined on U may be represented by listing all the elements that satisfy the definition of A in the case that A is

finite (the list method). If A is infinite, it can represented by specifying the rules that must be satisfied by the elements of U to be considered elements of A (rule method). The former method is limited. On the other hand, the later method is more general. The membership function $\mu_A(x)$of a classical set A defined on U by using the rule method is defined by:

$$\mu_A(u) = \begin{cases} 1, & if \quad u \in A \\ 0, & if \quad u \notin A \end{cases}$$

This means that an element u is either a member of set A (with $\mu_A(u) = 1$) or not a member (with $\mu_A(u) = 0$). A fuzzy set, introduced by Zadeh [27], is a set with graded membership. In a fuzzy set each element of U belongs to the set A with a membership degree characterized by a real number in the closed interval [0, 1] (i.e., $\mu_A(u) \in$ [0, 1]). An element may belong to the fuzzy set with lesser degree than another element; however, they both belong to the same fuzzy set.

Since a fuzzy set may contain elements with zero degree membership and elements with one degree membership, then we can consider the concept of a crisp (classical) set to be a special case of the more general concept of a fuzzy set.

2.5 Representation of a Fuzzy Set

There are two methods to represent fuzzy sets:

1. A set of ordered pairs representation : a fuzzy set A in U may be represented as a set of ordered pairs of a generic element u and its membership value, that is

$$A = \{(u, \mu_A(u)) | u \in U\}$$

When U is discrete A is commonly written as

$$A = \sum_{i=1}^{m} \mu_A(u_i)/u_i = \mu_A(u_1)/u_1 + ... + \mu_A(u_m)/u_m$$

In this equation the summation sign does not represent arithmetic addition, it represents the collection of all points $u \in U$ with the associated membership function $\mu_A(u)$.

Example: Let A = integer close to 10, then one possibility to define this fuzzy set is as follows:

A = 0.1/7 + 0.5/8 + 0.8/9 + 1/10 + 0.8/11 + 0.5/12 + 0.1/13

The following can be noted for this fuzzy set

- The integers not explicitly shown all have membership values equal to zero.
- The membership values were chosen by a specific individual; except for the unity membership value when u = 10, they can be modified based on our own personal interpretation of the phrase "close."
- The membership function is symmetric about u = 10, because there is no reason to believe that integers to the left of 10 are close to 10 in a different way than are integers to the right of 10. But again, we are free to make other interpretations.

2. Functional representation: In this representation functional description is used to represent fuzzy sets. An example is the functional description of a trapezoidal-shaped fuzzy set, as

$$\mu_A(u) = \begin{cases} \dfrac{u - a1}{a2 - a1} & If \quad a_1 \le u \le a_2 \\ 1 & If \quad a_2 \le u \le a_3 \\ 1 - \dfrac{a3 - u}{a3 - a4} & If \quad a_3 \le u \le a_4 \end{cases}$$

2.6 Basic Characteristics of Fuzzy Sets

We now introduce some important characteristics associated with the fuzzy sets. These characteristics are illustrated in Figure 2.1 and explained below.

1. The support of a fuzzy set A within a universe of discourse U is the crisp set that contains all elements of U that have nonzero membership in A, that is,

$$\text{supp}(A) = \{u \in U \mid \mu_A(u) > 0\}$$

2. The core of a fuzzy set A within a range U, is the crisp set that contains all elements of U that have the membership of unity in A, that is,

$$core\,(A) = \{u \in U \mid \mu_A(u) = 1\}$$

note that the core of a fuzzy set is a subset of its support.

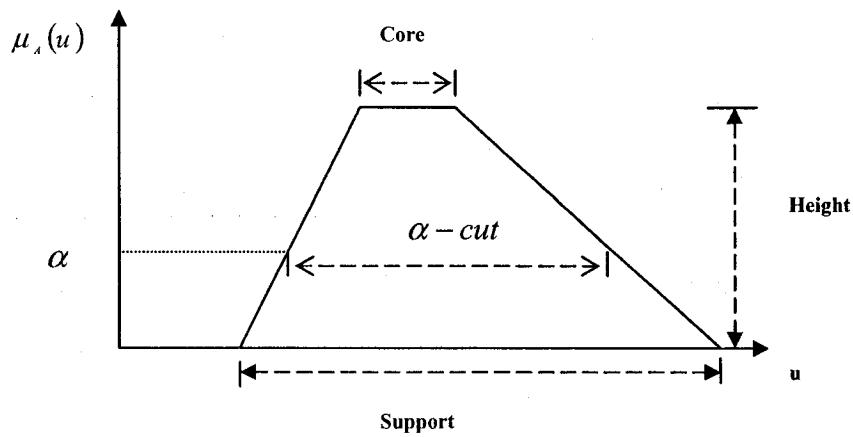

Figure 2.1: The support, alpha-cut, core and height of a fuzzy set

3. The height of a fuzzy set A is the largest membership grade obtained by any element in that set, that is,

$$h(A) = \sup_{u \in U} \mu_A(u)$$

A fuzzy set with a height equal to 1 is called normal and with $h(A) < 1$ subnormal.

4. An α-cut of a fuzzy set is a crisp set A_α that contains all the elements in U that have membership values in A greater than or equal to α, that is,

$$\alpha - cut(A) = \{ u \in U \quad | \mu_A(u) \geq \alpha \}$$

The core of a fuzzy set is an α-cut with α = 1.

Chapter 3

A Module Selection Scheme in the High Level Synthesis using Fuzzy Logic

3.1 Introduction

Module selection is one of basic architectural synthesis tasks that allows optimizing the cost of a target circuit under a real time constraint. Adding the area factor to the optimization problem changes the working domain from one dimension (time) to two-dimensions (time and area). However, solving this problem by the best selection of modules from a complex library of modules remains unresolved.

Most of the techniques for high level synthesis of real time architectures use a specific hardware module for a given type of functional units to optimize the design area. Although by using a specific module for each type of functional units reduces the complexity of the synthesis process, it may prevent a designer from meeting the design goal. On the other hand, having various choices of the modules to be used during the synthesis process is more flexible in the high-level synthesis; it provides the designer the ability to find a more optimized design. In this chapter a new framework for the module selection in high-level synthesis is proposed. The proposed framework employs a fuzzy rule base and fuzzy theory to find the *module set* that satisfies a given time constraints while at the same time tends to minimize the total design cost in terms of the area. However, the meaning of the term *module set is* the complete set of modules that are

selected from the components library (*CL*) to implement the design. This set may include no or many instances of a given module that exists in the *CL*.

3.2 Previous Work

During the design space exploration process, the designers seek to find an optimized implementation for a given behavioural specification. In the SEHWA algorithm [33] effort is devoted to select the modules by analyzing the time/area design space. This selection is performed at the first step during the high-level synthesis followed by the scheduling and process allocation. This algorithm allows the use of either fast or slow module in a design, but a simultaneous use of both fast and slow modules is not considered.

Gajski in [34] perform the selection of modules prior to the scheduling and process allocation. Then, the scheduling and process allocation is carried out and the results are evaluated. If the constraints are not met, the set of modules is changed by an if-then rule base system. This iterative process is only partial, since for each selection of the modules, a complete design has to be generated and evaluated, which is a time consuming process.

MSSR algorithm [35], attempts to solve the scheduling, process allocation and module selection at the same time. The algorithm is based on an iterative improvement and structured in three procedures. The procedure *OPSL*, which is the third procedure in the algorithm structure, is used to select an appropriate module from the associated library to be mapped to each operation in a DFG. This procedure starts with modules that have the largest area. Then, it iteratively investigates modules with smaller area until the

time constraint is violated. Depending on the given time constraint, this approach may need to explore the possible solutions.

In [20], an algorithm for module selection using unrestricted module library is proposed. The algorithm starts by initial module selection, and then the scheduler tries to meet the time constraints with this selection. If the scheduler does not succeed, the module selection has to be reviewed until a correct selection is made. The algorithm formulates the module selection problem using the ILP formulation and iterates the module selection, which is a very time consuming process, before finding the best schedule

Sllame and Drabek in [17] used a genetic algorithm for the scheduling, allocation and module selection. Genetic algorithms are global probabilistic search techniques that start from an initial population of generated potential solutions to a problem, and gradually evolve towards better solutions through a repetitive application of genetic operators such as crossover and mutation. In this approach, each chromosome in the initial population consists of two parts: the priorities of nodes, and the possible modules of each type of functional units. The priority of each node in a chromosome is determined by the number of nodes succeeding that node in given DFG, which is a weak priority function that will lead to inefficient schedules; hence, it will affect the module selection process.

3.3 The Proposed Module Selection Scheme

In this section, a scheme for the module selection problem is presented. In this scheme, each module is associated with a utility value, which represents the usefulness of a

module for a given design goal. Ideally, the utility values of modules are either 0 or 1. The design using those modules with the utility value of 1 should be of the highest quality. However, in reality, the usefulness of a module is dependent on the other modules that can be used. The module may be present in good design which optimizes a certain goal and/or part of a bad design that does not satisfy the design goal. Accordingly, we allow the utility value of a module to be any real number between 0 and 1 to handle the ambiguity in estimating the usefulness of a module.

In order to construct a general schedule based on utility assignment, we borrow some techniques from the fuzzy theory. In particular, the modules and their respective utility values are modeled as a fuzzy set with respect to the corresponding functional unit. For a functional unit f and its eligible module set M_f, let $\mu_f(m) \in [0,1]$, $\forall m \in M_f$, describe a utility value of module m.

Consider an application with only addition and multiplication operations, that is only two types of functional units *f1,f2*. Assume that there are 5 possible modules of adders ADD={add1, add2,..., add5} and 5 possible modules of multipliers MUL={mult1, mult2,...,mult5}. Let $\mu_{f_i}(m)$ be the utility value of module m for the functional unit f_i. Note that $m \in$ ADD for f_1 and $m \in$ MUL for f_2. It follows that $\mu_{f_i}(m) \in [0, 1]$ and can be considered as a membership function of f_i with respect to eligible module set M_{f_i} (See Figure 3.2).

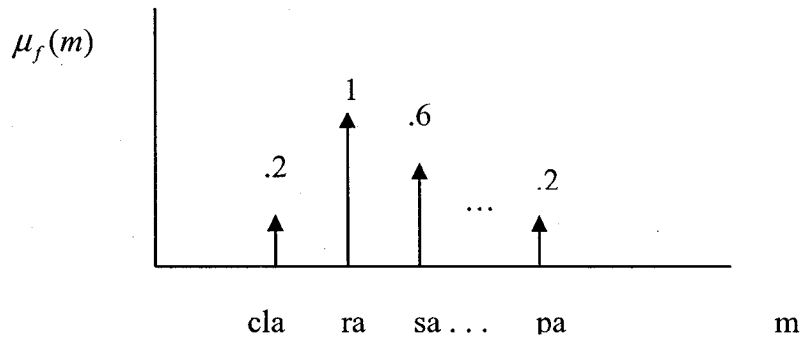

Figure 3.1: A utility memberships for some possible modules of addition functional unit

The module selection scheme starts *first* by reading the following inputs: (1) initial schedule produced from list-based scheduler that already satisfies the constraints on the maximum number of functional units allowed in each control step; (2) the required completion time (total latency) of the final design; (3) a component library with initial utility values for each of the available modules. *Next,* based on the input information, the latency as well as the area of the given schedule are calculated by using the fuzzy addition operation given by

$$\mu_{A+B}(z) = \bigvee_{z=x+y} (\mu_A(x) \wedge \mu_B(y))$$

where $\vee$ and $\wedge$ denote max and min operations respectively, A and B are two fuzzy sets.

Both the latency and area resulting from the fuzzy addition operation are fuzzy sets. In these fuzzy sets each latency element and area element is associated with utility degree (membership), this is obvious because both the fuzzy latency and the fuzzy area are calculated based on the fuzzy sets representing the available modules of each functional unit. Notice that based on the fuzzy addition operation, each module can contribute to the individual elements within the supports of the latency fuzzy set and the

area fuzzy set. The amount of contribution from a module to the elements of each of the two fuzzy sets varies during the fuzzy addition calculations, the number of references $freq_{(t,a)}(m)$ made by each element of the latency and area fuzzy sets to a certain module is recorded. This record is very important in the proposed scheme, since it will be used to calculate the positive contribution and the negative contribution of a module in a certain design in order to update its utility.

If a particular module contributes a great deal to an unacceptable latency and area values, the utility of that module should be decreased. On the other hand, if a module contributes to a great deal to a highly acceptable latency and area value, the module's utility value should be increased.

The statistics of a module usage $freq_{(t,a)}(m)$ for each pair latency and area values are used to scale the acceptability of the corresponding design, giving the module's contribution to that design. Based on this idea, we now develop a heuristic to compute the relative adjustment of the utility values of the modules. The adjustment is then applied to update the previous utility value. The utility value adjustment process is repeated until no further adjustment is need. The final utility values of the modules obtained through this process are compared with a threshold value as an acceptable utility values. The modules with utilities greater than this threshold are selected as a primary set to implement the system, this primary set is very small compared to *CL;* hence, it is easy to test every possible combination to find the best one. However, any combination in this set is acceptable and can be used to implement the system. The various steps of the proposed scheme are depicted in the block diagram of Figure 3.1.

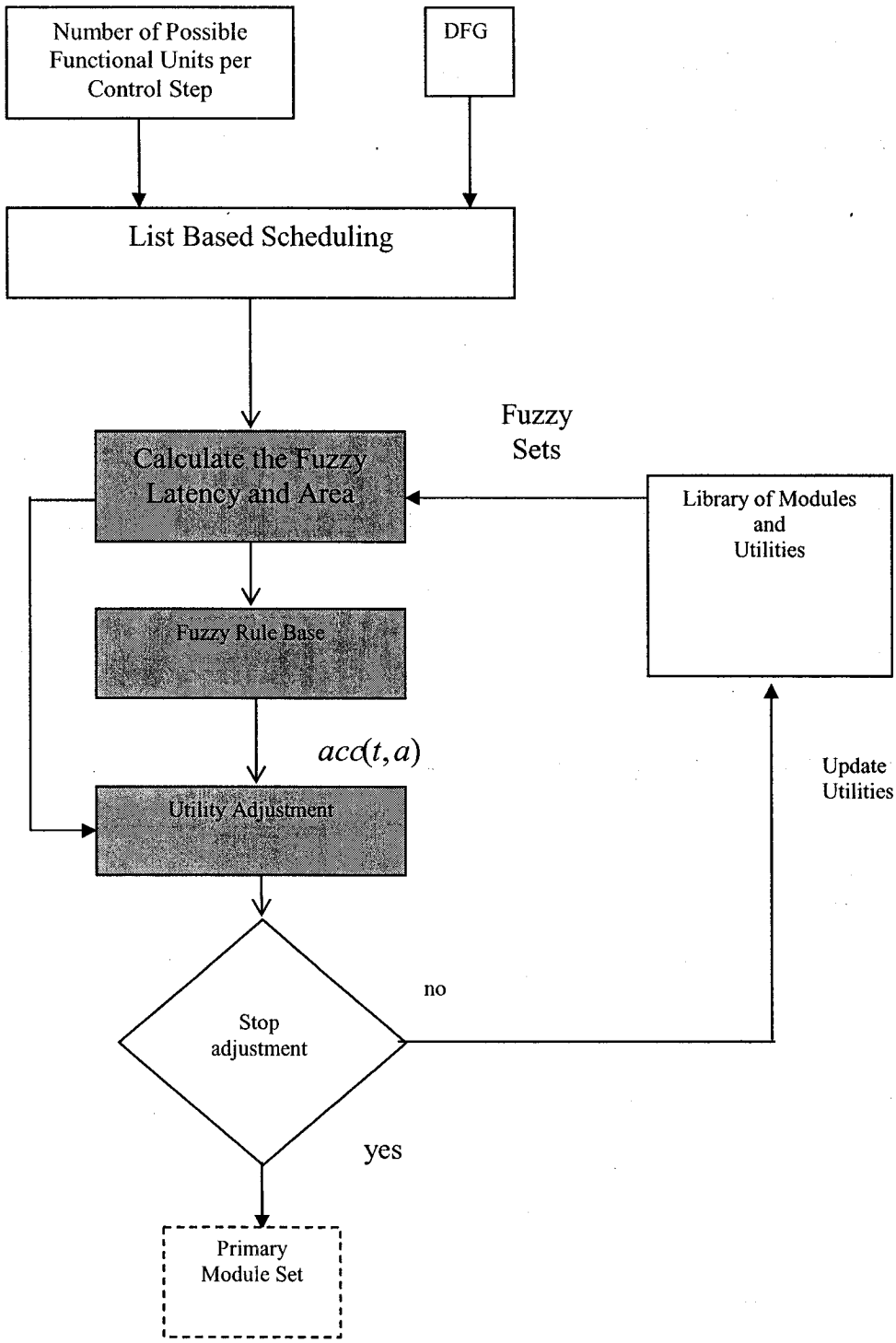

Figure 3.2: The proposed module selection scheme

3.3.1 Fuzzy Rule Base System to Infer the Acceptability of the Design

In the proposed scheme, a fuzzy rule base system is formulated to infer the acceptability of a design and it consists of four membership functions for each of the two variables latency and area (see Table 3.1). The given latency constraints and the expected highest area constitute the range of the latency and area linguistics variables, respectively. The acceptability *acc (t, a)* of a design depends both on the latency as well as the area, and it should range between 0 (not acceptable at all) to 1 (fully acceptable). While determining the acceptability of design, the fuzzy rule base gives preference to the area over the latency in the sense that any latency within the given latency constraint is acceptable as long as the latency corresponds to a minimum area. Figure 3.2 represents the membership functions for each of the latency, area, and acceptability.

There is a large list of shapes that can be used to represent a membership function. However, in practice two types of membership functions, namely triangle or trapezoidal functions, are commonly used to describe membership of a linguistic variable, and these two shapes are proven to be the best shapes to represent human reasoning. However, the proposed fuzzy rule base is just a kind of cost function and can be formulated based of the designer believe.

The fuzzy rule base has two inputs representing the total latency and the total area of design that a certain module can contribute to, and one output value between 0 and 1 representing the acceptability of the corresponding design.

TABLE 3.1: ACCEPTABILITY RULE BASE FOR THE AREA AND LATENCY

Area\ Latency	Low	Medium	High	Very high
Low	VH	VH	H	M
Medium	VH	H	M	L
High	H	H	L	L
Very high	M	L	VL	VL

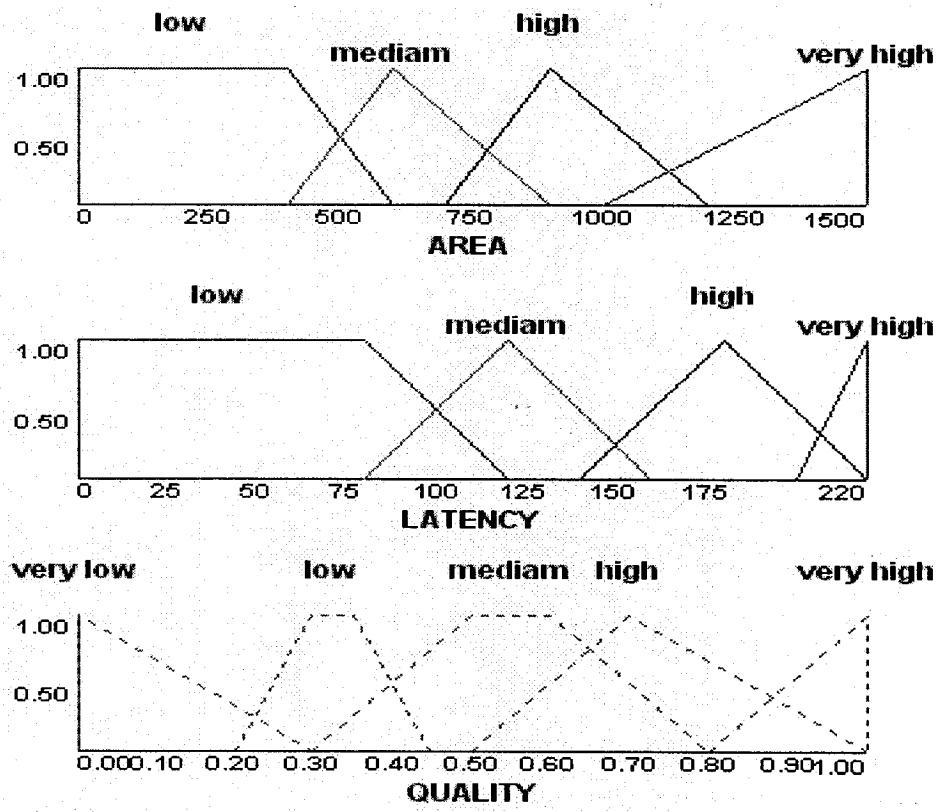

Figure 3.3: Membership functions of area, latency, and acceptability linguistic variables

3.3.2 Utility Adjustment

The utility values of modules should reflect the usefulness of the modules towards a design goal. However, the initial assignment of utility values may not satisfy the given design goal. The utility adjustment scheme is used to modify the utility values in order to determine the most appropriate utility assignments. In other words, we attempt to give high utility values to modules which contribute a great deal to most highly acceptable latency and area pairs and assign low utility values to the modules contributing to latency and area with low acceptability values. Set the number of references to a module m in a highly acceptable design be represented by $freq_{(t,a)}(m)$, and assume a threshold for the acceptability of a design as $acc_{threshold}(t,a)$. The positive contribution of m to the designs to the designs that have high acceptability is computed by

$$\eta_+(m) = \sum_{\forall(t,a)\ni acc(t,a)\geq acc_{threshold}(t,a)} freq_{(t,a)}(m) acc\ (t,a)$$

as was mentioned before $acc(t,a)$ for latency and area pair is calculated using fuzzy rule base system which was explained in Section 3.3.1. A higher $\eta_+(m)$ value indicates that using m can potentially lead to system with higher acceptability values. Similarly, the negative contribution of m is given by

$$\eta_-(m) = \sum_{\forall(t,a)\ni acc(t,a)\geq acc_{threshold}(t,a)} freq_{(t,a)}(m)(1-acc\ (t,a))$$

From $\eta_+(m)$ and $\eta_-(m)$, an adjustment of the utility value for each module of a functional unit say f can be estimated as

$$adj_f(m) = \frac{\eta_+(m)-\eta_-(m)}{\eta_+(m)+\eta_-(m)}$$

if $adj_f(m)$ is negative, a module tends to cause more bad latency and area pairs, then $\mu_f(m)$ should be decreased. On the other hand, if $adj_f(m)$ is positive, $\mu_f(m)$ is increased.

In our experiments, the $adj_f(m)$ is applied as following to calculate new $\mu_f(m)$, denoted by $\mu^{new}{}_f(m)$ and given as

$$\mu^{new}{}_f(m) = \begin{cases} \mu_f(m) + \mu_f(m) \times adj_f(m) & if\ 0 < adj_f(m) \le 1 \\ \mu_f(m) + \mu_f(m) \times \dfrac{adj_f(m)}{2} & if\ -1 \le adj_f(m) < 0 \end{cases}$$

Since the value of $adj_f(m)$ is always between [-1,1], if $adj_f(m)$ equals 1, the value of $\mu_f(m)$ is doubled and if $adj_f(m)$ equals -1, $\mu_f(m)$ is reduced to one-half. If $adj_f(m)$ is between (-1,0], the change is proportional to $\mu_f(m)$ by half of $adj_f(m)$, if $adj_f(m)$ is between (0,1), the change is proportional to $\mu_f(m)$ by $adj_f(m)$. After the adjustment in the utility value of all modules is made, all $\mu^{new}{}_f(m)$ are normalized with respect to the highest one. If the difference between $\mu^{new}{}_f(m)$ and $\mu_f(m)$ from the previous iteration is not appreciable for every *m,* the adjustment is no longer needed.

The normalized $\mu_f(m)$ produces a relative utility among all the modules eligible for implementing *f.* After successively updating $\mu_f(m)$, the utility values of some modules may converge to zero. A module whose utility becomes zero (or very close to zero) indicates that its contribution to obtaining superior design is not as significant as other modules. Such a module is then be excluded from the primary module set. A

primary module set is formed by inspecting the rest of the modules and choosing the ones with the utility values higher than a given threshold.

3.4 An Example of Module Selection

In this section, we consider an example of a DSP filter, a *fifth order elliptic wave filter* (EWF) [12], in order to demonstrate the process of module selection using the proposed scheme. The initial list- based schedule with a maximum number of functional units given as two multipliers and two adders per control step is illustrated in Table 3.2. It is clear from the table that operations are still not mapped to their modules that will execute them. An example of different modules available in *CL* for the two functional units and the corresponding initial utility values is given in Table 3.3.

Given a constraint on the latency as 452 ns, after applying the proposed scheme to the above benchmark problem, the highest utility of adder functional unit change from Adder5 in the first iteration to Adder1 in the final iteration, and also that of multiplier functional unit change from Mult5 in the first iteration to Mult1 in the final iteration.

Assume that we select the modules having a utility value of 0.9 or higher. This leads to select Adder1, Adder2, as possible modules of addition functional unit and only Mult1 for the multiplication functional unit. These selected modules represent the primary module set. It is clear that primary module set is much smaller than *CL,* and it contains only the modules that can lead to a highly acceptable design; hence, it is easy to explore all the designs that use a combination of these modules in the primary module set. Any combination of these modules in the primary set will lead to acceptable design.

TABLE 3.2: INITIAL LIST-BASED SCHEDULE WITH GENERIC RESOURCES

Control step				
1	+	+		
2	+			
3	+			
4	+			
5	*	*		
6	+	+		
7	+	+		
8	+	*	*	
9	+	+		
10	+	+		
11	+	+		
12	+	*	+	*
13	*	+	*	+
14	+	+		
15	+	+		
16	+	+		

TABLE 3.3: COMPONENTS LIBRARY AND THE ASSOCIATED UTILITIES FOR EACH COMPONENT MODULE

Component name	Cost (gates)	Delay (ns)	Initial utility
Mult1	2300	58	0.2
Mult2	2400	44	0.3
Mult3	2600	36	0.4
Mult4	2900	29	0.6
Mult5	3500	27	1
Add1	50	26	0.5
Add2	100	20	0.6
Add3	200	14	0.65
Add4	250	10	0.7
Add5	400	6	1

TABLE 3.4: THE UTILITY VALUES FOR ADDER AND MULTIPLIER MODULES OF *EWF*

Func. unit	Iteration	Add1	Add2	Add3	Add4	Add5
Adder	**First**	.5	.6	.65	.7	1
	final	**1**	**.92**	.15	.1	.05
Func. unit	Iteration	Mult1	Mult2	Mult3	Mult4	Mult5
Multiplier	**First**	.2	..3	.4	.6	1
	Final	**1**	.4	.25	.08	0

3.5 Experimental Results

The proposed module selection scheme has been applied to two well-known benchmark problems, namely, *fifth order elliptic wave filter (EWF)* [12] and *discrete cosine transform (DCT)* [6]. The experimental results demonstrate the effectiveness of the proposed module selection scheme in terms of reducing the design cost of the two benchmarks. The proposed module selection scheme is applied to each problem with a view that the target designs is implemented using a pipelining architecture technique, that is, each control step in the schedule is implemented as a pipeline stage. After finding the primary module set, all the possible selection combinations in this primary module set are tested and the best combination in terms of the area cost is then selected.

In all the experiments, we compare the designs obtained using a large number of modules in *CL* with the designs obtained using a reduced number of components in *CL*. The reduced set includes only the fastest and the slowest components of each type functional units that are found in the complete *CL* (i.e. the large set). Keeping all design parameters other than the component library constant, for all experiments, we can remark that the lowest area designs are those obtained with modules selected using a complete set of modules. The graphs in Figures 3.4 and 3.5 illustrate the area-time curve with different time constraints for *EWF* and *DCT,* respectively.

In order to compare the results of module selection obtained from the proposed scheme with those obtained from the module selection scheme in [17], we applied the two schemes to *the fifth-order elliptic wave filter* and *fourth-order lattice filter*, given the same specifications, Figure 3.6 and Figure 3.7 illustrate the time-area curve with different

time constraints. It is clear that for the two benchmark problems that the proposed framework is better in finding a design with smaller cost in term of area for the same time constraints.

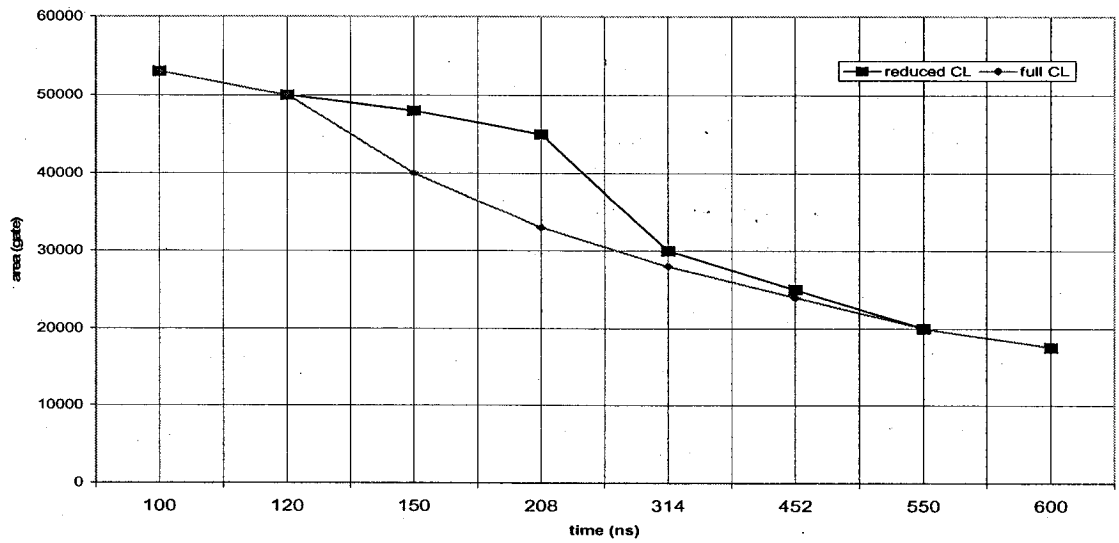

Figure 3.4: Area versus latency for EWF with module selection using complete CL and reduced CL, with resource set (+2,*3).

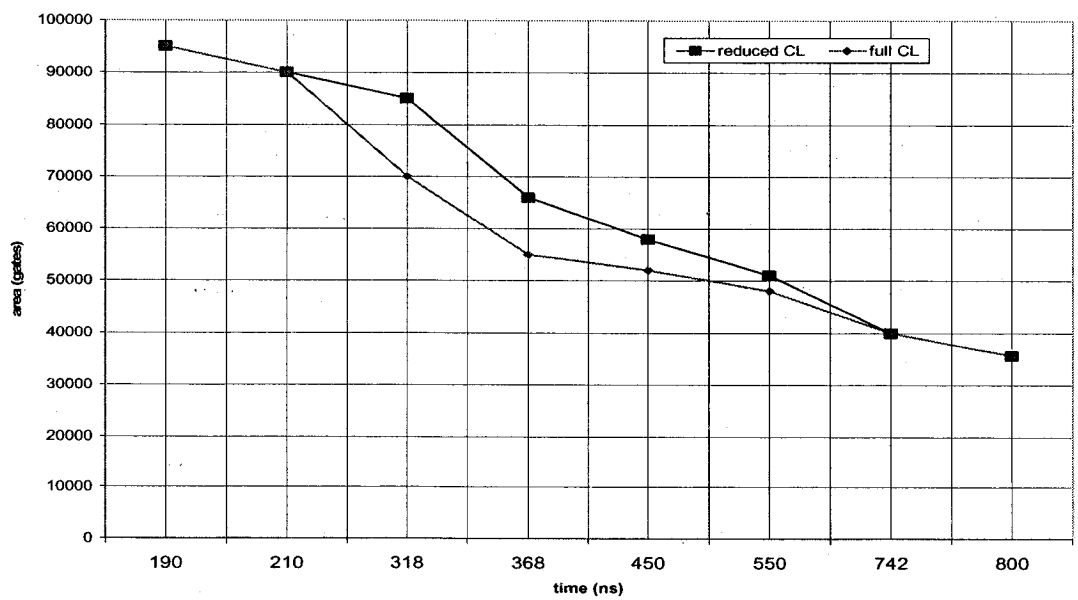

Figure 3.5: Area versus latency for DCT with module selection using complete CL and reduced CL, with resource set (+2,*3).

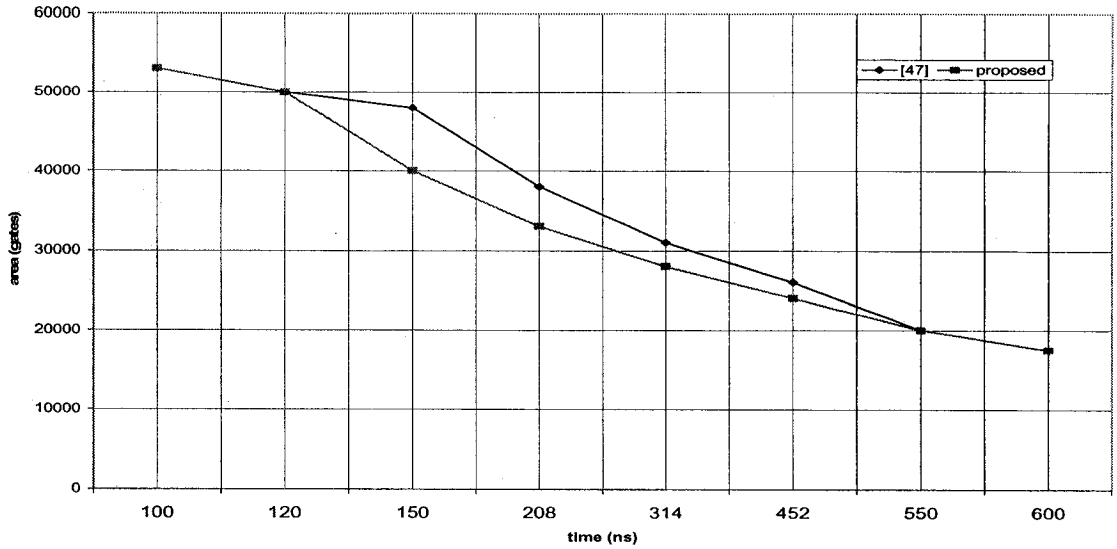

Figure 3.6: Area versus latency for fifth-order EWF with module selection using the proposed scheme and the scheme given in [17], with resource set (+2,*2).

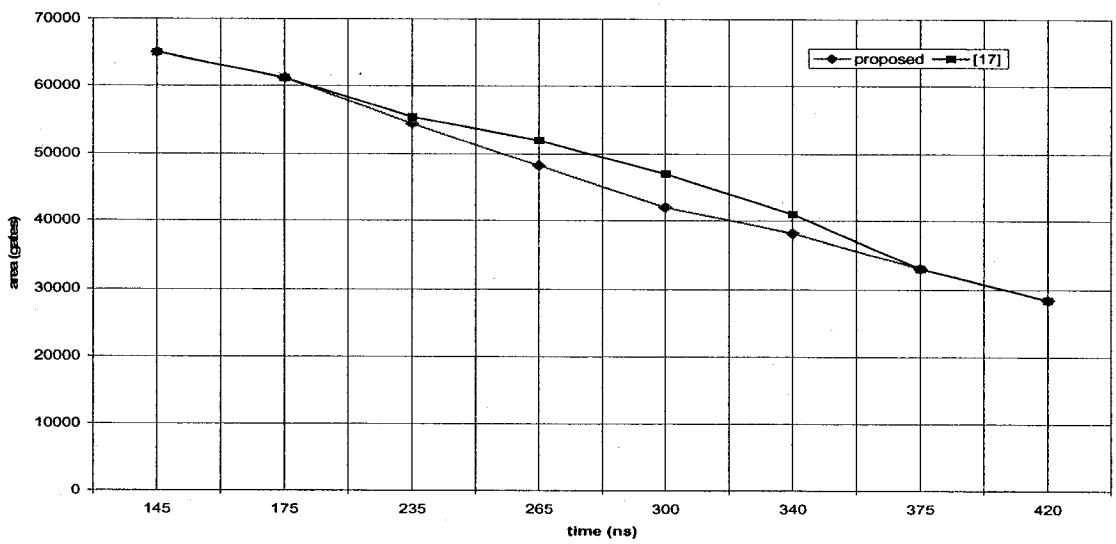

Figure 3.7: Area versus latency of fourth-order lattice filter with module selection using the proposed scheme and the scheme given in [17], with resource set (+2,*1).

3.6 Summary

In chapter 3, we have presented a module selection scheme in high-level synthesis based on fuzzy rule base and fuzzy theory. The proposed scheme has used the utility measure to model the degree of usefulness of a module meeting some design goal. The possible

modules for each type of functional units and their associated utilities have been represented as fuzzy set. Then, fuzzy addition operation has been used to calculate the latency and the area of a given schedule obtained by using list-based scheduler [17]. The calculated latency and the area thus obtained are fuzzy sets. A fuzzy rule base is used to infer the acceptability for each latency and area pair and then based on this result; the contribution of each module to good designs or bad designs is calculated. If a module contributes a great deal to highly acceptable designs, its utility is increased. On the other hand, if a module contributes significantly to a highly unacceptable design, its utility is decreased. A utility adjustment scheme that aims to update the associated utility for each module has been proposed. In the proposed scheme an adjustment function, which is based on the positive or negative contribution of the modules on the acceptable designs has been derived. The module utility adjustment process is iteratively continued and stopped only when the variations in the utility value of modules are not appreciable. The modules with utilities higher than certain given threshold are then being selected as a primary module set. This set is very small compared to the component library *CL* allows to explore all possible module selections from this primary module set. It is to be noted that any possible selection of modules from this primary set has been result in an acceptable design from the point of view of both the latency and the corresponding area

Many experiments with two benchmark problems have been conducted. These experiments demonstrate that the proposed scheme of module selection is efficient in that it allows a designer to identify the modules that satisfy a given total latency with a minimum design cost (area).

Chapter 4

Process Allocation of DSP Data Flow Graphs onto Multiprocessor Systems

4.1 Introduction

Allocation is a task of determining generic resources (functional units) on which operational nodes are executed. It involves assigning operations and variables to hardware resources while trying to minimize the amount of hardware resources needed. It assumed that a particular of a generic resource could execute only one operation at a time. In particular, the process allocation includes two subtasks: one is to determine the number of generic resources used and the other to bind nodes to resources. A generic resource type may be, for instance, an adder unit, a multiplier unit, or an arithmetic logic unit (ALU) which is capable of performing multiple operations such as addition, multiplication, etc.

In order to the minimize of the number of hardware resources required for the implementation of a digital system, the operations (nodes) of a data flow graph (DFG) 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.

Hardware functional units may be homogenous or heterogeneous. The functional units are homogenous if all of them are of the same type and execute the same set of

different types of operations. Each type of operations has different computational delay. Since such functional units usually require more hardware resources to support multiple operations, homogenous functional units, if used alone in the synthesis process, may result in inefficient designs. ALU functional units are used for homogenous synthesis. On the other hand, the heterogeneous functional units vary in the operations they support and have different execution delays. However, the use of heterogeneous functional units in the synthesis process results in a more efficient design in terms of the area.

A number of approaches for the process allocation problem have been developed. The most straightforward approach is to set no limit on the number of functional units available and then use them for scheduling and allocation [12]. This approach would for sure result in a design with large area, and thus it is not a suitable approach if the area of a design has to be optimized. Another approach [22] uses branch-and-bound technique to search through every possible allocation of process in the DFG to the corresponding functional units, keep track of the best solution so far in terms the area, and cut off the others. This technique requires a large amount of computations and thus results in a large processing time. Another technique uses the mathematical formulation using integer linear programming [20]; it involves assigning a variable to each possible allocation of a process in the DFG to its corresponding functional unit and then finds a solution that minimizes the area of the corresponding design. Due to the exhaustive search required by this technique, it is augmented with heuristics to limit the search space. This heuristic does not necessarily lead to an optimal solution in terms of the area.

In this chapter, a new process allocation technique that leads to a multiprocessor system with different configurations of for a rate-optimal scheduling of a fully static DFG

is proposed. A set of configurations is obtained by using the conventional heterogeneous functional units and/or general purpose functional units (e.g. ALU). A fuzzy rule base is then used to evaluate the multiprocessor configurations in terms of their areas and sharing ratios of the corresponding systems. The multiprocessor configurations obtained through the proposed scheme are compared with those obtained by using some existing approaches in terms of area and the sharing ratios of the configurations.

4.2 Time Scheduling

In order to find the best assignment of the processor to the nodes in a DFG given an iteration period T, the allocation process needs to know the time schedule for all the nodes, that is, the control steps at which an operations is to be scheduled. This information is important in the allocation process to know the amount of parallelism inherent in the given DFG. For this purpose, in this section a time schedule is built by using a modified version of the technique given in [36], which employs an iterative procedure based on the node mobility. As in the technique [36], the earliest and the latest firing times at which each node can be scheduled to fire, are iteratively calculated. The node mobility in the schedule or the range of control steps at which the corresponding node can be scheduled is equal to the difference between its calculated latest and earliest firing 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 [37]. Given a DFG, the time scheduling can be built using the longest path matrix Q^f [36, 37]. The entries in this matrix represent the length of the longest simple path between each pair of nodes $v_i v_j$ given by

$$Q_{ij}^{f} = \max_{all\ Pv_iv_j} len[Pv_iv_j[$$

The earliest firing time (EFT) and the latest firing time (LFT) for a node v_j relative to a reference node v_i are, respectively, given by

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

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

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

$$EFT(v_j) = \max_{\forall previous\ refernces} \left(EFT(v_j/v_i)\right)$$

$$LFT(v_j) = \min_{\forall previous\ refernces} \left(LFT(v_j/v_i)\right)$$

Hence, the mobility or the flexibility of any node is given by.

$$M(v_j) = LFT(v_j) - EFT(v_j)$$

This mobility always satisfy the condition that

$$M(v_j) \geq 0$$

The schedule is built by selecting a reference-node and by calculating the mobilities 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. 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 in the implementation of the final system. The chosen node is scheduled to fire at a control step that would results in a minimum number of functional units required. Taking into account that the target multiprocessor system can contains both heterogeneous and/or general purpose functional units, a significant modification for the original scheduling technique [36] is introduced. We defined the level of a control step to be the summation of sub-levels of the different types of operations and given by $level = level_{type_1} + level_{type_2} + \ldots$. The choosing of the best firing time is done by examining all the control steps within its mobility to find the control step having the minimum total level as a primary key or the minimum sublevel ($level_{type}$) as a secondary key. This modification is very significant to reduce the number of heterogeneous functional units needed. Due to the new firing time of this 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 is calculated. A new node is chosen for scheduling and the process is iteratively repeated until all the nodes are scheduled. In the modified version of the scheduling technique a priority in choosing the next node to be scheduled is given to the node that is of the same type of the current reference node and directly connected to it (predecessor or successor). This priority is useful to give more chance for the nodes of the same type to be allocated to the same processor and thus, the communication overhead is reduced. The result of the time schedule is a set of firing times of all nodes. The time scheduling algorithm is given below

Time scheduling algorithm

1. Calculate the minimum iteration period for rate-optimal schedules using well-known algorithms. For non-rate-optimal schedules select a higher iteration period.
2. Calculate the longest path matrix Q^f.
3. Take the input nodes as the reference nodes and schedule them first with respect to each other, and then, update the EFTs and LFTs of all the remaining nodes with respect to the input nodes.
4. Schedule all nodes that have zero mobility with respect to the input node especially those of zero local flexibility. There is no need to update the EFTs and LFTs of all the remaining nodes after scheduling zero-mobility nodes.
5. Calculate the current schedule range or the mobility for each node of the remaining non-scheduled nodes, which is the difference between the LFT and EFT for the node.
6. Choose the next node or the target node for scheduling according to the following priority:
   a. A node with zero mobility.
   b. A node that has minimum current mobility. If more than one node has minimum current mobility, chose from these nodes the one that is a predecessor or successor to the current reference node.
   c. A node that is a predecessor or successor of the reference node and it is of the same type.

d. A node that is a predecessor or successor of any scheduled node and it is of the same type.

e. Any first available node.

7. Find the best firing time position through the scheduling range of the target node within the duration of its minimum firing period in the corresponding scheduling matrix. The best firing time value must satisfy the following:

   a. Minimum highest level.

   b. If more than one value results in the same minimum possible highest level, choose the one that results with a minimum lowest sublevel with respect to the current reference node ($level_{(type)}$)

   The best firing time position found is the time schedule of the target node.

8. Set the target node to be new reference node

9. Update the earliest and latest firing times of all the remaining non-scheduled nodes, except when the current reference node has a zero mobility.

10. Go step 6 until all nodes have been scheduled.

4.3 The Process Allocation Algorithm

Given the time schedule derived using the scheduling algorithm presented in Section 4.2 for a given DFG with iteration period T, the allocation process assigns nodes to the functional units according to their type. The result of this process is the allocation matrix, which specifies as to which nodes are executed by a functional unit. The proposed algorithm has the flexibility of choosing the best configuration of the target

multiprocessor system, that is, the choice of execute the DFG on multiprocessor system with general purpose functional units (e.g. ALUs) or heterogeneous functional units or a hybrid of two types of functional units. The decision of preferring one configuration over the other possible configuration is made based on two designs factors, the area and the sharing ratio of the target multiprocessor system. In this context, the area of a multiprocessor configuration is the summation of the areas for all the functional units in the system. On the other hand, the sharing ratio is estimated by finding the ratio of total computations encapsulated in a certain DFG relative to the product of the number of functional units and the iteration period of multiprocessor system and it is given by

$$sharing\ ratio = \frac{total\ computaionan\ in\ DFG}{T \times number\ of\ functional\ units}$$

The sharing ratio is very important in the multiprocessor systems where the communication overhead between the functional units is very considerable. The higher the sharing ratio is the lower the communication overhead between the processors. In the homogenous multiprocessor systems the sharing ratio is high relative to that of the heterogeneous multiprocessor systems, since the functional units in a homogenous multiprocessor, generally, can execute a larger number of nodes in each iteration period, since in order to execute these nodes no communication overhead between the processors required.

In order to prefer one multiprocessor configuration over the others, we developed a fuzzy rule base to infer the quality of a multiprocessor configuration in terms of the area and the sharing ratio of that configuration. The fuzzy rule base consists of two input linguistic variables (area and sharing ratio) and one output linguistic variable (quality). The number and shape of membership functions as well as the rule set of the system can

be quickly formulated based on the designer's preference on area and sharing ratio of the system. Table 4.1 shows the formulated rule base and Figure 4.1 illustrates the membership functions of the fuzzy rule base system. Figure 4.2 shows a block diagram of the allocation scheme.

TABLE 4.1: QUALITY RULE BASE FOR THE AREA AND SHARING RATIO,

Area\ sharing ratio	Low	Medium	High	Very high
Low	VH	VH	H	M
Medium	VH	H	M	L
High	H	H	L	L
Very high	M	L	VL	VL

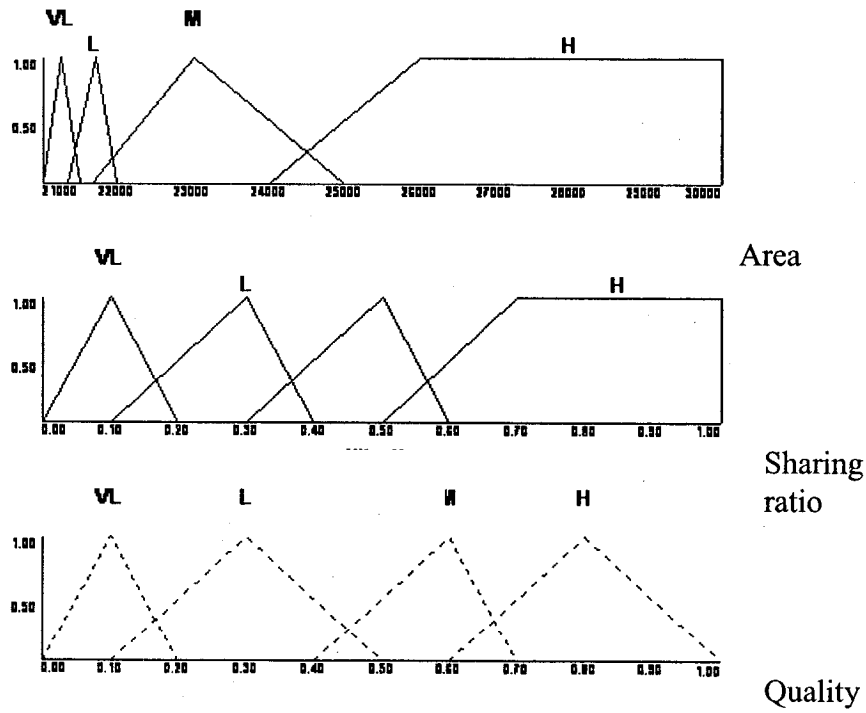

Figure 4.1: Area, sharing ratio, and quality membership functions

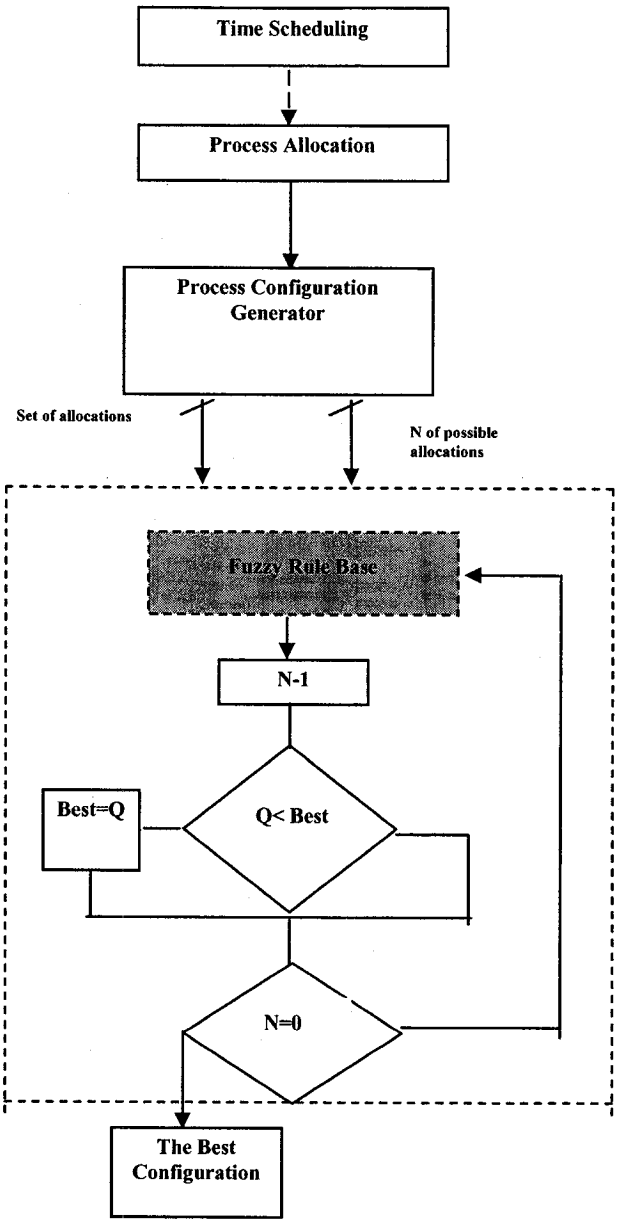

Figure 4.2: The proposed process allocation scheme

The allocation algorithm can be described in terms of the following steps.

Process allocation algorithm

1. Create an $NVL \times T$ matrix. where NVL is the number of virtual lines (VL). A VL could be an ALU, MULTIPLIER or ADDER and T is the iteration period, and for rate optimal scheduling $T=T_0$.

2. Sort the nodes of a given DFG in descending order of their computational delay.
3. Start with first node in the sorted list and assign it to the lowest index in the allocation matrix that can accommodate this node; if there is no space, increase the number of virtual lines VLs by unity and fit this node in the new created line, remove this node from the sorted list.
4. For empty list go to step 5, otherwise go to 2.
5. start the configuration generator phase
    a. Create empty set β that contains the set of the all generated configurations.
    b. Set N=1 replace each VL in the processor assignment matrix into ALU. Update β with the first allocation generation.
    c. Search for the VLs that are occupied by the same type of operations and replace these VLs by the functional unit (ADDER, MULTIPLIER) that can execute the same type of operation, rename these lines to be VL_{SINGLE} and replace the rest of the VLs by ALUs. Update β with the new configuration, N=N+1.
    d. EXRACT and MERGE. EXRACT is a function used to extract operations of a certain type from available VL and assign these operations into onto a functional unit of the same type. MERGE is a function used to combine VL_{SINGLE} with the extracted VL.
6. Infer the quality for each multiprocessor configuration in β using fuzzy rule base
7. Select the best configuration having the highest quality.

4.5 An Example

In this section, we consider an example of a DSP filter, a *fifth-order elliptic wave filter* (EWF), in order to demonstrate the allocation process using the proposed scheme. The DFG of this filter is shown in Figure 4.2. The firing times for all the nodes in the DFG of this benchmark problem are found by using the rate-optimal scheduling algorithm explained in Section 4.2. Then, the proposed process allocation algorithm is applied to this problem to assign functional units to the nodes. The multiprocessor systems with different configurations ranging from a fully homogenous system to a fully heterogeneous one are generated using the proposed scheme.

This DFG has an iteration period bound of 16 time units. In this example, the specifications for the functional units used to generate different configurations are as shown in Table 4.2. All the multiprocessor configurations that result from the configuration generator with the overall area, sharing ratio and the inferred qualities for these configurations are shown in Table 4.3. The best configuration is obtained when 2 units of ALUs, 1 unit of ADD, and 1 unit of MUL are used.

The allocation matrix for this particular configuration is shown in Table 4.4. This multiprocessor configuration results in a less area compared to a fully homogeneous, and in a larger sharing ratio compared to a fully heterogeneous multiprocessor. Figure 4.4 shows that moving from a fully homogenous to a fully heterogeneous configuration results in a reduced the area.

TABLE 4.2: SPECIFICATIONS OF FUNCTIONAL UNITS

Type	Time (+)	Time (*)	Area (gates)
ADD	1	-	200
MUL	-	2	7500
ALU	1	2	8000

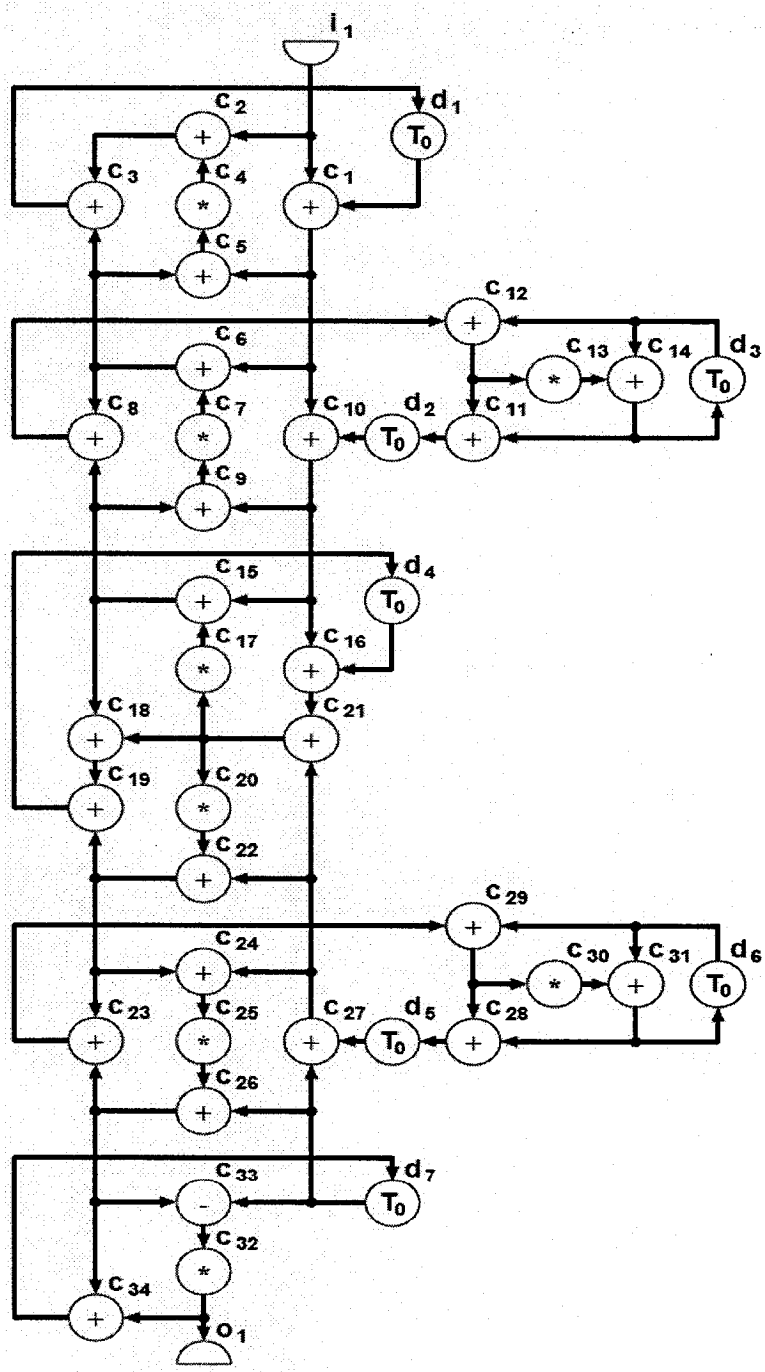

Figure 4.3: DFG of a fifth-order elliptic wave digital filter

TABLE 4.3: DIFFERENT CONFIGURATIONS AND THE CORRESPONDING QUALITIES

Configuration	Area (gates)	Sharing ratio	Quality
4xALU	32000	0.65625	0.3000
3xALU, 1xADD	24200	0.65625	0.7626
1xMUL, 2xALU, 1xADD	23700	0.65625	0.8000
2xMUL, 1xALU/, 2xADD	23400	0.5250	0.6220
3xMUL, 3xADD	23100	0.43745	0.5592

TABLE 4.4: ALLOCATION MATRIX OF THE FIFTH-ORDER ELLIPTIC WAVE DIGITAL FILTER CORRESPONDING TO THE BEST ALLOCATION

P/CS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MUL					20	20			7	7			32	32	30	30
ALU	2	11	18		17	17	16	19	25	25	8	34	24	15	15	5
ALU	12	35	28	22			23	10	21		27	6	4	4	3	14
ADD	33	29						26				9	13	31		

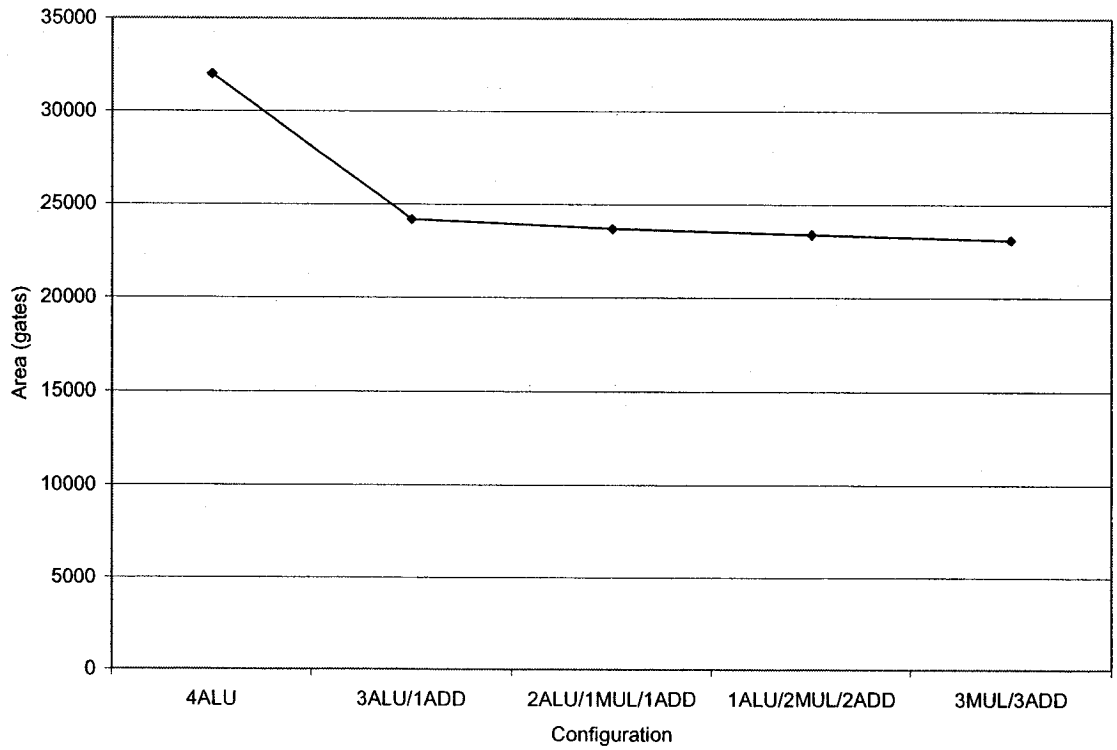

Figure 4.4: The total area versus configuration ranging from a fully homogenous to a fully heterogeneous multiprocessor

4.7 Summary

In this chapter, a new process allocation technique that leads to a multiprocessor system with different configurations for the rate-optimal scheduling of a DFG has been proposed. The time schedule is built iteratively based on the node mobility, and then, the firing times obtained from the time schedule for each node is used during the allocation process to find the allocation matrix. The proposed technique provides the designer with more flexibility to find the best multiprocessor configuration for a given DFG. A set of configurations has been introduced by using general-purpose functional units and/or heterogeneous functional units. Fuzzy rule base has been used to infer the quality of different multiprocessor configurations in terms of the area and the sharing ratio of the configurations. The best configuration has been shown to be the one that allocates the nodes in a DFG to a combination of heterogeneous and general-purpose functional units rather than a fully homogenous or fully heterogeneous configuration. The hybrid multiprocessor configuration brings about a trade off between the overall area and resource sharing.

Chapter 5

Scheduling of DSP Data Flow Graphs with Processing times Characterized by Fuzzy Sets

5.1 Introduction

Generally, the design flow of digital systems can be divided into three phases: high level synthesis, logic synthesis and layout synthesis. The higher the level of synthesis, the more the degrees of freedom, however, at higher level, only a little information about the final circuit parameters is available to guide the design. Wrong decision made at this phase could be very expensive. If the original specification, which can be verified only after the placing and routing, is not met, the whole design process might have to be restarted.

One of the most important issues in synthesis of a DSP task is to obtain a good schedule so that to reduce the total computation time and the total area. To implement and execute a system efficiently, one has to determine an execution order or schedule for all the operations in the data flow graph of a given DSP application. Further, to construct an effective execution order, the knowledge of the characteristics of the functional units should be available. However, during the high level synthesis the actual characteristics of the system components are imprecise, and ignoring the impreciseness in the synthesis may cause an overly expensive and time consuming design.

Scheduling is one of the basic tasks in 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 carry out the task of the application under certain constraints on the available hardware resources [15-17]. Over

the years researchers have tried to come up with various kinds of solutions to the scheduling problem. Several algorithms have been put forth and each having its own advantages and disadvantages. Some approaches use heuristics such as artificial intelligence methods [23, 24], while other techniques attempt to find an optimum solution via an integer linear programming approach [20]. None of these approaches consider the impreciseness in the design attributes.

In order to have a better design of a DSP application for multiprocessor implementation, the impreciseness in the design needs to be properly treated during the synthesis tasks in order to reduce the risk of over/under estimating the final design. In this chapter, the impreciseness of the computational times of the functional units, which accrues in the high level synthesis, is considered by representing the computational time as fuzzy sets instead of as crisp numbers.

In recent years a great deal of research has been conducted in the area of scheduling DSP data flow graphs onto multiprocessing systems. Most of the static scheduling techniques assume the worst case or the best case computational delay of the functional units used in the target architecture. This assumption is not realistic, since some of the computational times of the DSP tasks may be imprecise due to the fact that during early design phases, the characteristics of the final implementation of the functional units are not be known. In this chapter, the impreciseness of the processing times of the functional units is taken into consideration by considering them as fuzzy sets, and then using fuzzy arithmetic to build the time schedule [38]. The range of control steps (mobility) which represents the possible firing times of a task is determined, and a

fuzzy rule base is employed to infer the degree of selectivity in choosing a certain control step within this range.

5.2 Preliminary

To introduce a meaningful ordering of fuzzy numbers, we may extend the lattice operations min and max on real numbers to corresponding operations on fuzzy numbers, fuzzy max and fuzzy min. A partial ordering for comparable two fuzzy numbers A and B is defined as $A \leq B$ iff fuzzy min (A, B) = A or, alternatively $A \leq B$ iff fuzzy max(A, B) = B.

To handle the arithmetic operations [39] between two fuzzy numbers defined on $A * B$ with membership function $\mu_{A*B}(z)$, the extension principle can be used: $\mu_{A*B}(z) = \bigvee_{z=x*y} (\mu_A(x) \wedge \mu_B(y))$ where $\vee$ and $\wedge$ denote max and min operations respectively. In order to calculated the addition of two fuzzy numbers, the following equation is used: $\mu_{A+B}(z) = \bigvee_{z=x+y} (\mu_A(x) \wedge \mu_B(y))$. Figure 5.1 demonstrate a graphical result of adding two fuzzy numbers.

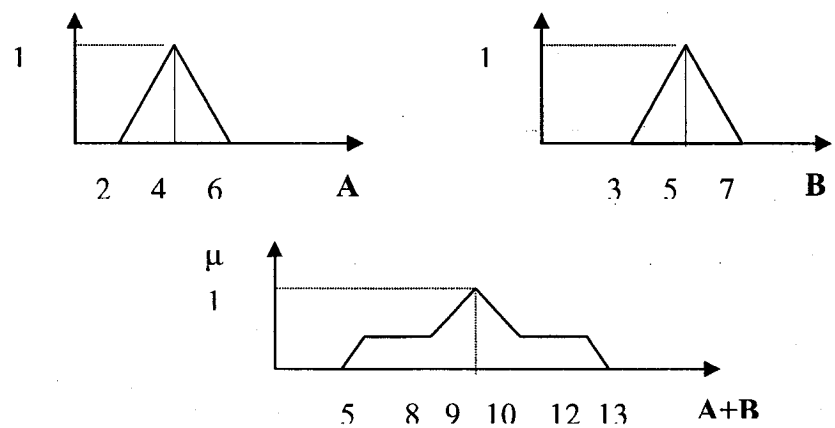

Figure 5.1: A fuzzy addition operation

5.3 Building the Time Schedule using Fuzzy Arithmetic

Since we deal with imprecise computational times of the functional unit, using crisp numbers to determine the iteration period bound are not valid. The iteration period bound T_0 ,is the minimum time between producing successive outputs. Fuzzy arithmetic is used to estimate the iteration period bound that result in a fuzzy set, thus reflecting the fuzzy nature of the computational delay. For a cyclic DFG T_0 is given by

$$T_0 = \underset{C \in circuits}{fuzzy\ \max} \left\lceil \frac{D_C}{N_C} \right\rceil$$

where D_C and N_C are, respectively, the summation of the fuzzy computational times and the total number of delays in the circuit C.

The scheduling algorithm found in chapter 4 is modified, and fuzzy arithmetic operations used to build the time schedule. The time schedule is 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. The node mobility in the schedule or the range of control steps at which the corresponding node can be scheduled is equal to the difference between its calculated latest and earliest times. These earliest and the latest firing times are found relatively to a reference node and are the result of intra and inter iteration precedence constraints. Since the computational time of the nodes is a fuzzy number, fuzzy arithmetic operations are used to handle all the calculations

Given a data flow graph of a DSP application, the time scheduling can be built using the longest path matrix Q^f . The entries of this matrix represent the length of the longest path between each pair of nodes $(v_i v_j)$ which given by

$$Q_{ij}^{f} = \underset{all\ Pv_iv_j}{fuzzy\ \max}\ (fuzzy\_len\lfloor P_{v_iv_j} \rfloor)$$

where $fuzzy\_len\lfloor P_{v_iv_j} \rfloor$ is calculated using fuzzy addition operation for the all fuzzy computational times of the nodes in the path $P_{v_iv_j}$. The earliest firing time (EFT) and the latest firing time (LFT) for a node v_j relative to that of a reference node v_i are, respectively, given by

$$EFT\left(v_j/v_i\right) = FT(v_i) + \underset{all\ Pv_iv_j}{fuzzy\ \max}\ len\lfloor P_{v_iv_j} \rfloor = FT(v_i) + Q_{ij}^{f}$$

$$LFT\left(v_j/v_i\right) = FT(v_i) - \underset{all\ Pv_jv_i}{fuzzy\ \max}\ len\lfloor Pv_jv_i \rfloor = FT(v_i) - Q_{ji}^{f}$$

where $FT(v_i)$ is the scheduled firing time of node v_i. *LFT and EFT* are fuzzy numbers. 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. Thus, EFT and LFT of node v_j are, respectively, given by

$$EFT(v_j) = \underset{all\ i<j}{fuzzy\ \max}\left(EFT(v_j/v_i)\right)$$

$$LFT(v_j) = \underset{all\ i<j}{fuzzy\ \min}\left(LFT(v_j/v_i)\right)$$

Thus, the mobility or the flexibility in the schedule of any node is given by.

$$M(v_j) = fuzzy\_sub\ (LFT(v_j), EFT(v_j))$$

For more simplicity in the calculation the resulting mobility is defuzzified. All the nodes of the DFG is first put in a set of non-scheduled nodes, and a schedule is built by selecting a reference-node and by calculating the flexibility of all 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 balance between minimizing the number of functional units required and minimizing the latency of the system by using fuzzy rule base and examining all control steps within the mobility of the node. Due to the new fixed schedule firing time of this 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 is 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 scheduling of a DFG, if during a given iteration more than one node is found to have the same minimum mobility, then as to be explained in Section 5.4, such nodes are treated as a special cases for their scheduling.

5.3.1 Preferred Firing Time of a Node

As seen earlier for the case of non-zero mobility, each firing position from the earliest firing time to the latest firing time for the target node has to be considered as a possible firing time of that node. The best firing time for the node to be scheduled is chosen based on the impact this will make on the overall latency and area of the system to be designed. The effect of selecting a certain firing position on the latency and the area of a system can be estimated by finding the latency_ratio and the area_ratio and defined as

$$latncy\_ratio = \frac{\Delta latency}{latency_{current}}$$

$$area\_ratio = \frac{\Delta area}{area_{current}}$$

where $\Delta latency$ ($\Delta area$) is the difference between the values of the latency (area) in the current iteration and the previous one.

The area in this context is equivalent to the *level* of each control step, which is the number of nodes that will be eventually fired at that control step. The highest level of schedule will determine the number of functional units required in the system to be designed.

These two ratios are calculated for each possible control step, and then a fuzzy rule base is used to infer the degree of the selectivity for each possible control step, and the one with the highest degree of selectivity is chosen. The fuzzy rules can be quickly formulated based on designer preference on area and latency of the system. The membership functions for latency_ratio and area_ratio can vary based on the designer's point of view. Depending on the designer formulation of the fuzzy rule base, membership functions as well as the rule set can be identified. For example, the membership functions may range from 0 to 0.2, 0.15 to 0.5, 0.4 to 1 for low, medium, and high latency_ratio, respectively, and that from 0 to 0.3, 0.25 to 0.6, 0.5 to 1, for low, medium, and high area_ratio respectively, and all can have triangle shape. In this paper, the fuzzy rule based is selected as shown in Table 5.1, the mesh surface of the selected membership functions and fuzzy rules is shown in Figure 5.3.

5.4 Time Scheduling Algorithm

The result of the time schedule is a set of firing times of all nodes. These firing times are used later in the processor assignment operation. The proposed algorithm can be applied to both homogenous and heterogeneous multiprocessors. For a heterogeneous multiprocessor system, the time schedule is done by generating synchronized distinct

schedules, one for each type of processors (functional units). The proposed fuzzy scheduling algorithm can be described as follows.

1. Use fuzzy arithmetic to calculate the minimum iteration period for rate optimal schedules. Find the fuzzy longest path matrix Q^f.
2. Take the input node as the reference node and schedule it first to fire at control step zero. Then, update the earliest and latest firing times of all remaining nodes with respect to the input node.
3. Calculate the fuzzy earliest and latest firing times of all the remaining nodes with respect to the input node.
4. Schedule all the nodes that have zero mobility (with respect to input node). There is no need to update the earliest and latest firing times of remaining nodes after scheduling such a zero-mobility node.
5. Using fuzzy arithmetic, Calculate the current schedule range or fuzzy mobility for each of the remaining non-scheduled nodes.
6. Defuzzify the resulting mobility for each unscheduled node. One node has to be chosen at this step of the algorithm (except when more than one node result in a zero-mobility, these are chosen together and scheduled to fire at the only and the fixed control step in their mobility). Choose the next node or the target node for scheduling according to the following priority:
   a. A node with zero mobility.
   b. A node that has minimum current mobility. If more than one node has minimum current mobility, chose from these nodes the one that is a predecessor or successor to the current reference node.

c. A node that is a predecessor or successor of the reference node.

d. A node that is a predecessor or successor of any scheduled node.

e. Any first available node.

7. Use fuzzy rule base to find the best firing time position through the scheduling range of the target node. The fuzzy rule base chooses the best firing time based on the impact it will make on the latency and area of the system to be designed. The area can be measured by the number of nodes schedule to fire in the same control step (the level of the control step). This level also fixed the number of functional units that will be required later in the processor assignment operation.
8. The best firing time position found is the time schedule of the target node.
9. Set the target node to be new reference node
10. Update the earliest and latest firing times of all the remaining non-scheduled nodes. This updating is not required in the case when the current reference node has zero mobility.
11. Go step 6 until all nodes have been scheduled.

5.5 An Example

In this section, we apply the proposed scheduling algorithm to a well-known benchmark problem of a second-order IIR filter. The target system is heterogeneous with imprecise processing times. In this experiment, the processing times of adders and multipliers are obtained from Texas Instruments [40]. The confidence interval of computational time in nano-seconds for the adder is (10, 14, 24) which represent the (best, typical, worst) times. While the confidence time for the multiplier is (14, 20, 30).

The proposed scheduling algorithm is applied to the second order IIR filter depicted in Figure 5.4. The fuzzy set boundaries of the iteration period is (24, 34, 54), the matrix Q^f which represents the fuzzy longest path is shown in Figure 5.5. Using the proposed time scheduling algorithm, the defuzzified time schedule is shown in Table 5.2.

TABLE 5.1: SELECTIVITY RULE BASE FOR AREA_RTATIO AND LATENCY_RATIO

	Latency_ratio		
Area_ratio	Low	Med	High
Low	H	H	M
Med	M	M	L
High	M	L	L

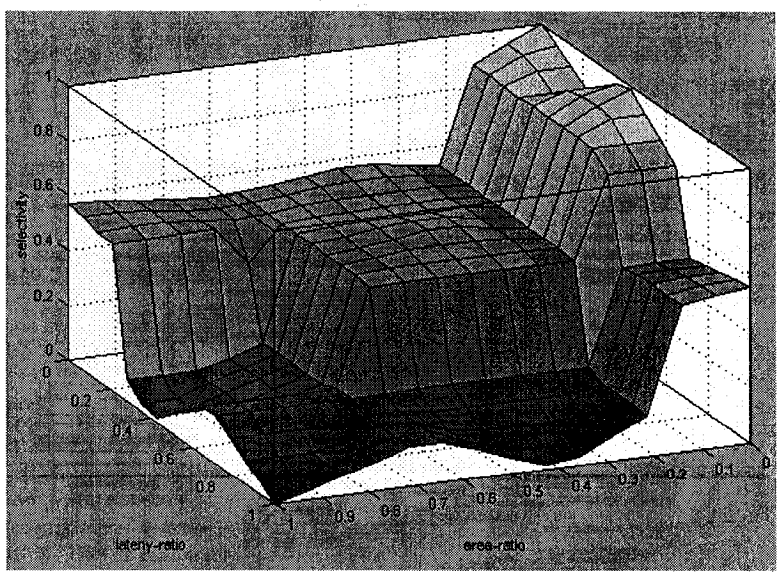

Figure 5.2: The surface of the fuzzy rule based

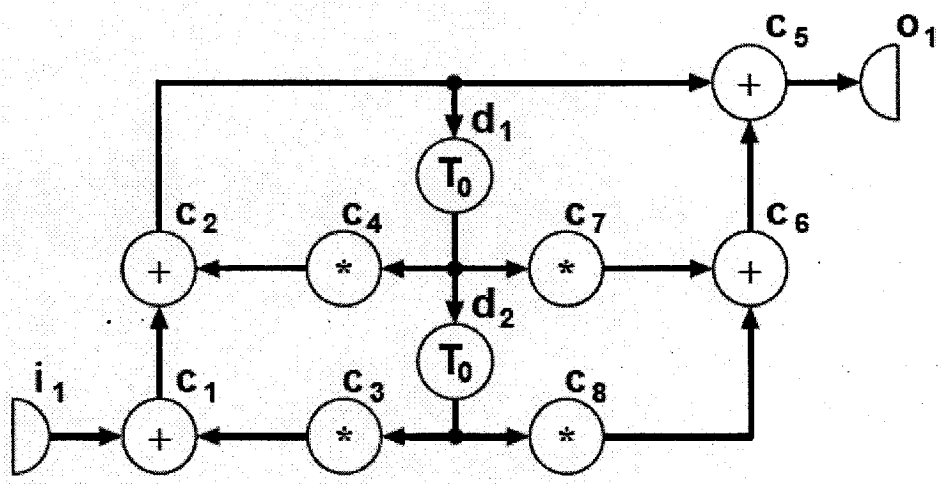

Figure 5.3: A DFG of a second-order filter

$$Q^{f}_{FUZZY} = \begin{bmatrix}
(-74,-30,30) & (10,14,24) & (-88,-40,0) & (-34,-6,24) & (-34,-6,24) & (20,28,48) & (-88,-40,0) & (30,42,72) & -\infty & (40,56,96) \\
(-84,-34,6) & (-30,0,30) & (-98,-54,-24) & (-44,-20,0) & (-44,-20,0) & (10,14,24) & (-98,-54,-24) & (20,28,48) & -\infty & (30,42,72) \\
(14,20,30) & (24,34,54) & (-74,-20,30) & (-20,14,54) & (-20,14,54) & (34,48,78) & (-74,-20,30) & (44,62,102) & -\infty & (54,78,126) \\
(-70,-14,36) & (14,20,30) & (-84,-34,6) & (-30,0,30) & (-30,0,30) & (24,34,54) & (-84,-34,6) & (34,48,78) & -\infty & (44,62,102) \\
-\infty & -\infty & -\infty & -\infty & -\infty & (14,20,30) & -\infty & (24,34,54) & -\infty & (34,48,78) \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & (10,14,24) & -\infty & (20,28,48) \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & (14,20,30) & -\infty & (24,34,54) \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & (10,14,24) \\
0 & (10,14,24) & (-88,-40,0) & (-34,-6,24) & (-34,-6,24) & (20,28,48) & (-88,-40,0) & (30,42,72) & -\infty & (40,56,96) \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & (-96,-56,-40) & -\infty
\end{bmatrix}$$

Figure 5.4: The fuzzy longest path matrix of the second order filter.

TABLE 5.2: THE TIME SCHEDULE OF THE SECOND ORDER FILTER

Node number	1	2	3	4	5	6	7	8	9	10
Scheduling time	0	16	-22	-5	11	32	-42	48	0	64

5.6 Comparison with the Conventional Approaches

The proposed approach is much more efficient than the conventional scheduling approach (which assumes the worst or the best case of the computational time), because of its ability to consider all possible computational times of the functional units and find the schedule in one step, instead of finding it for every possible combinations of computational times. In the conventional approach, an exhaustive method can be used to consider each possible combination of the computational times; however, in this case, a new schedule should be found for each possible combination of computational times which is time consuming.

We have experimented the proposed algorithm on different benchmarks, the processing times of the adders and the multipliers were obtained from Texas instruments [40]. The confidence intervals of these processing times are the same as was explained in Section 5.5. These benchmark problems were also scheduled using the conventional scheduling approach assuming the worst case or the best case of processing time. The result latency obtained for each benchmark problem applying both the proposed algorithm and the conventional algorithm are shown in Table 5.3. One can easily conclude that using the worst case or the best case of processing time leads to over estimate or under estimate the schedule latency, while the proposed leads to more realistic schedule latency. Since using imprecise timing can give more information about the latency of the schedule and, thus, more accurate estimation can be done.

TABLE 5.3: THE LATENCY RESULTS FOR DIFFERENT BENCHMARK PROBLEMS USING THE PROPOSED APPROACH VS. THE CONVENTIONAL APPROACH [21]

Benchmark problem	Worst case	Best case	Fuzzy
All-pole lattice filter	180	122	138
Volterra filter	294	204	225
Fifth-order elliptic filter	288	240	266
Fourth-order Jaumann filter	194	134	148

5.8 Summary

In this chapter, a new algorithm for scheduling of DSP cyclic data flow graphs onto multiprocessor systems has been proposed. In this algorithm, the computational times of the functional units have been characterized as fuzzy sets in order to obtain a more realistic scheduling. Fuzzy arithmetic has been used to build the time schedule efficiently, and a fuzzy rule base has been employed to select the best firing time of a node within its mobility.

The proposed approach has been experimented on a well-known DSP filters, and the time schedule is seen to provide a good compromise between the latency and the area. The proposed technique gives more accurate estimation of the schedule latency. It can provide a good initial design and thus can be expected to reduce the number of design refinement iterations.

Chapter 6

Conclusions and Future work

6.1 Conclusion

This thesis has been concerned with devising techniques for efficient designs of DSP applications by incorporating the impreciseness arising from the lack of the exact knowledge of the relationship between the architecture of the functional units and their final implementations and the performance of the overall design during the high-level synthesis. The impreciseness considered in this research could arise from different sources. It could be due to the fact that there exist various choices of off-the-shelf processor modules that can be used to implement a functional unit; hence, the decision as to which module should be chosen is ambiguous. Another source of impreciseness could be because of the fact that the characteristics of the functional unit may not be known during the synthesis stage. That is, since the exact implementation is unknown, the area and time characteristics of the functional unit remain imprecise.

To deal with the first source of impreciseness, in Chapter 3, a module selection scheme in a high-level synthesis based on fuzzy rule base and fuzzy theory has been presented. The proposed scheme has used the utility measure to model the degree of usefulness of a module meeting some design goal. The possible modules for each type of functional units and the associated utilities have been represented as fuzzy sets. Then, fuzzy addition operation has been carried out to calculate the latency and the area of a

given schedule obtained by using the list-based scheduler [16]. The latency and the area thus obtained are fuzzy sets. A fuzzy rule base has been used to infer the acceptability for each pair of latency and area and then based on this result, the contribution of each module to design is calculated. If a module contributes a great deal to highly acceptable designs, its utility is increased. On the other hand, if a module contributes significantly to highly unacceptable designs, its utility is decreased. A utility adjustment scheme that aims to update the associated utility for each module has been proposed. In the proposed scheme an adjustment function, which is based on the positive or negative contribution of the modules on the acceptable designs has been derived. The modules with utilities higher than certain given threshold are then selected as a primary module set. This set is very small compared to the component library allows to explore all possible module selections from this primary module set. It is noted that any possible selection of modules from this primary set has resulted in an acceptable design from the point of view of both the latency and the corresponding area

Many experiments with two benchmark problems have been conducted with a view that the target design is implemented using pipeline architecture. These experiments demonstrate that the proposed scheme of module selection is efficient in that it allows a designer to identify the modules that satisfy a given total latency with a minimum design cost (area). The results of the proposed module selection scheme have been compared with those obtained by using the module selection scheme of [17]. The proposed scheme has been shown to provide a design with smaller cost in terms of area than the one in [17] for the same time constraints in all the experiments.

In Chapter 4, a new process allocation technique that leads to different multiprocessor configurations for the rate-optimal scheduling of a DSP data flow graph has been proposed. The proposed technique provides the designer with more flexibility in finding the best multiprocessor configuration for a given DFG. A set of configurations has been obtained by using general-purpose and heterogeneous functional units. A fuzzy rule base has been used to infer the quality of different configurations in terms of the area and the sharing ratio of each configuration. It has been shown that the best configuration is the one that allocates the nodes in a DFG to a combination of heterogeneous and general-purpose functional units during the allocation process. The proposed algorithm has been applied to well-known benchmark problems, and it is found that a hybrid multiprocessor configuration can be used to bring about a trade off between the area and the resource sharing.

In Chapter 5, a new algorithm for scheduling of DSP cyclic data flow graphs onto multiprocessor systems has been proposed. In this algorithm, the computational times of the functional units have been characterized as fuzzy sets in order to obtain a more realistic scheduling. Fuzzy arithmetic has been used to build the time schedule efficiently, and a fuzzy rule base has been employed to select the best firing time of a node within its mobility. The proposed approach has been experimented on well-known DSP filters, and the time schedule is seen to provide a good compromise between the latency and the area design. The proposed technique gives a more accurate estimation of the schedule latency. It can provide a good initial design, and thus, can be expected to reduce the number of design refinement iterations. The proposed approach provides a more accurate estimate of the latency than the one obtained by using the conventional scheduling approach (which

assumes the worst or the best-case computational time), because of its ability to incorporate all possible computational times of the functional units. Finally, a consequence of using all possible computational times of functional units in the proposed technique is the ability to find the final schedule in a single step.

6.2 Future Research Directions

In this thesis, each processor has been assumed to be capable of uniformly communicating with all other processors in the system. The topology of such architecture can be represented by a complete graph in which there is an edge between any two nodes. Further research need to be conducted to deal with the problem of scheduling on a pre-defined architecture whose topological graph is not complete. With this constraint, it is no longer possible to neglect the communication delays associated with the edges of the DFG, since these delays are not fixed and may not be determined without the knowledge of the processor assignment. As a matter of fact, in this case, the communication delay associated with an edge is dependent as to where the operations associated with the end nodes of that edge are scheduled. For future research, communication delays, synchronization needs and memory management should be considered in order to obtain more practice results. Fuzzy logic can be used to resolve the impreciseness in estimating the communication delay in a multiprocessor system. This impreciseness in the communication delay between a pair of nodes of a DFG can be described by associating a fuzzy set with the corresponding edge in the DFG.

REFERENCES

[1] 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, pp. 301-318, February 1990.

[2] A. Shatnawi, M.O. Ahmad, and M.N.S. Swamy, "Rate-optimal static scheduling of DSP graphs onto multiprocessors using circuit contraction," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 197-200, May 1995.

[3] P.D. Hoang and J.M. Rabaey, "Scheduling of DSP programs onto multiprocessors for maximum throughput," *IEEE Transaction on Signal Processing*, vol. 41, pp.858-888, February 1993.

[4] M.N.S. Swamy and K. Thulasiraman, *Graphs, Networks, and Algorithms*, New York: John Wiley & Sons, Inc., 1981.

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

[6] 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, pp. 386-389, M ay 1999.

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

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

[9] S.M. Heemstra de Groot. *Scheduling Techniques for Iterative Data-Flow Graphs.* PhD thesis, University of Twente, Enschede, 1990.

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

[11] C. H. Genotys and M.I. Elmasry, "A VLSI methodology with testability constraints," i*n Proc. 1987 Canadian Conference. VLSI*, Manitoba, Canada, October 1987.

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

[13] J. Lee, Y. Hsu, and Y. Lin, "A new integer linear programming formulation for the scheduling problem in data-path synthesis," in. *Proc. of the International.Conference. on Computer-Aided Design*, pp. 20-23, November 1989.

[14] P.G. Paulin and J.P. Knight, ``Force-directed scheduling in automatic data path synthesis," in *Proc. 24th Design Automation Conference,* Miami Florida, pp. 263-270, July 1987.

[15] C.Y. Wang and K. Parhi, "High level DSP synthesis using concurrent transformation, scheduling, and allocation," *IEEE Transaction. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 14, pp. 274-295, March 1995.

[16] A.M. Sllame and V. Drabek, "An efficient list-based scheduling algorithm for high-level synthesis of digital system," in *Proc. Euro Micro Symposium on Digital Systems Design,* pp: 316 – 323, September 2002

[17] A.M. Sllame and V. Drabek "*A design space exploration scheme for high-Level synthesis systems,*" in *Proc. of 36th International Conference Modelling and Simulation of Systems MOSIS '02,* Ostrava, Czech Republic, pp. 305-312, April 2002.

[18] G. M. Leive, *The design Implemenetaion and Analysis of an Automated Logic Sysntheis and Module Selection System,* PhD thesis, Carnegie-Mellon University, 1981.

[19] R. Jain and A. Parker, "Module selection for pipleined syntheis," in *Proc. of the Design Automation Conference,* New Jersey, United States, pp. 542-547, July 1988.

[20] M. Balakrishman and P. Marawedel, "Integrated scheduling and binding," in *Proc. of the 26th Design Automation Conference,* pp. 68-74, May 1989.

[21] C.H. Gebotys and M.I. Elmasry. "Optimum synthesis of high perofrmance architectures," *IEEE Journal of Solid-State Circuits,* vol. 27, no.3, pp. 389-397, March 1992.

[23] M. K. Dhodhi. "Datapath synthesis using a problem space genetic algorithm," *IEEE Trans. on Computer Aided Design of International Circuits and Systems,* vol. 14, no. 8, pp. 936-944, Augest 1996.

[24] E. Torbey and J. Knight "Performing scheduling and storage optimiztion simulatnsouly using genetic algorithms," in *Proc. of Midwest Symposium on Circuit and Systems,* Notre Dame IN, pp. 282-287, Augest 1998.

[25] H. Zimmeran, "Uncertainty modeling and fuzzy sets," To appear *in Proc. of Workshop on Modeling Uncertainty,* New York, July 2004.

[26] G. Klir, "Where do we stand on measure of uncertainty, ambiguity, fuzziness, and the like," *Fuzzy Sets and Systems,* vol. 24, pp.141-160, 1987.

[27] D. Dubois and H. Prade, *Possibility Theory,* New York: Plenum Press, 1988.

[28] L. Zadeh, "Fuzzy sets," *Informaton and control*, vol. 8, pp. 338-353, 1965.

[29] L. Zadeh, "Outline of a new approach to the analysis of complex systems and decision process," *IEEE Transactions on Systems*, Man and Cybernetics, vol. 1, pp.28-44, 1973.

[30] M. Sugeno and T. Yasukawa, "A fuzzy logic approach to qualitative modeling," *IEEE Transaction on Fuzzy Systems,* vol. 1, pp. 7-31, February 1993.

[31] J. Bezdek, "Cluster validity with fuzzy sets," *Journal of Cybernetics,* vol. 3, pp. 58-71, 1974.

[32] K. Demirly and I. Turksen, "Fuzzy logic based mobile robot localization with sonar data," in *Canada-Japan bilateral Workshop on Intelligent Manufacturing and Process Design*, Toronto, Canada, pp. 28-30 , April 1999.

[33] N. Park and A.C. Parker, "SEHWA: A program for synthesis of pipelines," *in Proc. 23rd Design Automation Conference*, New York, USA, pp. 595-601 1986.

[34] F. Brewer and D. Gajski, "Chippe: A system for constraint driven behavioral synthesis", *IEEE Transactions on CAD*, vol. 9, pp. 35-44, July 1990

[35] M. Ishikawa and G. De Micheli, "A module selection algorithm for high-Level synthesis," in *Proc. of the IEEE International. Symposium on Circuits and Systems*, pp. 1777-1780, April 1990.

[36] Ali Shatnawi, M.O. Ahmad, and M.N.S. Swamy, "Optimal scheduling of digital signal processing data-flow graphs using shortest-path algorithms," *The Computer Journal,* vol. 45, no. 1, pp.88-100, 2002.

[37] S.M. Heemstra de Groot, S.H. Gerez, and O.E. Herrmann, "Rate-optimal scheduling of recursive DSP algorithms based on the scheduling range chart," in *Proc. IEEE International Symposium* on Circuits and Systems, New Orleans, LA USA, pp. 1805-1808, May 1990.

[38] Awni Itradat, M.O. Ahmad, and Ali Shatnawi, "Scheduling of DSP data flow graphs with processing times characterized by fuzzy sets," to appear in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, Niagara Falls, Ontario, May 2004.

[39] A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic Theory and Application,* New York: Van Nostrand Reinhold, 1991.

[40] Texas Instruments. *The TTL data book*, volume 2, Texas instruments incorporation, 1985.