7-8-2003

CHESS: A Tool for CDFG Extraction and High-Level Synthesis of VLSI Systems

Ravi K. Namballa
University of South Florida

CHESS: A TOOL FOR CDFG EXTRACTION AND HIGH-LEVEL SYNTHESIS OF VLSI SYSTEMS

by

RAVI K NAMBALLA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: N.Ranganathan, Ph.D.
Murali Varanasi, Ph.D.
Abdel Ejnoui, Ph.D.

Date of Approval:
July 8, 2003

Keywords: High-Level Synthesis, Resource Optimization, Low Power Binding, CDFG Extraction, Tabu Search, Game Theory

DEDICATION

To My Mother

ACKNOWLEDGEMENTS

I would like to express gratitude to my major professor, Dr. N. Ranganathan, for his encouragement, guidance, support and friendship throughout my Master's program. Without his patience and his valuable suggestions, this thesis would not have been completed. I would also like to thank Dr. Varanasi and Dr. Abdel for guiding me as my committee members.

I would also like to thank Ashok Murugavel for his ideas and his help throughout my thesis work. I wish to thank Sarju Mohanty for providing me with his collection of related works in VLSI. I would also like to thank all members of VCAPP group for their help and support.

I really appreciate the invaluable support that I received from my brother without which this work would not have been possible. Also, I would like to acknowledge the support of my roommates and friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHESS: A TOOL FOR CDFG EXTRACTION AND HIGH-LEVEL SYNTHESIS OF VLSI SYSTEMS

RAVI K NAMBALLA

ABSTRACT

In this thesis, a new tool, named CHESS, is designed and developed for control and data-flow graph (CDFG) extraction and the high-level synthesis of VLSI systems. The tool consists of three individual modules for:(i) CDFG extraction, (ii) scheduling and allocation of the CDFG, and (iii) binding, which are integrated to form a comprehensive high-level synthesis system. The first module for CDFG extraction includes a new algorithm in which certain compiler-level transformations are applied first, followed by a series of behavioral-preserving transformations on the given VHDL description. Experimental results indicate that the proposed conversion tool is quite accurate and fast. The CDFG is fed to the second module which schedules it for resource optimization under a given set of time constraints. The scheduling algorithm is an improvement over the Tabu Search based algorithm described in [6] in terms of execution time. The improvement is achieved by moving the step of identifying mutually exclusive operations to the CDFG extraction phase, which, otherwise, is normally done during scheduling. The last module of the proposed tool implements a new binding algorithm based on a game-theoretic approach. The problem of binding is formulated as a non-cooperative finite game, for which a Nash-Equilibrium function is applied to achieve a power-optimized binding solution. Experimental results for several high-level synthesis benchmarks are presented which establish the efficacy of the proposed synthesis tool.

CHAPTER 1

INTRODUCTION

VLSI technology has advanced to a level where it would be extremely difficult to design digital systems starting at the transistor level or at the physical level. The increasing complexity of the designs and the ever growing competitiveness in the design market have made inevitable, the need to take the design process to much higher levels of abstraction where the design tradeoffs of time and efficiency could be carefully evaluated by the design engineer. This led to the automation of the design process based on a top-down methodology starting from the conceptualization of the design to its realization on silicon. Now, VLSI technology has gradually evolved to a point where the high-level synthesis of VLSI design systems has become more cost effective and less time consuming than the traditional method of designing everything by hand.

A typical VLSI design flow is shown in figure 1.1.. The first level of the design flow is the systems level specification, which is the most abstract form of representation of the design and mostly gives its description in plain English. The next level is the behavioral description which gives a functional description of the design while avoiding the structural details of the design. The RTL description, on the other hand, is composed of instances of modules such as adders, multipliers, registers, etc. that provide the structural details of the design. The process of translation of a behavioral description into a structural description is termed as High-Level Synthesis.

The process of synthesizing an RTL structure from the functional description during the high-level synthesis involves three phases:

- Allocation: determining the number of instances of each resource needed.
- Binding: assignment of resources to computational operations.
- Scheduling: timing of computational operations.

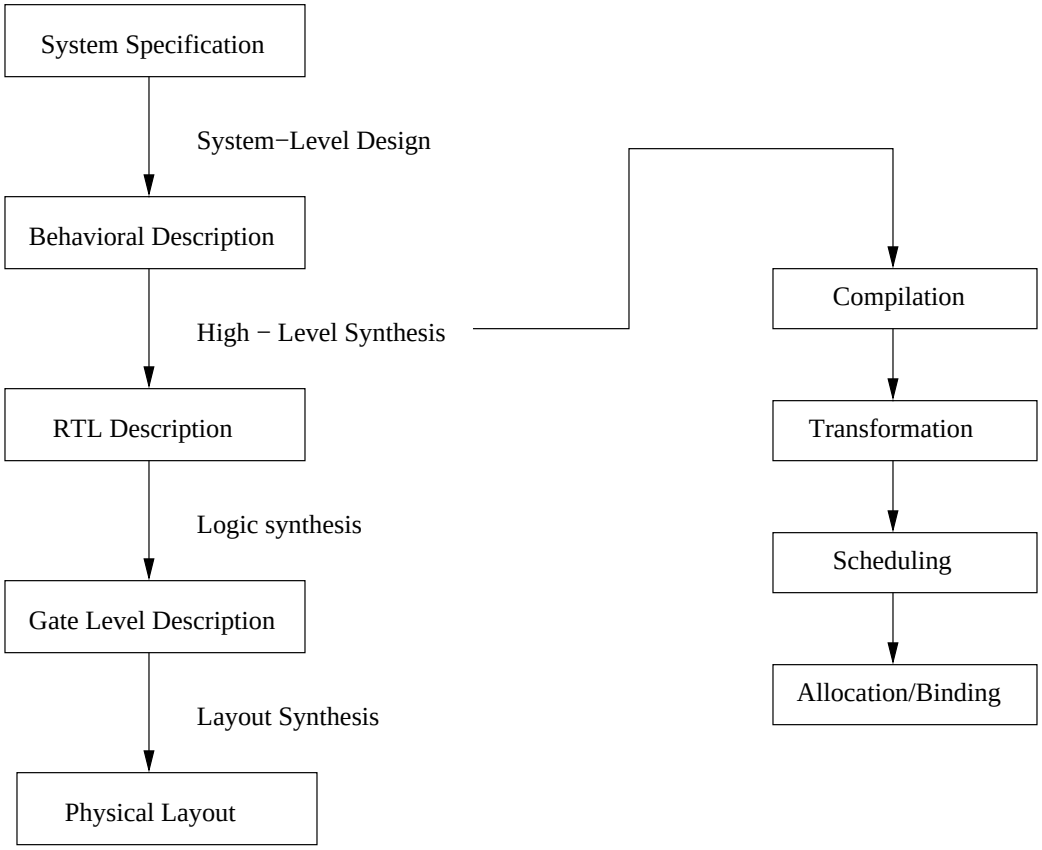


Figure 1.1.. VLSI Design Flow

High-level synthesis starts at the system level and proceeds downwards to RTL level, passing through each of the above phases, each time adding some additional information needed at the next level.

Behavioral synthesis requires transformation of the VHDL code into an internal representation which extracts registers, combinational logic equations and macros like '+', '-', etc. for the scheduling, allocation and binding processes. Most systems use a representation like the control flow graph and/or the data flow graph or the combination of the two like the CDFG as their intermediate format.

1.1 System Description and Intermediate Representation

The system to be designed is described at the most abstract level in plain English, i.e., in a form most easily understood by the user. The behavior of the system is captured at the algorithmic level

through a programming language such as Ada, Pascal , or a hardware description language such as VHDL, HardwareC [73] , MIMOLA [114] and SILAGE [40].

The behavior of the system, specified in a high-level language, is compiled into a internal representation that would be suitable for the rest of the synthesis process. The transformation of the behavioral specification into its unique graphical representation is analogous to the non-optimizing compilation of a programming language.

The data representation adopted by several behavioral synthesis systems may vary slightly in style and structure, but, in general, the control and data dependencies are encapsulated in one or two graphs. The data flow graph is a directed graph which depicts the flow of data, while the control flow graph is a directed graph which indicates the sequence of operations.

1.2 Scheduling

Scheduling is defined as that step in high-level synthesis in which the operations are grouped into control-steps based on their types and dependencies in such a way that the operators in the same control step could be executed simultaneously. A wide variety of approaches exist in efficient scheduling which are directed at either reducing the total time of execution or minimizing the number of resources needed for the design. Broadly, these approaches could be classified into four categories: Basic scheduling, time constrained scheduling, resource constrained scheduling and miscellaneous scheduling.

The control and data flow graphs depict the inherent parallelism in a design, based on which, each node could be assigned a range of control steps. Most of the scheduling algorithms require the earliest and the latest bounds that define the range of control steps for each node in the CDFG. Two simple schemes that are widely used to determine these bounds are called the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) algorithms.

The ASAP algorithm begins with scheduling the initial nodes, i.e. nodes without any predecessors, in the first time step, and assigns the time steps in increasing order as it proceeds downwards. The algorithm is guided by the simple principle that a particular node can be executed only if all of its predecessors have been executed. Ignoring resource constraints, this algorithm gives the least

number of control steps required for the design, and hence, could be used for near-optimal micro code compilation [84] .

The ALAP algorithm is analogous to the ASAP scheme, except that the operations here are intentionally postponed to the latest possible control step. The algorithm begins at the bottom of the CDFG, i.e., with nodes that have no successors, and proceeds upwards to nodes that have no predecessors. This algorithm gives the slowest possible schedule for a given design.

1.2.1 Time-Constrained Scheduling

The time-constrained scheduling approach is often adopted for designs targeted towards applications in real-time systems, like the digital signal processing systems, which are often limited by the response time. Here, the main objective would be to realize the design with minimum possible hardware while meeting the time constraint. Time constrained scheduling is usually implemented using three different techniques: - Mathematical programming - Constructive heuristics - Iterative Refinement.

Integer Linear Programming The ILP method is a mathematical formulation of the scheduling problem, which applies a branch-and-bound search algorithm with backtracking to find the optimal schedule.

$$P(x,z) = P(x) \text{ whenever } P(y,z) > 0 \tag{1.1}$$

The ILP approach begins with finding the earliest (E_k) and the latest (L_k) time-bounds for each operation using the ASAP and ALAP algorithms respectively. From these, the mobility range for each operation is calculated as

$$M = \{S_j \| E_k \leq j \leq L_k\}, \tag{1.2}$$

and the scheduling problem is formulated by the equation,

$$\text{Minimize } (\sum_{k=1}^{n}(C_k * N_k)) \text{ and } \sum_{E_i \leq L_i} x_{i,j} = 1, \forall 1 \leq i \leq n, \text{ no. of operations }, \tag{1.3}$$

where, $1 \leq k \leq m$ operation types are available, and N_k is the number of functional units of operation type k, and C_k is the cost of each FU. $x_{i,j}$ is equal to 1 if the operation i is assigned in control step j and 0, otherwise.

Such a formulation could be extended to further include resource and data dependency constraints using the equations,

$$\sum_{i=1}^{n} x_{i,j} \leq N_i \text{ and } ((q * x_{j,q}) - (p * x_{i,p})) \leq -1, p < q, \tag{1.4}$$

where p and q are the control steps assigned to the operations x_i and x_j respectively.

One major drawback of the ILP formulation is that its complexity increases rapidly with the number of control steps. For a single additional control step, n additional x variables have to be considered. The ILP approach is computationally intensive and hence, can be applied only to very small problems.

One other approach for time constrained scheduling is a heuristic method, called the Force directed scheduling. This algorithm tries to reduce the total number of functional units used by uniformly distributing the operations of the same type over the available control steps.

1.2.2 Resource Constrained Scheduling

Resource constrained scheduling algorithms are used in applications where the design is restricted by the silicon area. The goal of these algorithms is to minimize the number of control steps while satisfying the resource constraints. The schedule is built one operation at a time, so that the resource constraints and data dependencies are not violated. The total number of control steps are minimized in such a way that the number of operations scheduled in any control step does not exceed the number of FUs available.

Two popular approaches for scheduling operations with resource constraints include list-based scheduling and static-list scheduling.

List-based scheduling is based on including resource constraints in the ASAP algorithm. A priority list of ready nodes is maintained, and each such list is associated with a priority function that resolves any resource conflicts. A ready node is a node whose predecessors have already been scheduled.

The algorithm proceeds by first scheduling operations with higher priority while the lower priority operations are deferred to later control steps. At every step, the successors of a scheduled operation are added to the priority list of ready nodes.

The efficiency of such a list scheduling algorithm depends mostly on the priority function employed. A simple priority function could be chosen as to assign a priority that is inversely proportional to the mobility of the operation, and thereby, ensure that operations with large mobility are deferred to later control steps since they could go into more number of control steps.

Alternatively, we could assign a priority based on the length of the longest path from the operation node to a node with no immediate successor. One major drawback of the list-based scheduling is the increased time and space complexity because of the several lists that have to be maintained dynamically.

The static-list scheduling is based on building a single list of operations statically, as opposed to the normal list-based scheduling, where the list grows dynamically. The ASAP and ALAP algorithms are applied initially to find the mobility range for each operation. The operations are sorted in ascending order based on their greatest control step assignment, and then, the operations with the same greatest control step value are sorted in descending order of their least control step value. The operations are then scheduled sequentially in the descending order of their priority. The operations that cannot be scheduled in a control step due to unavailability of resources are deferred to later control steps.

1.2.3 Other Scheduling Approaches

Apart from the previously discussed scheduling algorithms, several other approaches, like the Simulated Annealing, have been successfully used to solve the scheduling problem.

In the Simulated Annealing based approach [26], scheduling is treated as a placement problem, where the operations are to be placed in a two-dimensional table of control steps versus available functional units. The algorithm begins with an initial placement of operations, and iteratively modifies the table by displacing an operation. The new schedule is evaluated based on the cost of displacement, and is accepted with a probability, even when it may not be better from the previous one, in order to overcome local minima in the solution space. The simulated annealing approach

is, thus, suitable for obtaining globally optimum solutions, but, requires long execution times for finding them.

Another approach is the Path-based scheduling [13], which is based on minimizing the number of control steps needed to execute the critical path in a CDFG. Initially, all possible paths of the CDFG are extracted and scheduled independently and later these schedules are combined to get the final schedule. The algorithm transforms the problem of introducing minimum control step constraint into a clique-partitioning problem. A clique partitioning solution would indicate the minimum overlapping of intervals in a given path.

1.3 Allocation and Binding

Allocation is the process of determining the functional units of each type for performing the operations while binding includes the process of assigning each such operation to a particular functional unit. Allocation ensures that sufficient number of resources are available for executing the operations and binding decides the actual components to be used for each operation. Binding has an impact on the amount of multiplexing and interconnections in the final design.

Allocation and binding could be classified into three categories based on their objective: allocation and binding for functional units, for memory units, and for interconnections. Allocation and binding for functional units consists of grouping operations in such a way that each group consists of mutually exclusive operations while the total number of groups is minimized. In memory unit allocation/binding, values that are generated in one control step and used in another are assigned to memory units for storage. Here, the objective is to minimize the number of memory units and also to simplify the communication paths. Interconnection allocation and binding includes assignment of buses, multiplexer and de-multiplexer connections to perform the data transfer in each time step.

1.4 Motivation for Our Thesis

The automation of design process has been deemed necessary by the increasing complexity of the designs and the decreasing marketing-time requirements of the design market. Shifting the design process to higher levels of abstraction has been the motivating factor for several research works in the High-level synthesis phase. Despite the availability of several tools for synthesizing

behavioral descriptions of designs, their application in research work is quite limited since most of them are commercially-oriented tools. Moreover, most of the previous works on high-level synthesis target data-dominated designs, but, are not adequate enough to handle control-dominated designs. Control-flow intensive behaviors with inherent loops and conditionals are quite possible in network-centered systems. This has motivated us to develop a comprehensive high-level synthesis system that could be used for both data-flow and control-flow intensive designs. The system, generating outputs at different stages of the synthesis process, aids researchers by providing them with the flexibility of several entry and exit points in the system.

The high-level synthesis process requires the compilation of the behavioral description of the design into a graphical representation, capturing the control and data dependencies. The derivation of such a Control and Data Flow Graph (CDFG) has been done mostly manually, which makes this process time-consuming and error-prone at least in the earlier stages of synthesis. Our synthesis system, therefore, includes a tool for automatic conversion of a given behavioral VHDL description into its corresponding CDFG. Such a CDFG is generated in several formats to accommodate different implementation approaches.

Traditionally, the design automation tools were developed with the objective of reducing area and improving the speed of designs. However, with the introduction of portable wireless devices and other micro equipment like laptop computers, power dissipation of the circuits has slowly evolved as a major concern of the design process. Such a trend has placed the problem of power optimization in the early design cycle. We have addressed this problem of power optimization in the binding phase of our synthesis system.

1.5 Thesis Outline

The rest of the thesis is organized as follows: We enumerate some of the previous works related to this field in chapter 2. An automatic conversion tool that is used to extract a CDFG from the given behavioral description is described in chapter 3. Chapter 4 gives a brief overview of the scheduling approach used in our synthesis system. Chapter 5 describes our game-theory based binding algorithm that incorporates power optimization. Experimental results obtained upon some

of the standard high-level synthesis benchmark circuits are presented in chapter 6. Finally, we give the concluding remarks in chapter 7.

CHAPTER 2

RELATED WORK

The advent of design automation has resulted in a significant amount of work at many levels of design abstraction. A number of techniques have been proposed for high-level synthesis, some of which are briefly discussed here. A taxonomy of related works in High-level synthesis(HLS) has been provided in Figure 2.1.. The related works are classified on the basis of the intermediate representation they use (DFG or CDFG), and the tasks that they target in HLS (scheduling, allocation or binding). We have also enumerated works on transformations of initial behavioral descriptions. We now present a summary of these works according to our classification.

2.1 Compiler-Level Transformations in High-Level Synthesis

In this section, we cite various works on compiler-level transformations of original behavioral descriptions that aid in the next steps of HLS.

Aho et al. [7] proposed the application of several compiler optimization techniques, such as constant folding and redundant operator elimination, on the flow-graph representation. Arrayed variables were another source of compiler-level optimizations for HLS considered in [36] and [79]. Since arrays in the behavioral descriptions get mapped to memories, it was proposed in [55] that reducing the number of array accesses decreases the overhead resulting from accessing memory structures.

Lis and Gajski [67] identified some of the advantages of capturing design requirements in a behavioral form. These include:

- Technology dependent details of implementation are not embedded in the design specification.

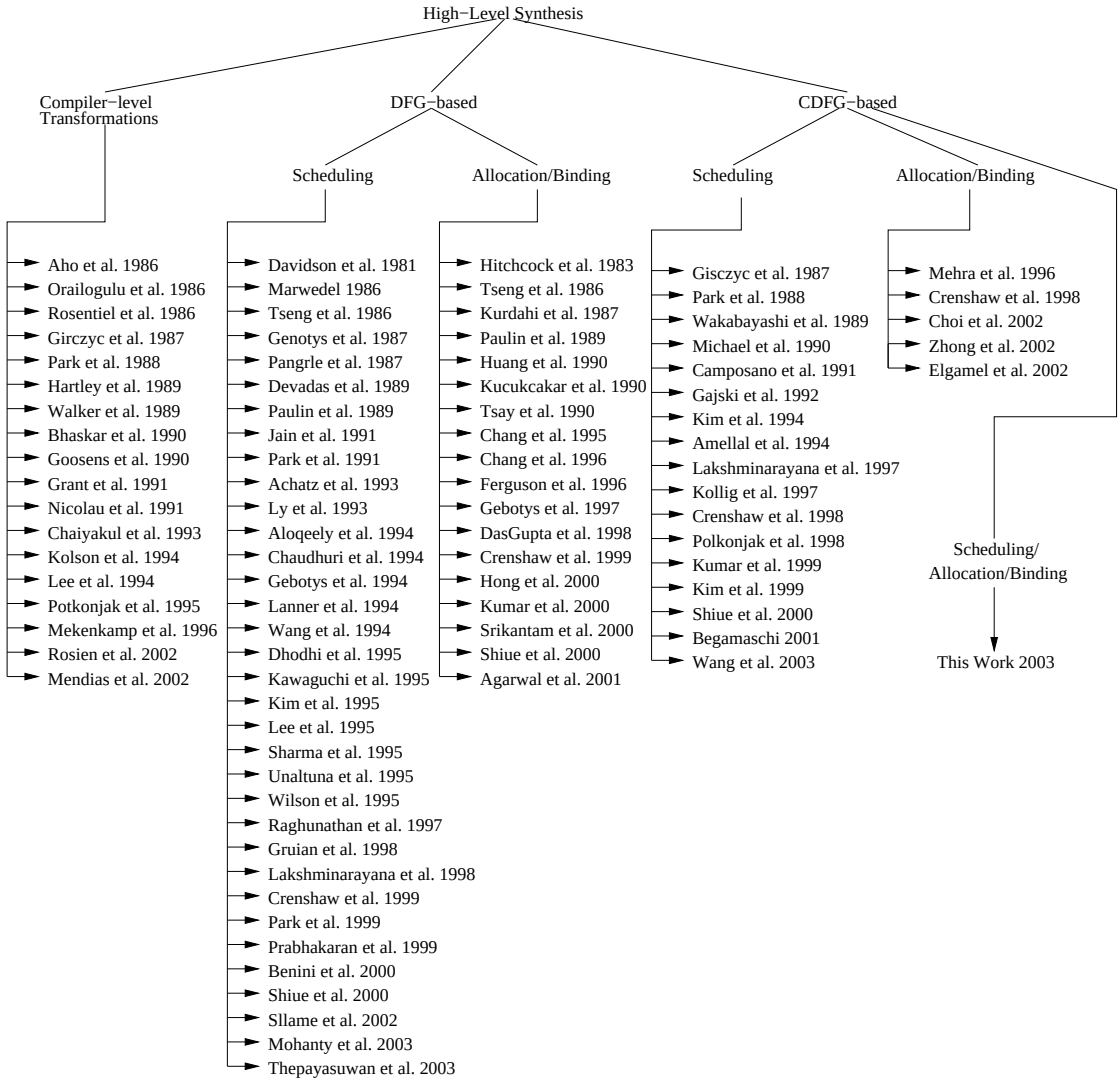


Figure 2.1.. Taxonomy of Related Works in High-Level Synthesis

- The behavioral description could be applied to a simulator to verify the correctness of a new design, or to validate an existing design specification.
- As the implementation technologies change, the available behavioral description could be used to redesign a circuit to make it compatible with the new technology.
- Behavioral synthesis increases productivity, minimizes errors, decreases design time without any technology specific expertise from the designer.

With the increasing use of VHDL for design description, some approaches have been proposed that are specific to transformations on VHDL. Bhasker and Lee [10] proposed approaches to identify specific syntactic constructs and replace them with attributes on signals and nets to indicate their functions. In order to reduce the syntactic variation of descriptions with the same semantics, Chaiyakul et al [14] proposed a transformation technique that uses assignment decision diagrams to minimize syntactic variance in the given description.

The COMET (ClusterOriented and Minimum Execution Time) design system proposed by Chang, Rose and Walker [20] synthesizes synchronous pipeline ASICs. It uses VHDL for describing the behavioral specifications. Such a description is restricted to statements with arithmetic and logical operations, control constructs like if, case and loops. They designed a subsystem, named VCOMP, for converting such a behavioral description into a DFG representation. The DFG is then subject to optimizing transformations.

Another approach for flow-graph transformations is the tree height reduction [39] that tries to improve the parallelism of the design. A similar method described in [77] uses the commutativity and distributivity properties of the language operators to decrease the height of a long expression chain, exposing the inherent parallelism in a complicated data-flow graph. Other commonly used transformations include pipelining [81], loop folding [33], software pipelining [35] and retiming [70]. Some pattern-matching transformations were applied by Rosenstiel in [90], which are based on RT semantics of the hardware components corresponding to flow-graph operators. Walker et al. [107] applied system level transformations to divide parts of the flow graph into separate processes that run concurrently or in a pipelined fashion. Mekenkamp et al. described a system, called TRADES (Transformational DEsign System) in [72], which uses a syntax based translation to transform a subset of VHDL constructs into a CDFG on a per statement basis. Due to such a

syntax based approach, the VHDL event mechanism appears in the CDFG without imposing any guidelines on the synthesis process.

Nijhar and Brown [78] identify significant differences between the optimizations of VHDL code and that of a conventional, sequential programming language which are often assumed to be on the same line. According to them, transformations applied on sequential programming languages are limited by a fixed target architecture, i.e., the architecture on which the program is to be run. VHDL optimization, however, has an extra degree of freedom associated with it in that it can manipulate the executing hardware itself.

Potkonjak et. al. [70] proposed methods for transforming a behavioral description so that synthesis of the new description requires less area overhead. They proposed a two-stage objective function for estimating the area and testability as well as for evaluating the effects of a transformation. From there, a randomized branch-and-bound steepest decent algorithm was employed to search for the best sequence of transformations.

In [91], Rosien et. al. present a method to automatically generate a CDFG from a C/C++ source code. Such a CDFG is used to automate the programming of a Field Programmable Function Array (FPFA), which is a flexible and energy efficient reconfigurable device. Their CDFG is represented using the hypergraph model, in which the operations are represented by edges (hyperedges) and the inputs and outputs are represented by the nodes which connect the edges. With such a representation, an operation can have any number of distinguishable inputs/outputs. Also, a hypergraph itself can be used as a definition of a new hyperedge and a whole hierarchical graph can be created this way. The authors have divided the process of generating such a CDFG from C/C++ code into several steps. First, a parse tree is generated from the code, from which the language constructs are converted into a list of hypergraph templates. A complete CDFG is built from these templates, which is then subjected to a series of behavior preserving transformations. Finally, a *clean* CDFG is obtained in which the control lines and the statespace are trimmed as much as possible.

2.2 DFG-Based Works

The works discussed in this section have used a Data Flow graph as their intermediate representation.

2.2.1 Scheduling

Paulin and Knight [85] introduced the force-directed scheduling (FDS) that uses a global selection criterion to choose the next operation for scheduling. Their FDS algorithm relied on the ASAP and ALAP scheduling algorithms to determine the range of control steps for every operation. The algorithm achieves its objective of reducing the number of Functional Units by uniformly distributing the operations of the same type into all the available control steps.

The HAL system, which is based on their force-directed scheduling approach, performs behavioral synthesis on a global scheme with step-wise refinement. Some of the constraints and features supported by the FDS algorithm include,

- multicycle and chained operations.
- mutually exclusive operations.
- scheduling with fixed global timing constraints, aimed at minimizing functional unit costs, register costs and global interconnect requirements.
- scheduling with local timing constraints.
- scheduling with fixed resource constraints.
- functional pipelining.
- structural pipelining.

The FDS scheme does not take into account future scheduling of operations into the same control step which leads to a lack of compromises between early and late decisions, which may result in a sub-optimal solution. Park et al. [81] overcome this weakness by iteratively rescheduling some of the operations in the given schedule. An initial solution is obtained using a standard algorithm, and that solution is maximally improved by rescheduling a sequence of operations till no improvement is attainable. The COMET system [20] applies the concept of Force-directed scheduling to interacting with cluster structure information. Their system is based on a tool called the Cluster Oriented Scheduling (COS), which uses pattern matching techniques to recognize the

cluster structures of a new algorithm as an instance of a dependency structure for mapping an algorithm to architecture.

In [38], Gupta et al. present a latency-constrained scheduling algorithm to optimize a design for dynamic power. Their work is motivated by the force directed scheduling algorithm proposed by Paulin and Knight [84]. Their algorithm reduces dynamic power by reducing switched capacitance inside resources, after evaluating the switched capacitance of combinations among DFG operations that could share resources. A force is associated with each feasible combination corresponding to the power consumption, and a distribution of such forces is obtained, whose mean, standard deviation and skew are used to produce a power-optimized schedule.

Rim and Jain [89] demonstrate a performance extension tool that computes a lower-bound completion time for non-pipelined resource-constrained scheduling problem for a given data-flow graph with a set of resources and for a specified resource delay and a clock cycle. Chaudhuri and Walker [18] produced an algorithm for computing lower bounds on the number of functional units of each type required to schedule a data-flow graph in a specified number of control steps. The bounds are found by relaxing either the precedence constraints or the integrity constraints on the scheduling problem.

A list-based scheduling algorithm that uses information from a DFG to guide its search for optimal / near-optimal schedules is presented in [100]. A DFG analysis is performed initially, which includes, finding the successors and predecessors of every node and the tree to which the node belongs to. With this available knowledge, the scheduler is supposed to make a perfect choice for the operation to be scheduled next.

The most basic constructive approaches for HLS, the ASAP and ALAP algorithms, have no priority assigned to operations, while the list scheduling approaches use a global criterion for selecting the next operation to be scheduled. Pangrle et al. [12] used the mobility of an operation as its global priority function, where mobility is defined as the difference between the ASAP and ALAP values of that operation. Another priority function, named urgency, was used by Girczyc et al. in [33], which is defined as the minimum number of control steps from the bottom at which an operation can be scheduled before a timing constraint is violated. The list of ready operations is ordered according to these priority functions and processed for each state.

Other scheduling approaches were proposed to address the problems of memory and storage. Kim and Liu [52] laid emphasis on minimizing the interconnection and then tried to group the variables to from memory modules. Lee and Hwang [65] proposed taking multiport memory into account as early as during scheduling. A multiport access variable (MAV) was defined for a control step, and the MAVs across all the control steps were equalized in order to achieve a better memory utilization. Aloqeely and Chen [5] proposed a sequencer-based architecture, where a sequencer is a stack or queue connecting one functional unit to another. High quality datapaths could be synthesized for many signal processing and matrix computation algorithms by letting the variables to either *stay* or *flow* through the sequencers for future use.

Achatz [1] proposed an extension to the ILP formulation so that it can handle multifunctional units as well as units with different execution times for different instances of the same operation type. Wang and Grainger [109] came up with a method to reduce the number of constraints in the original ILP formulation without reducing the explored design space, thereby, making the computation more efficient and more applicable to larger-sized problems. Chaudhuri et al.[19] described a well-designed ILP formulation for exploiting the structure of the assignment, timing, and resource constraints, and they further improved the well-structured formulation by adding new valid inequalities. Landwehr et al. proposed the OSCAR system in [63], which represents a 0/1 integer programming model for solving the three tasks of HLS. In [32], Gebotys proposed an integer programming model for the synthesis of multi-chip architecture which can simultaneously deal with partitioning, scheduling, and allocation. Wilson et al.[110] generalized the ILP approach in an integrated solution to the scheduling, allocation and binding in datapath synthesis.

Ly et al. [68] proposed a method for using behavioral templates for scheduling, where each template locks a number of operations into a relative schedule with respect to one another. It eases the handling of timing constraints, sequential operation modeling, prechaining of certain operations, and hierarchical scheduling. Unaltuna et al. presented a three-phase neural network based scheduling algorithm in [105], while, Kawaguchi and Tokada combined simulated annealing with neural networks in [50] for solving the scheduling problem.

Dhodhi et al. [28] proposed the application of a problem-space genetic algorithm for datapath synthesis, that performs concurrent scheduling and allocation with the objective of minimizing the resource cost and the total execution time. Ly et al. [69] adapted the simulated annealing procedure

to high-level synthesis that explores the design space by repeatedly ripping up parts of a design in a probabilistic manner and reconstructing them using application-specific heuristics. Sharma et al. [96] combined the allocation and scheduling of functional, storage and interconnect units into a single phase, using the concept of register state (free, busy or undecided) for optimizing registers in a incomplete schedule where the lifetimes of variables are yet to be available.

Langevin and Cerny [64] described a recursive method for estimating a lower bound on the performance of schedules under resource constraints for acyclic finite Data-Flow graphs. The recursive method is based on the greedy lower-bound estimator of Rim and Jain [89], which was formulated as resolving a relaxation of the general scheduling problem and allowed for chaining of operations, and pipelined and mylticycle operations.

Kollig and Al-Hashimi [54] described a new simulated annealing-based algorithm capable of solving scheduling, allocation and binding tasks simultaneously without the need of independent interconnect optimization. Their algorithm begins with an initial solution, and proceeds by generating new solutions which are either accepted or rejected based on an acceptance criterion defined in the algorithm. The probability of accepting solutions with increasing cost depends ona cost parameter, which is gradually lowered as the annealing process proceeds [25] . The moves used applied in the annealing procedure are chosen in a way to cover scheduling, allocation and binding tasks simultaneously. The four different kinds of moves that could applied on a randomly chosen operation in the algorithm at any time are:

- Randomly schedule the operation one control step earlier or later.
- Bind the value to a new register from the set of available registers.
- Bind the operation to a new functional module from a set of available modules.
- Swap the inputs of the operation if it is commutative.

Zhu and Gajski [113] established a theoretical framework for another concept of scheduling called soft scheduling. In soft scheduling, the decisions made are soft, i.e., they could be adjusted later. The authors discuss the applicability of soft scheduling to alleviate the phase coupling problem of HLS.

Shantnawi et. al.[4] presented a novel technique to obtain a rate-optimal and processor-optimal schedule for a fully-static data flow graph onto a multiprocessor system. The authors employ the Floyd-Warshall's shortest path algorithm to evaluate the relative firing times of the nodes of the DFG.

2.2.2 Allocation and Binding

Thepayasuwan et al. [102] proposed a novel technique for resource binding and operation scheduling to maximize the latency of the digital hardware such that its simultaneous switching noise is kept within feasible limits. The technique involves the automatic generation of performance models for each input specification and then applying an exploration algorithm to find the best resource binding and operation scheduling alternative.

Tseng and Siewiorek [104] divided the allocation problem into three tasks of storage, functional-unit, and interconnection allocation which are solved independently by mapping each task to the popular clique-partitioning problem of graphs. In the graph formulation, operations, values, or interconnection are represented by nodes. An edge between two nodes indicates those two nodes can share the same hardware. The allocation problem is thus transformed to the problem of finding the minimal number of cliques in the graph. Since the problem of finding the minimal number of cliques in a graph is a NP-hard problem, Tseng and Siewiorek adopted a heuristic approach to tackle it.

The clique-partitioning problem can minimize the storage requirements when applied to storage allocation. However, it totally ignores the interdependence between storage and interconnection allocation. Paulin and Knight [85] extend this approach by augmenting the graph edges with weights that reflect the impact on interconnection complexity due to register sharing among variables.

Hitchcock et al. [41] proposed a allocation system, named EMUCS, that starts with an empty datapath and builds it gradually by adding functional, storage and interconnection units as necessary. A similar approach was used in the MABAL system of Kucukacakar and Parker [59] which uses a global criterion based on functional, storage and interconnection costs to determine the next element to assign and where to assign it.

Kurdahi and Parker [60] used the left-edge algorithm to solve the register-allocation problem The left-edge algorithm has the advantage of a polynomial time complexity when compared to the clique-partitioning approach which is NP-complete. While the left-edge algorithm can successfully allocate the minimum number of registers, it fails to consider the impact of register allocation on the interconnection cost, which could be taken care by a weighted version of the clique-partitioning algorithm.

The register and functional-unit allocation problems have been transformed into weighted bipartite-matching algorithms in [43]. The authors use a polynomial time maximum weight matching algorithm that allocates a minimum number of registers and also takes, partially, into consideration, the impact of register allocation on interconnection allocation. Kumar and Bayoumi [3] considerd the binding of function units operating at multiple voltages. Their work is aimed at minimizing the power consumption due to switching activities on the physical components. They transformed the problem of binding into a graph-theory problem which was later solved using two approaches: greedy approach and an optimal approach. Shiue et. al. [98] presented a novel approach to low power binding in high-level synthesis based on linear programming methods. The binding problem was mapped on to a graph called the parallel graph(PG), upon which the linear-programming techniques were applied to search all paths to find the optimal binding that minimizes the overall power consumption due to switching activity. A datapath synthesized by constructive or decomposition methods, could be further improved by an iterative refinement approach, named reallocation. Tsay et al. [103] propose the application of a sophisticated branch-and-bound method by reallocating a group of different types of entities for datapath refinement.

In [2], the authors explored the potential of precision sensitive approach for the high-level synthesis of multi-precision DFGs. They focus on fixed latency implementation of the DFGs. They present register allocation, functional unit binding and scheduling algorithms to exploit the multi-precision nature of the DFGs for optimizing the area. An iterative improvement approach is developed with cost function being formulated in terms of number of bits of arithmetic operators and storage units.

Dasgupta and Karri [24] proposed algorithms for scheduling and binding to minimize data bus transitions. The algorithm was based on a simulated annealing process. Hong and Kim [42] proposed the repeated application of the computation of maximum flow of minimum cost in networks

for low power bus optimization during scheduling and binding. Chang and Pedram [16] proposed a new technique for reducing power consumption through register allocation and binding. The problem, in their algorithm, was formulated as a minimum cost clique covering problem, and solved for optimality using a max-cost flow algorithm. The same authors proposed an approach of power reduction in [17] for binding of functional units. The problem, in this case, was formulated as a max-cost multi-commodity flow problem and solved for optimality. Since the multi-commodity problem is NP-hard, the functional unit binding problem domain was restricted to functionally pipelined designs with shorter latency. The approaches provided in [16] and [17] have the drawback of their application being limited to a number of specific small sized low-power problems.

2.3 CDFG-Based Works

2.3.1 Scheduling

The works described in preceding sections have considered only blocks of straight-line code. However, in addition to blocks of straight-line code, a realistic design description usually contains both conditional and loop constructs.

The problem of scheduling for control-dominated applications is considered in [111] and [58]. Several scheduling techniques based on a control flow graph (CFG) model are presented in [13] [11] [9] [31]. The CFG is basically a graphical description of a sequential program based implementation of the functionality. Even though such a CFG model could be comfortably used to capture the execution of instructions on a general-purpose uniprocessor, its application in exploiting the parallelism inherent in typical control flow intensive applications is limited.

Kim et al. [53] have proposed some techniques to schedule conditional constructs. In [106], a conditional vector is used to identify mutually exclusive operations so that an operation can be scheduled in different control steps for different execution instances. Camposano et al. [13] proposed a path-based approach, called the As Fast As Possible (AFAP) scheduling, which first extracts all possible execution paths from a given behavior and schedules them independently. The schedules for the different paths are then combined by resolving conflicts among the execution paths. Similarly, different approaches have been proposed for handling loop constructs, like the pipelining method described in [82] and loop folding [33].

Some graph representations that combine control and data flow into a single graph are presented in [47][29][56]. In [47], the control dependencies are expressed, but only the data dependencies are taken into account, while the control flow is not exploited. Here, the loops are represented using a branch node at the beginning of the loop and a merge node at the end. Such a branch-merge loop construct is developed for each variable in the loop, thus, resulting in a complex sub graph with a high degree of redundancy of branch and merge nodes. A similar representation is used in [29]. The Sprite Input Language described in [56] uses a single signal flow graph and is more confined to DSP applications.

Amellal and Kaminska [6] presented a control and data flow graph (CDFG) model for system representation which includes a new representation of conditional branches. They had developed a mutual exclusion testing procedure that provides for optimized resource sharing and critical path reduction. Some of the salient features of their CDFG representation include,

- With the representation of the data and control flows in the same graph, both the datapath and the controller can be synthesized from that same graph.
- The CDFG is an optimized representation of the control and data flows without any redundant dependency representations.
- Their CDFG representation of the behavioral descriptions does not impose any restrictions on the scheduling tasks, thereby, resulting in a better exploration of the design space.
- A branch numbering scheme is developed to solve the problem of resource sharing among mutually exclusive operations of the CDFG.

Apart from the single-graph model, the authors have formulated a new mathematical approach for the scheduling problem based on penalty weights. They had used the Tabu Search technique, which has been effective in finding optimal solutions for many types of large and difficult combinatorial optimization problems. The authors claim that the fast and intelligent solution space exploration provided by the Tabu technique makes their scheduling algorithm quite powerful.

The Wavesched scheduling algorithm presented in [61] uses a CDFG model that preserves parallelism inherent in the application. This algorithm is aimed at minimizing the average execution time of control-flow intensive behavioral descriptions. With its ability to overlap the schedules

of independent iterative constructs, the bodies of which share resources, the Wavesched algorithm could explore previously unexplored regions of the solution space. A general loop handling technique was developed to incorporate other optimization techniques like loop unrolling. Also, the algorithm can support multi-cycled and pipelined functional units and can use chaining to enhance the cycle time utilization.

Potkonjak and Srivastava [86] introduced a transformation, named rephasing, to manipulate the timing parameters in a CDFG during the high-level synthesis of data-path intensive applications. They use the rephasing approach to manipulate the values of the phases (or the relative times when corresponding samples are available at input and delay nodes) as an algorithmic transformation before the scheduling/allocation stage. They have shown that phase values can be chosen to transform and optimize the algorithms for factors like area, throughput, latency and power. In effect, the authors presented a technique for behavioral optimization through the manipulation of timing constraints.

Sentieys et al. [94] presented an architectural synthesis tool dedicated to DSP applications, in which, synthesis is achieved under time as well as silicon cost constraints. The algorithm is described in VHDL behavioral language, from which a CDFG is obtained and synthesized into processing, control, memory and communication units. The specifications of the designs in VHDL allowed for the interconnection with CAD and simulation tools. Kim et al. present a verification method for VHDL behavioral level design in [51]. To identify coding errors that a compiler cannot detect, the VHDL code is converted into a CDFG and verification patterns are applied on the CDFG. They have also proposed other algorithms like backward training and forward training algorithm to actuate coding error and propagate it.

In an attempt to bridge the gaps between high-level and logic synthesis, Bergamaschi [8] presented a novel internal model, called the Behavioral Network Graph (BNG), that represents both data and control constructs, for synthesis that covers the domains of both high-level and logic synthesis. This model is an RT-level network capable of representing all possible schedules that a given behavior may assume. It allows high-level synthesis algorithms to be formulated as logic transformations and effectively overlapped with logic synthesis. The author has also addressed the problem of a lack of formal representation to be used by different algorithms and systems, which makes the sharing of benchmark examples difficult.

Kollig et al. [54] described a new simulated annealing-based algorithm capable of solving all HLS tasks concurrently without the need of independent interconnect optimization. The HLS problem is formulated with a schedule time a module binding for each operation, a register binding for generated values and a Boolean variable indicating whether the inputs of commutative operations are to be swapped. This formulation is subject to constraints including data dependencies, execution time and module availability. Starting with an initial solution, new solutions are generated and either accepted or rejected depending on the acceptance criterion defined in the simulated annealing algorithm. The probability of accepting solutions with increasing cost depends on a control parameter, which is gradually decreased while the annealing process proceeds.

In GEM (Geometric Algorithm for Scheduling), proposed by Raje and Sarrafzadeh [88], a critical path based approach is used for scheduling operations. The algorithm uses weighted geometric point dominance matching from the operations onto the control steps. It starts with a CDFG and converts it into directed acyclic graphs by breaking the loops and removing the feedback edges while maintaining the conditions for the loops. Such an algorithm is performed in O(nplogp) steps, where n is the number of disjoint paths in the CDFG and p the number of nodes in the longest path. The problem of scheduling a path is transformed into the problem of obtaining a matching from a set of points OPi's, which represent the various operations in a path, onto a set of points Ci's, which represent the control steps. Certain constraints are imposed onto the matching owing to the data dependencies of the operations to be matched. The dependencies are usually described as precedence relations.

Wang et al. [108] described a comprehensive high-level synthesis system for control-flow intensive as well as data-dominated behaviors. Their algorithm is based on an iterative improvement strategy and performs clock selection, scheduling, module selection, resource allocation and assignment simultaneously, and also consider the interactions between these tasks to benefit completely from the design space exploration at behavior level. Their scheduling algorithm supports concurrent loop optimization and multicycling under resource constraints. The authors use a variation of the general CDFG model for their synthesis system, where their CDFG model includes some additional nodes to represent the start of a loop, etc. However, their system assumes that the initial CDFG representation of the application is available.

2.3.2 Allocation/Binding

In [30], Elgamel et al. present a novel approach for utilizing genetic algorithms to solve High-level synthesis tasks with multiple voltages. They have incorporated a new way of modeling and encoding the resulting chromosomes. Their system, takes as its inputs, a CDFG, hardware library, and the time, area constraints. With this information, the algorithm solves the tasks of scheduling, allocation and binding simultaneously in order to generate a solution optimized for average and peak power. The evaluation function is formulated so as to consider the average and peak power consumptions while satisfying the given constraints.

Choi and Kim [21] proposed an efficient binding algorithm for power optimization in high-level synthesis. The authors claim that the traditional approach of formulating binding problem as a multi-commodity flow problem is limited to a class of small sized problems owing to the NP-hard nature of the multi-commodity flow problem. They have developed a new technique that uses the property of efficient flow computations in a network so that it is extensively applicable to practical designs while producing near-optimal results. They propose a heuristic algorithm, named BIND-lp, that finds a feasible binding by utilizing the flow computation steps and later refining them incrementally.

The application of increased parallelism to allow voltage reduction for the same computational throughput is depicted in [15]. This work led to several other works like [74], which uses slack to avoid unnecessary computations, and [5] which shows how a DFG might be partitioned for multiple voltages. In [7], the voltage idea was combined with an iterative improvement approach using a square switching matrix as the basis for a signal correlation matrix.

Crenshaw et al. proposed several heuristics in [22] to investigate the problem of exploiting signal correlation between operations to find a schedule and binding which minimizes switching. They describe an algorithm for scheduling communications on a bus, which reduces bus switching upto 60% without any increase in the number of cycles required for the schedule. Their technique of capturing signal correlation information during behavioral simulation can be applied in addition to popular voltage reduction methods. The switching information, thus obtained through simulations, is stored in the from of a switching table. A cubic table represents the switching activity for conditional nodes by including data for switching from a node n_i to a node n_{i+j}, where j $>$ 1. In

the absence of conditional nodes, all the data represents switching between node n_i and n_{i+1}, for all i. Thus, their method could be applied to graphs with both conditional and data nodes.

Mehra et al. proposed the partitioning of CDFG into groups with minimized inter-group communication in order to reduce the switched capacitance.

Zhong et al. [112] presented a general sufficient condition for register binding to ensure that a given set of functional units is perfectly power managed, i.e., does not contain any spurious switching activity. Their method is applicable to both data-flow intensive and control-flow intensive behaviors and leads to a straightforward power-managed register binding algorithm. The authors claim that their algorithm, begin independent of the functional unit binding and scheduling steps, could be easily incorporated into existing high-level synthesis systems. In [27], a technique is proposed to redesign the control logic to configure existing multiplexer networks to minimize spurious switching activity. A register binding algorithm which guarantees an RTL circuit for control and data-flow intensive behaviors, which is free of spurious switching activity, is presented in [62].

2.3.3 Our Work

The work presented here describes a High-Level Synthesis tool that could be used to solve each of the aforementioned tasks. This tool is targeted at both control-flow intensive and data-flow intensive behaviors, and incorporates several additional features like optimization for resources and power consumption. Our system uses a single graph representation of control and data flow dependencies as its intermediate form, and also incorporates a tool for transforming the original VHDL description into the corresponding CDFG.

CHAPTER 3

CDFG EXTRACTION FROM VHDL

The first, and often ignored, step in the high-level synthesis (HLS) process is the conversion of the original behavioral description in VHDL into an intermediate representation that captures the details of the description in a form suitable for the next steps of HLS. Some of the most commonly used intermediate representations include, the Data Flow Graph (DFG), the Control Flow Graph (CFG), and the Control and Data Flow Graph (CDFG). While DFG is the most prominent form of representation, the CDFG has the advantage of depicting both control and data constructs in a single graph, giving a better state-space exploration capability for the later steps. In this chapter, we describe a conversion tool that extracts such a CDFG from a given behavioral VHDL code. This tool is based on compiler-like transformations and other behavior-preserving transformations.

3.1 Introduction

Today, Very High Speed Integrated Circuit(VHSIC) Hardware Description Language (VHDL) has emerged as the default hardware description language. VHDL is intended to describe a hardware. This description can be fed to a simulator, which simulates the behavior of hardware modeled in the VHDL description. If the description is correct, the simulation exhibits the same behavior as the hardware.

Initially, VHDL was intended for use as an input to the VHDL simulator and not for synthesis. The simulation of the code was done to accurately predict the voltage values on all nets at any time and verify timing relationships between changes on these nets. On the contrary, the goal of the synthesis is to implement the given behavior by interconnecting components from a given library. Hence, simulation deals with timing while synthesis deals with connectivity . For a simulation to generate the correct models for the behavior of nets of interest, the description driving those nets

need not be minimal as long as it produces the correct behavior. Therefore, different descriptions for the same functionality are likely to synthesize into designs of varying quality.

VHDL models the hardware as concurrently running processes. Each process contains an algorithmic description of the process' behavior. Thus, modeling hardware in a single process results in a purely behavioral description at the highest level of abstraction. However, as soon as a VHDL description contains more than one process, it defines some structure within the hardware. This structure implicitly adds to the model's behavior. It also lowers the level of abstraction, which can come close to the hardware if each process only has the functionality of a gate. VHDL modeling is analogous to the concept of a container. The values within these containers change with time as they are affected by varied input stimuli. Such a model is suitable for both synchronous and asynchronous behavior.

The process of synthesizing an RTL structure from the functional description during the high-level synthesis involves three phases: the timing of computational operations (*scheduling*) determining the number of instances of each resource needed (*allocation*), and the assignment of resources to computational operations (*binding*). Behavioral synthesis requires transformation of the VHDL code into a internal representation which extracts registers, combinational logic equations and macros like '+', '*', etc., for the scheduling, allocation and binding processes. Most systems use something like the control flow graph and/or the data flow graph or the combination of the two like CDFG as their intermediate format.

A CDFG is expected to capture all the control and data flow information of the original VHDL description while preserving the various dependencies [93]. This CDFG undergoes incremental refinement as it passes through various stages of a high-level synthesis system to finally yield a register transfer level representation of the behavioral specification. Scheduling partitions the set of arithmetic and logical operations in the CDFG into groups of operations so that the operations in the same group can be executed concurrently, while trying to minimize the total execution time and/or the hardware cost.

Researchers working on the high-level synthesis problem begin with the assumption that a flow-graph representation of the behavioral description is available. Here, an often over-looked significant step is the conversion of the VHDL code into a CDFG. This step, normally carried out manually prior to the high-level synthesis, may take few minutes to few days based on the size and

the complexity of the VHDL code. Also, the chances for errors in the resulting CDFG increase with the increasing complexity of the code, and being one of the initial steps, this would drastically affect the accuracy of the whole design-flow. To counter this problem, we have developed a new conversion tool that automates the process of obtaining CDFG from a VHDL code, thereby reducing the design time significantly and providing a useful aid to the developer. The task of such a VHDL to CDFG compiler would be to extract a fully behavioral description which would be fed to an architectural synthesis system that will add structural information to that description.

3.2 Preliminaries

In this section, we provide the basic definitions and concepts which from the basis for our conversion tool.

3.2.1 Control and Data Flow Graph

The Control and Data Flow Graph is a directed acyclic graph in which a node can be either an operation node or a control node (representing a branch, loop, etc.) [92]. The directed edges in a CDFG represent the transfer of a value or control from one node to another. An edge can be conditional representing a condition while implementing the if/case statements or loop constructs. Figure 3.1. shows a CDFG representation for the following VHDL code fragment.

```
A := B * C + D;
while( A > 0 ) loop
                    A := A - 1;
end loop;
```

Nodes

In general, the nodes in a CDFG can be classified as one of the following types [92].

- Operational nodes: These are responsible for arithmetic, logical or relational operations.
- Call nodes: These nodes denote calls to subprogram modules.

- Control nodes: These nodes are responsible for operations like conditionals and loop constructs.
- Storage nodes: These nodes represent assignment operations associated with variables and signals.

In the CDFG representation shown in figure 3.1. , nodes numbered 1, 2, and 6 are operational nodes, while node number 3 is a storage node, and nodes 4 and 5 are control nodes.

Dependencies

Given any two operational nodes in the CDFG, represented as N_i and N_j, where N_i is a predecessor or ancestor of N_j in the CDFG, the possible dependencies between them are defined as follows:

- N_j is *flow-dependent* on N_i if the output of N_i is one of the inputs of N_j.
- N_j is *anti-dependent* on N_i if the output of N_j is one of the inputs of N_i.
- N_j is *output-dependent* on N_i if the output of N_j is also the output of N_i.

A data dependency is said to exist between two nodes in the CDFG if any one of the above three types of dependencies holds between them.

3.2.2 Basic VHDL Constructs

VHDL is a language for describing digital electronic systems and is designed to fill a number of needs in the design process[45]. It allows description of the structure of a system, how it is decomposed into subsystems and how those subsystems are interconnected (*structural description*). It also allows the specification of the function of the design using familiar programming language forms (*behavioral description*).

The *entity*, in VHDL, is the most basic abstraction that represents the design model. It is a structural model of some piece of hardware. An entity consists of processes and instantiations of other entities, and when fully expanded, only the processes remain. These processes are connected through signals, and the processes and signals together define the structure of the hardware model.

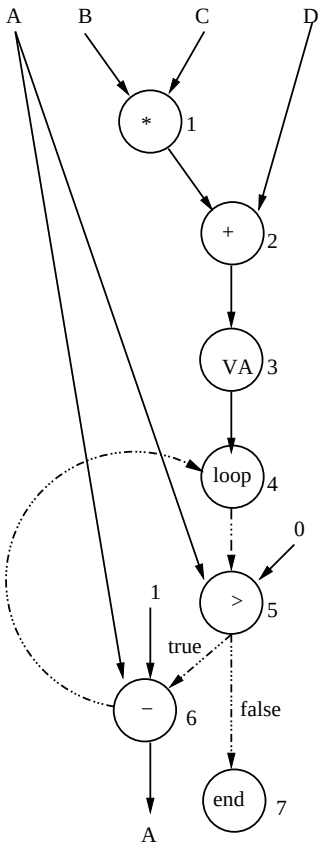


Figure 3.1.. CDFG Representation

A description of the internal implementation of the entity is given in the *architecture body* of that entity. The behavioral architecture body of entity describes its function in an abstract way and the concurrent statements in it are limited to process statements, subprogram calls and signal assignments.

A process is a sequence of statements which can affect the values of the signals interconnecting the processes. Each such process implements a part of the behavior of the entity in which it is contained. All processes of an entity constitute the behavioral model of that entity.

The process statements are further made up of sequential statements that are much like the kinds of statements we see in a conventional programming language such as statements evaluating expressions, statements assigning values to variables (*variable-assignment statements*), conditional execution statements (*if-then-else, case, etc.*), repeated execution statements (*loops*) and subprogram calls. In addition, there is the signal assignment statement, which is unique to hardware modeling languages. This statement is similar to variable assignment statement, except that it causes the value on a signal to be updated at some future time.

VHDL is defined as an event-driven model. An event is said to have occurred on a signal if updating that signal causes its current value to change [44]. Each event is the result of an assignment to a signal. Every signal assignment is bound to a specific moment in simulated time. Due to the event driven model and the timing information in signal assignments, the concept of absolute time is buried deeply into the semantics of VHDL [44] .

The tool described in this chapter handles the behavioral description and the various constructs involved in it. The process body of the behavioral architecture embodies the data and control flow of the design specification for synthesis purposes [92]. The concurrent signal assignment statements outside the process body can be viewed as equivalent process statements with the signal assignments being made within the process body.

3.3 Implementation Details

In this section, we will see the various details involved in the implementation of the tool. Section 4.1 describes the methodology followed in the extraction of the CDFG from VHDL code. Section 4.2 gives the transformations of various VHDL constructs and section 4.3 presents the

various outputs generated by the tool. Section 4.4 describes a couple of VHDL constructs that are not handled by the tool.

The flow-graph representation of the behavioral description is obtained by parsing the input VHDL code. Figure 3.2. depicts the various steps involved in this methodology. These steps are quite similar to the phases of a compiler, each of which transforms the source program from one representation to another.

The *lexical analysis* phase translates the source program into a stream of tokens, where each token is a sequence of characters with collective meaning, such as an identifier, a keyword, an operator or a punctuation character [7]. This stream of tokens is further subjected to *syntactic analysis* which imposes a hierarchical structure on them to verify the syntax of the program. The codes for these two phases were generated by applying the standard compiler construction tools, Lex and YACC [45], upon the IEEE standard VHDL syntax [44].

3.3.1 Methodology

While the parsing of the tokens using YACC code imposes a hierarchical structure that could be visualized as a tree-structure (*parse tree*), it does not actually generate such a parse tree. The parse tree, that is assumed to be available, is just a conceptual visualization of the syntactic structure of the program. Explicit codes are required to extract such a parse tree from the YACC code [92].Specific C++ codes were used for this purpose in our tool. The parse tree, thus obtained, is further compressed to obtain a *syntax tree* in which the operators appear as the interior nodes, and the operands of an operator are the children of the node for that operator.

The syntax tree is transformed, through another C++ code, into the final control and data flow graph that depicts the total flow of the control and data in the original description. The various data dependencies that are inherent in the flow graph can be revealed through one full scan of the graph. Figure 3.3. gives a vivid description of the above methodology through an example.

3.3.2 Algorithmic Description

In this section, we provide an overview of the algorithm, followed by the details. Figure 3.4. presents the pseudo-code for the conversion. First, we generate codes for lexical-analysis and the

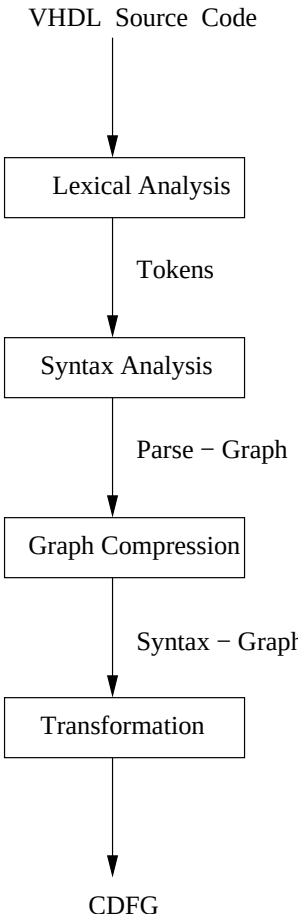


Figure 3.2.. Steps Involved in Extraction of the CDFG From VHDL Code

syntax-analysis using the standard Lex and YACC tools. The YACC program includes explicit functions to create nodes at each step in the syntactic-hierarchy. Each such node is a collection of information on its function or purpose at that stage in the hierarchy, and links to its predecessors and successors in that hierarchy. For example, an *architecture* statement would create an *architecture* node that stores the details of the corresponding *entity* node (a predecessor) and *process* nodes (successors) and nodes corresponding to statements within those processes (successors), etc. In short, each node in the hierarchy stores all the information relevant to that node.

Once we have extracted all the details regarding various operations and their order of execution, the nodes corresponding to the constructs like ”architecture” and ”process”, which were used to obtain the links between external inputs and the operations on them, can now be safely discarded. Our next step in the algorithm truncates the parse tree by removing all such nodes while preserving

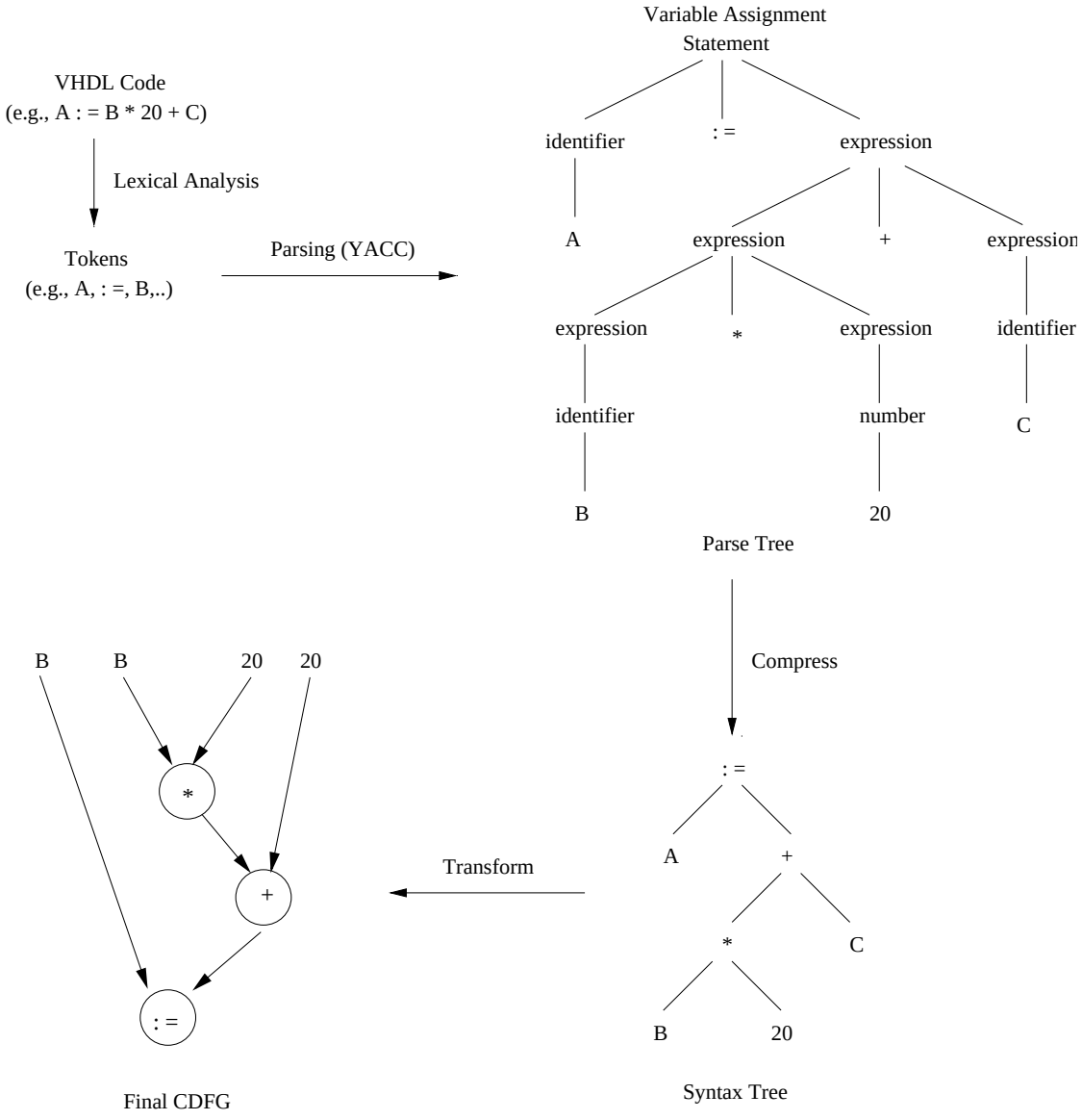


Figure 3.3.. Extraction of CDFG From a Sample VHDL Code

```
(01) VHDL_lex ← Generate Lex code
(02) VHDL_parse ← Generate YACC code
(03) token_set ← VHDL_lex(VHDL code)
(04) parse_tree ← VHDL_parse(token_set);
(05) CDFG ← truncate(parse_tree);
(06) Generate_adjacency_lists
(07) begin
(08)      Create an empty list for each node in the CDFG
(09)       for each node N_i in the CDFG
(10)           for each input k_j in the input-set of N_i
(11)               if k_j is a variable, constant or a signal,
.                      add its node to the adjacency_list of N_i
(12)               if k_j is another node,
.                      add N_i to the adjacency_list of k_j
(13)           end for
(14)       end for
(15) end
(16) Generate_set_representation
(17) begin
(18)       V ← {}; E ← {};
(19)       for each node N_i in the graph
(20)           V ← V ∪ N_i;
(21)           for each node N_j in the adjacency_list of N_i
(22)               E ← E ∪ (N_i,N_j);
(23)           end for;
(24)       end for;
(25) end
```

Figure 3.4.. Algorithm for CDFG Extraction

their dependency information. Nodes representing several punctuation marks and other syntactic details of the language, which are of least significance to the hardware design, are also scrapped in this step.

A succinct representation of the CDFG is thus obtained and further transformed into the adjacency - list representation by creating lists for each node in the CDFG and adding nodes corresponding to the input-set to that list. A depth-first search procedure is then employed on the adjacency-lists to obtain the set-representation, from which the output file for the VCG tool [37] is generated. Next, we use the adjacency-lists and the original node-representation to obtain the dependency-relations. While obtaining such relations for each node, we traverse the CDFG up-

wards from that node until we reach a control node, so that we cover the dependencies within each control block [92].

3.3.3 Transforming VHDL Constructs

In this section, we describe the transformations of each of the VHDL constructs that are handled by this tool. Each node in the transformation can be viewed as a 3-tuple given as (id, type, input_set). The *id* denotes the unique identifier (usually, a number) of that node. The *type* denotes the type of the node - operational(*,+,-,AND,OR,etc.), conditional (if-then-else, case). The input_set gives the set of all elements input to the node, which could be either variables/signals or expressions denoted by other node-ids. The id of a node is used to denote the output of that node. The input_set of each node automatically enforces a data or control dependency and thus reduces the overhead of maintaining output and dependency sets for each node. This simplifies, to a large extent, the final graph representation while preserving all the dependencies that exist in the original VHDL description.

We now describe the various VHDL statements and their corresponding transformations below:

3.3.3.1 Operational Statements

These statements involve arithmetic operators(+, -, *, /) or logical operators(AND, OR, NOT, NAND, NOR, XOR, XNOR). The node type indicates the type of operator involved and the input_list includes the two operands involved in the operation. In case of an unary minus (-) , the input_list involves only one operand. A typical AND statement and its corresponding representation are shown below. The corresponding graphical representation is shown in figure 3.5..

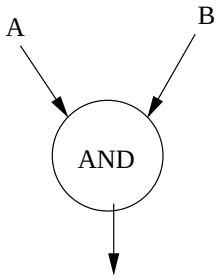


Figure 3.5.. CDFG: AND Operation

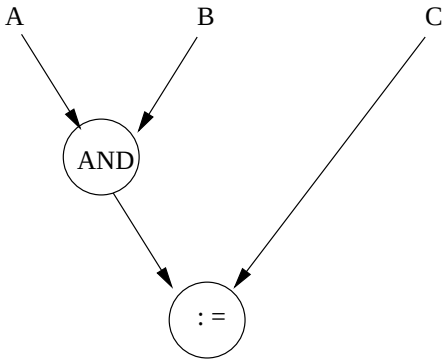


Figure 3.6.. CDFG: Variable Assignment

VHDL Statement:

A and B

Textual CDFG:

Node_Id: 1

Type : And

Input1 : A

Input2 : B

3.3.3.2 Assignment Statements

A VHDL assignment statement has a storage (variable or a signal) on the left side and an expression (consisting of primitive operations) on the right hand side.

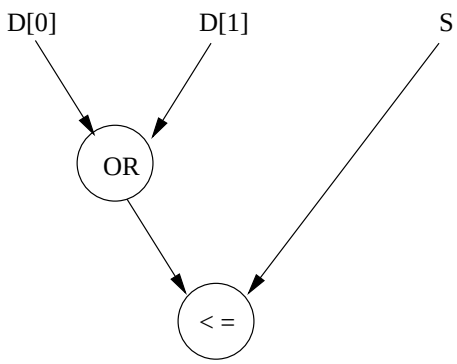


Figure 3.7.. CDFG: Signal Assignment

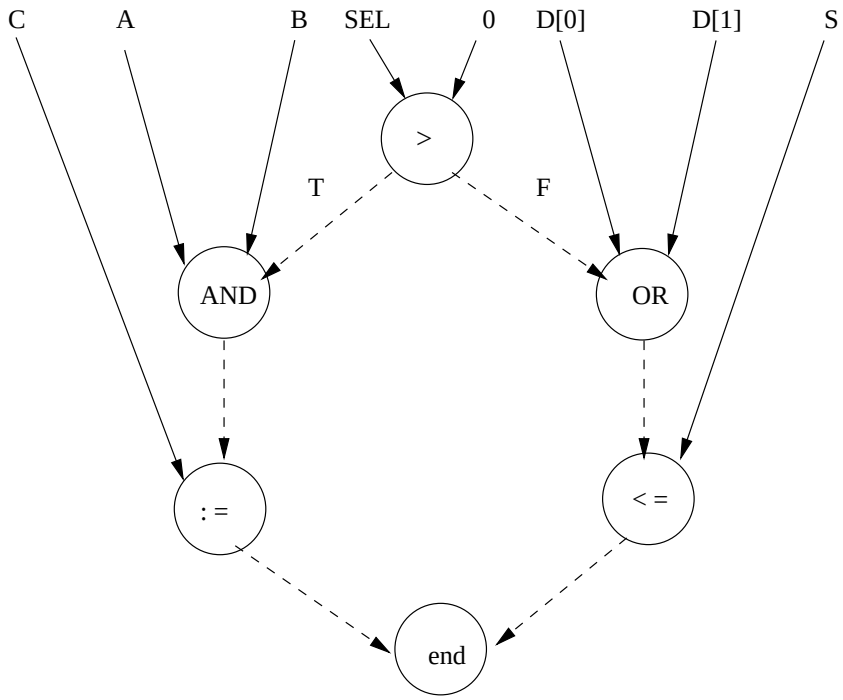


Figure 3.8.. CDFG: If-Then-Else Statement

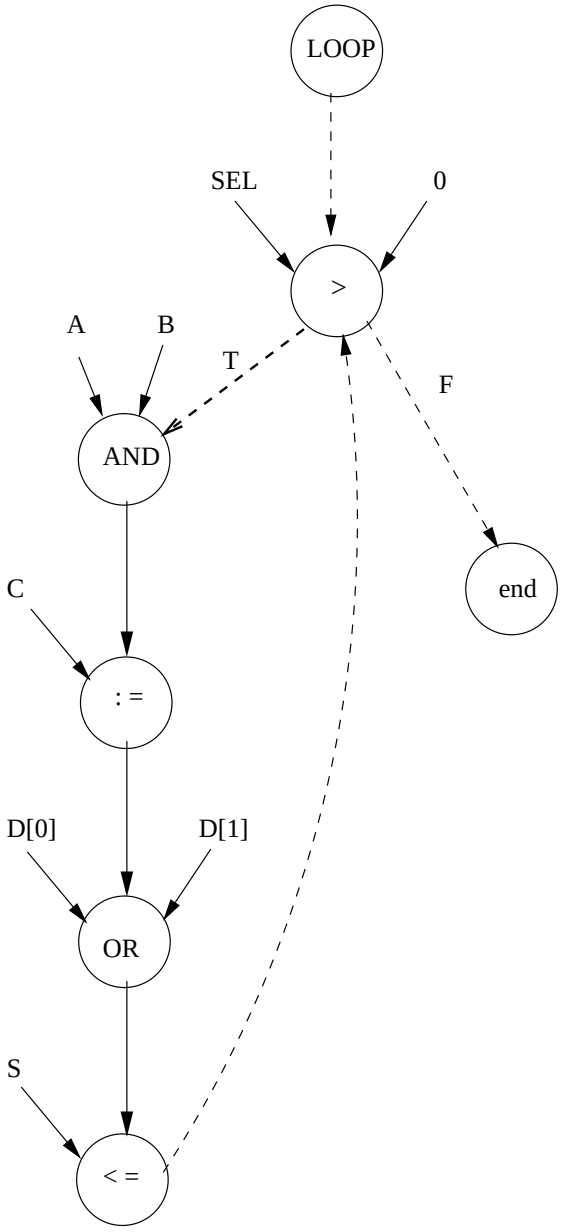


Figure 3.9.. CDFG: Loop Statement

Variable Assignment: A variable assignment statement assigns a new value to a variable. The syntax of such a statement is given below: variable_assignment_statement ::= target ':=' expression';'

The target denotes the variable which is assigned the value of the expression.

The variable assignment operator (:=) is represented as a "var_assign" node. The input_set of such a node contains the final data flow edge of the corresponding expression tree. A typical variable assignment statement and its textual representation are given below. The corresponding graphical representation is given in figure 3.6..

VHDL Statement:

C := A and B;

Textual CDFG:

Node_Id: 2

Type : Var_Assign

Input1 : C

Input2 : Node 1

Signal Assignment: A signal assignment statement is represented similar to that of variable assignment statement except that the type of the node is "signal_assign" as shown below. This statement is represented in the CDFG as shown in figure 3.7..

VHDL Statement:

S <= D[0] or D[1];

Textual CDFG:

Node_Id: 3

Type : Or

Input1 : D[0]

Input2 : D[1]

Node_Id: 4

Type : Signal_Assign

Input1 : S

Input2 : Node 3

3.3.3.3 Conditional Statements

if-then-else statement The representation for such a statement is required to capture information about the condition, the statements that are executed when the condition is true and the statements that are executed when the condition is false. Therefore, the node for an if-then-else statement is accompanied by a node describing the condition for that statement. Such a representation is shown below. The graphical representation for such a node is given in figure 3.8..

VHDL Statement:

if(SEL $>$ 0) then

C := A and B;

else S $<=$ D[0] or D[1];

Textual CDFG:

Node_Id: 5

Type : Comparator

Input1 : SEL

Input2 : 0

Node_Id: 6

Type : If

Condition: Node 5

Statements When True : Node 2

Statements When False: Node 4

Case Statement This is similar to the if-then-else representation, except that its node is accompanied by a node for the expression in case statement and one node each for the 'when' clauses of the case statement. An example is given here. The corresponding graphical representation is similar to that of the if-then-else statement.

VHDL Statement:

```
Case SEL
when 0 =>
C := A and B;
when others =>
S <= D[0] or D[1];
end case;
```

Textual CDFG:

```
Node_Id: 7
Type : Case
Expression: SEL
Statements When 0 : Node 2
Statements When Others: Node 4
```

3.3.3.4 Loop Statements

The input_set of the nodes representing loop statements consists of node-ids representing statements in the loop. Also, these nodes are accompanied by a condition node(in case of *while* loop) or a range node (in case of *for* loop). The *range* node is used to depict the range used for the iteration of the loop. Following is such the representation for the two loop constructs. Such a loop is represented in the CDFG as shown in figure 3.9..

VHDL Statement:

while(E >= 0) loop

C := A and B;

S <= D[0] or D[1];

end loop;

Textual CDFG:

Node_Id: 8

Type : Comparator

Input1 : E

Input2 : 0

Node_Id: 9

Type : Loop

Condition: Node 8

Statements In Loop : Node 2, Node 4

(v).Subprogram Calls: The input_set of such a node would consist of the name of the subprogram called and the parameter list passed to that subprogram. The type is simply ”subprogram_call”. The resulting format is shown below:

VHDL Statement:

VECT2INT(DATA);

Textual CDFG:

Node_Id : 10

Type : Subprogram Call

New_Control: VECT2INT

Parameters : Data

3.3.4 Output Formats

The conversion tool was developed in this work with an objective to ease the process of generating an intermediate format from the behavioral description which is used as the input to the high-level synthesis. [93] [83] and [57] describe such intermediate formats used for synthesis. However, there is no standard representation for such an intermediate format [95] and researchers across the world employ different formats. Therefore, we have come up with different output formats to pertain to the needs of different users. Apart from the *node representation* of the flow-graph described in the previous sub-section, the other representations generated by the tool are described here:

3.3.4.1 Adjacency-List Representation

Probably, the most widely used form of representation for the CDFG is the adjacency list representation in which a linked-list structure is used to describe the edges of the graph. Typically, a node N_j is present in the linked list associated with node N_i if there is a directed edge from node N_i to node N_j in the graph. The adjacency list representation for the CDFG of figure is shown in figure 6(a). The dependency relations can be extracted through a simple depth-first traversal of the list structure.

3.3.4.2 Set Representation

In this, the CDFG is represented as a 2-tuple (V,E), where V is the set of nodes and E is the set of all edges in the CDFG. Set E is made up of tuples of the form (N_i,N_j) if there is a directed edge from node N_i to node N_j.

3.3.4.3 Visual Representation

The flow information and the dependencies in the CDFG are best understood through a visualization of the graph. A simple textual visualization of graphs is often too confusing and unreadable, especially when the graphs are huge. The *Visualization of Compiler Graphs (VCG)* tool described in [37] can be used to obtain a graphical view of the CDFG. Our tool generates an output file in the *Graph Description Language (GDL)* used by the VCG tool to produce a visualization of the

CDFG. The graphical view obtained with the VCG tool is similar to the CDFG depicted in figure 3.3..

3.4 Unhandled Features

Our tool can efficiently handle most of the VHDL constructs as specified in the IEEE standard 1164, but fails to include a small subset of the VHDL. This subset includes,

- Wait Statement
- File declaration: the concept of files is not used in hardware.
- Alias declaration
- Attribute declaration and attribute specification: The attribute specification is mostly used to annotate entities. An attribute specification, in conjunction with attribute declaration, allows the user to define and specify attributes. These attributes could be used to allow the user to provide information to the synthesis system. For example, time constraints could be indicated using attributes.
- Configuration specification and declaration.

It has been observed that such constructs are least likely to be used in most of the VHDL synthesis problems, and therefore, do not question the applicability of our tool in the synthesis flow.

3.5 Summary

We have presented a tool for the conversion of a behavioral VHDL description into its corresponding control and data flow graph representation that captures all the flow information of the original description. Experimental results presented in chapter 6 show that the tool is quite efficient in converting various VHDL constructs into their corresponding flow-graph representations. The output files generated by the tool are expected to meet the requirements of various researchers across the world. With its accuracy and speed in conversion, our tool would be a significant aid to speed-up the initial steps in the VLSI design flow.

CHAPTER 4

SCHEDULING THE CDFG

In this chapter, we describe a scheduling approach, which is based on the Tabu search method, and is aimed at reducing the number of resources under the given time constraints.This scheduling algorithm is an adaptation of the Tabu-search based scheduling algorithm described in [6], that uses a new mathematical formulation based on penalty weights. This algorithm is applied on the CDFG representation obtained from the first module to find the optimum number of resources that would be needed for the design.

4.1 Introduction

Scheduling and allocation are two important steps in the synthesis process after translating the algorithmic description into an internal representation. Scheduling is the process of assigning a control step to each operation of the Control and Data Flow Graph. A control step refers to the clock cycle in which the corresponding operation would be executed. Allocation is the process of assigning various functional units to operations, storage units to values, and buses to data transfers. A control unit could then be synthesized to synchronize the execution of operations based on the way that operations are scheduled and the hardware units allocated.

A number of scheduling and allocation methods have been proposed so far. Amellal and Kaminska [6] enumerate certain quality measures that could be used to evaluate the performance of scheduling and/or allocation algorithms:

- The quality of the solution produced; an optimal or a sub optimal design.
- The complexity of the algorithm; the CPU runtime.
- The solution space exploration.
- The possibility of handling large applications efficiently.

- The controllability of the synthesis process through designer constraints: area, delay, design rules, power consumption, etc.

- The possibility of predicting, with a maximum degree of accuracy, the previous parameters at a high level.

The scheduling approach used in our synthesis system is derived from the one described in the TASS synthesis system [6]. In this approach, a time-constrained scheduling is performed based on a popular mathematical optimization technique called the Tabu Search. The mathematical formulation of this approach is developed in a way to overcome some of the limitations of previous algorithms. The Tabu search technique provides for an efficient and fast solution space exploration. Upon obtaining a satisfactory schedule, hardware units are allocated to the graph. From the scheduled and allocated graph, an RTL structure could be extracted and a Finite State Machine synthesized for the controller.

This scheduling approach differs from other techniques since it is based on a control and data flow graph model, rather than just a Data Flow graph, where the control flow is implemented using more powerful procedures than existing ones. Moreover, the representation of the whole behavioral description by a single CDFG results in a low-synthesis CPU runtime, as well as an optimized design due to the availability of global information in a single graph. Also, the mathematical formulation used in this approach is based on penalty weights instead of cost estimation.

A complete formulation of the scheduling problem is given as a mathematical description which takes into account almost all the area parameters, without any increase in the complexity. The cost functions used in most of the other scheduling approaches impose a restriction on the solution space by fixing the Functional Unit(FU)s performing each type of operation. With the scheduling approach adopted in our synthesis system, module selection could be performed after scheduling in order to better optimize the design that is to be generated.

The Tabu Search technique used in the current scheduling approach is a meta-heuristic procedure originally developed by Glover for solving combinatorial optimization problems [34], in which heuristics are applied at different levels and certain un-optimal moves are made to come out of locally optimal solutions. The application of this technique to a number of large and difficult combinatorial optimization problems [34] has shown that the Tabu Search technique is faster and

more efficient than the better known optimization techniques like Simulated Annealing for finding optimal solutions.

Most of the ILP-based synthesis systems [97] [66] [23] characterize the cost of the hardware units needed by the scheduling. The ILP models tend to simplify the objective function, sometimes including only the functional units [97] , often ignoring the area cost of storage, interconnections and control circuitry. The inclusion of these parameters increases the complexity of the formulation and would not be manageable for large size applications. Also, their search space is restricted by their rigid cost functions and the prior specification of Functional Units for each operation. Better solutions could be obtained by considering the availability of different ALUs for executing the same type of operations. The area cost could be reduced by executing different operations by the same Functional Unit and when more than one ALU is available to execute the same type of operation.

The mathematical formulation of the current scheduling approach is based on penalty weights rather than on cost evaluation. These penalty weights, that include the real costs of the hardware units, make it possible to take into consideration different area parameters of the design. An objective function based on these penalty weights can be used to optimize the number of functional and storage units, as well as the number of interconnections.

Most of the scheduling algorithms proceed by assigning control steps to operations one at a time, and hence, their results are strongly affected by the order of such assignments. The ILP method, however, being a global method, is more suitable for scheduling control and data interdependent nodes than other scheduling approaches like the list scheduling. The transformational approach used in our system begins with an initial scheduling, and then the nodes are moved in order to minimize a defined objective function, which is defined so as to reduce the area required by the current schedule. The assignment of control steps to nodes is done simultaneously instead of a single one at a time. In order to avoid the inflexibility associated with other cost-based formulations, a more complete and efficient formulation is developed taking advantage of the use of real unit costs.

The objective function for our scheduling approach is given as follows:

$$f = \sum_{i} W_i, \tag{4.1}$$

where each weight W_i is a cost-based penalization of a worst case assignment of operation nodes requiring the greatest amount of a given hardware resource [6] .

The scheduling approach, adopted from [6] , is formulated as follows: Given the number of control steps K,

$$\text{Minimize } f(y1, y2, ..., yn) = \sum_i W_i, \tag{4.2}$$

satisfying the constraints,

$$S_i \leq y_i \leq L_i, \qquad \text{for } 1 \leq i \leq n, \tag{4.3}$$

$$y_i + d_i \leq y_j, \qquad \text{for } (i,j) \| i- > j, 1 \leq i \text{ and } j \leq n, \tag{4.4}$$

where, n is the total number of nodes of the control and data flow graph, y_i is the control step to which node i is assigned, S_i is the As Soon As Possible step and Li is the As Late As Possible step of node i, d_i is the propagation delay associated with an operation node i, and, $i- > j$ denotes node j is data dependant on node i.

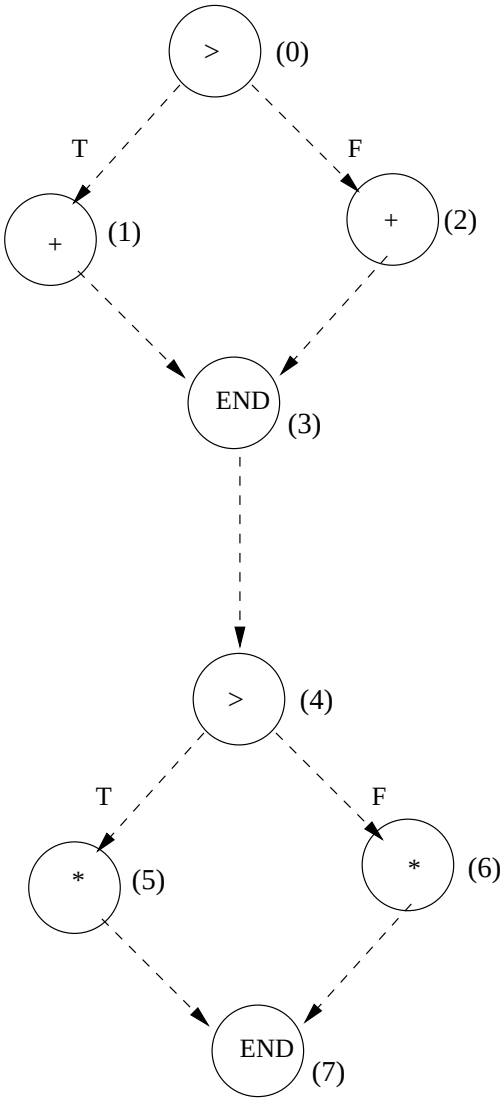
The penalty weights Wi are selected in such a way that minimizing these weights would optimize the assignment of operation nodes, reducing the number of resources of a given type required for these operations, while utilizing the mutual exclusion among nodes in a control step arising from control dependencies.

4.2 Mutual Exclusion Among Operations

The control dependencies inherent in a CDFG induce certain amount of mutual exclusion among the operations in the CDFG. The mutual exclusion property could be exploited for potential resource sharing among such operations.

Definition : Two nodes, N_i and N_j, in a CDFG, are said to be mutually exclusive, if and only if, they cannot exist in any execution path of the CDFG at the same time.

It can be clearly seen that the nodes N_i and N_j have to be present in different branches of the same conditional block.



(0)
T
F
+
(1)
+
(2)
END
(3)
>
T
F
*
(5)
*
(6)
END
(7)

Figure 4.1.. Mutually Exclusive Nodes

Consider the CDFG with a two conditional statements shown in figure 4.1.. Four possible execution paths exist for the given CDFG, which are, (0) (1) (3) (4) (5) (7), (0) (2) (3) (4) (5) (7), (0) (1) (3) (4) (6) (7) and, (0) (2) (3) (4) (6) (7). We can see that nodes (1) and (2) cannot exist together in any of these paths, and hence, they are mutually exclusive. Similarly, nodes (5) and (6) are mutually exclusive. However, node (2) is not mutually exclusive with node (7), and neither is node (3) with node (6). Hence, only nodes in the same conditional block can be mutually exclusive.

We observe that nodes (1) and (2) can be scheduled in the same control step with a single adder allocated to them, since we know that only one of them would be executed. Similarly, nodes (5) and (6) could be scheduled in a control step with a single multiplier allocated to them. Hence, it is advantageous to schedule two mutually exclusive operations of same type in the same control step.

The CDFG representation, described in the previous chapter, helps in determining all possible mutual exclusions among the operations in the CDFG, which are then used in the scheduling algorithm described in the later sections.

4.3 Penalty Weights

As discussed earlier, the objective function for the scheduling algorithm is based on a set of penalty weights that penalize any assignment taht requires more hardware resources. Four factors that could affect the number of resources are considered:

- For each operation type m, the control step that has maximum number of operation nodes of type m is penalized, since that dictates the maximum number of resources executing operations of type m. Such a penalty weight is calculated as:

$$W_1 = \sum_m (C[m] * maxK_{ms} \| 1 \leq S \leq k) \tag{4.5}$$

  where, K_{ms} is the maximum number of operations of type m that can be executed in control step s, $C[m]$ is the cost of function unit executing operation of type m, and po is the maximum number of operation types. The value of K_{ms} is obtained by considering the number of non mutually exclusive operations of type m in each control step.

For the CDFG depicted in figure 4.2., there are a maximum of 2 multiplications in the control step 5, and a maximum of 2 additions in control step 2. Hence, the penalty weight associated would be W = 2 * cost[*] + 2 * cost[+].

- The next weight penalizes control steps which have a maximum number of non mutually exclusive operations of same equivalence class.

  Definition : Two operations of type i and j are said to belong to the equivalence class m, if and only if, both of them can be executed by a functional unit of type m.

  For example, addition and subtraction operations could be performed by a single adder / subtracter module.

  Two operations of type i and j belonging to equivalence class m have to be executed by 2 functional units of type i and j when assigned to the same control step, but, they could be executed by a single functional unit of type m, when assigned to different control steps. The penalty for assigning them to the same control step is calculated as,

$$C[i] + C[j] - C[m] \tag{4.6}$$

  For the CDFG in figure 4.2., the cost of assigning both addition and subtraction in same control step 4 is given as 1 * (C[+] + C[-] - C[+/-]).

- Another penalty weight is associated with the life times of weights. Whenever a node j uses the output of node i, the number of control steps separating i and j is the lifetime of the variable storing that value, The penalty factor for storing such values is calculated as:

$$W = \sum_i \sum_j [Su_{ji} * (y_j - y_i - d_i) + Su_{ij} * (y_i - y_j - d_j)] * cost[storage]. \tag{4.7}$$

  where, Su_{ij} = 1, if $i -> j$, 0, otherwise.

  In figure 4.3., the output of node (1) is used by nodes (2) and (4). Node (4) is 2 control steps away from node (1), and hence, it requires the output to be stored, leading to a weight of 1 * cost[storage].

- The final penalty weight is associated with the number of buses needed, which depends on the maximum number of distinct inputs used in a control step. In figure 4.3., the maximum number of inputs is 4 (used in control steps 1 and 3). Hence, the penalty weight associated with it is 4 * cost[bus].

The development of objective function based on the above mentioned penalty weights adds flexibility to the scheduling algorithm.

4.4 Scheduling Algorithm

A pseudo-code for the Tabu-search based scheduling algorithm, extracted from [6] is shown in figure 4.4..

The ASAP and ALAP algorithms are employed initially to obtain the least and greatest control steps for each node in the CDFG. The output from either of these algorithms could be chosen as the initial solution S. Such a solution is then evaluated based on the penalty weights described in the previous section.

The algorithm then undergoes a number of iterations, in each of which, a best solution is selected from the neighborhood of current solution. If such a solution is better than the best one found so far, then it is saved as the best solution. The new neighborhood is calculated from the new solution. If this solution was visited during the previous iterations, it is discarded to prevent cycling of solutions. The iterations are repeated until a fixed number of iterations pass without any improvement in the solution. This whole procedure could be repeated by changing the time constraints until a good schedule is obtained.

The neighborhood of a solution is obtained by moving nodes from their current control step to another control step within the ASAP and ALAP values of that node. The nodes considered for movement are chosen from the control steps that contribute most to the penalty weights in the current solution. Such nodes are moved to control steps where there are lesser number of nodes of same operation type and of same equivalence class.

With the large solution space explored by the Tabu search method, the scheduling algorithm would be able to perform global optimization of the number of resources used. Some experimental results to prove the efficiency of this approach are presented in chapter 6.

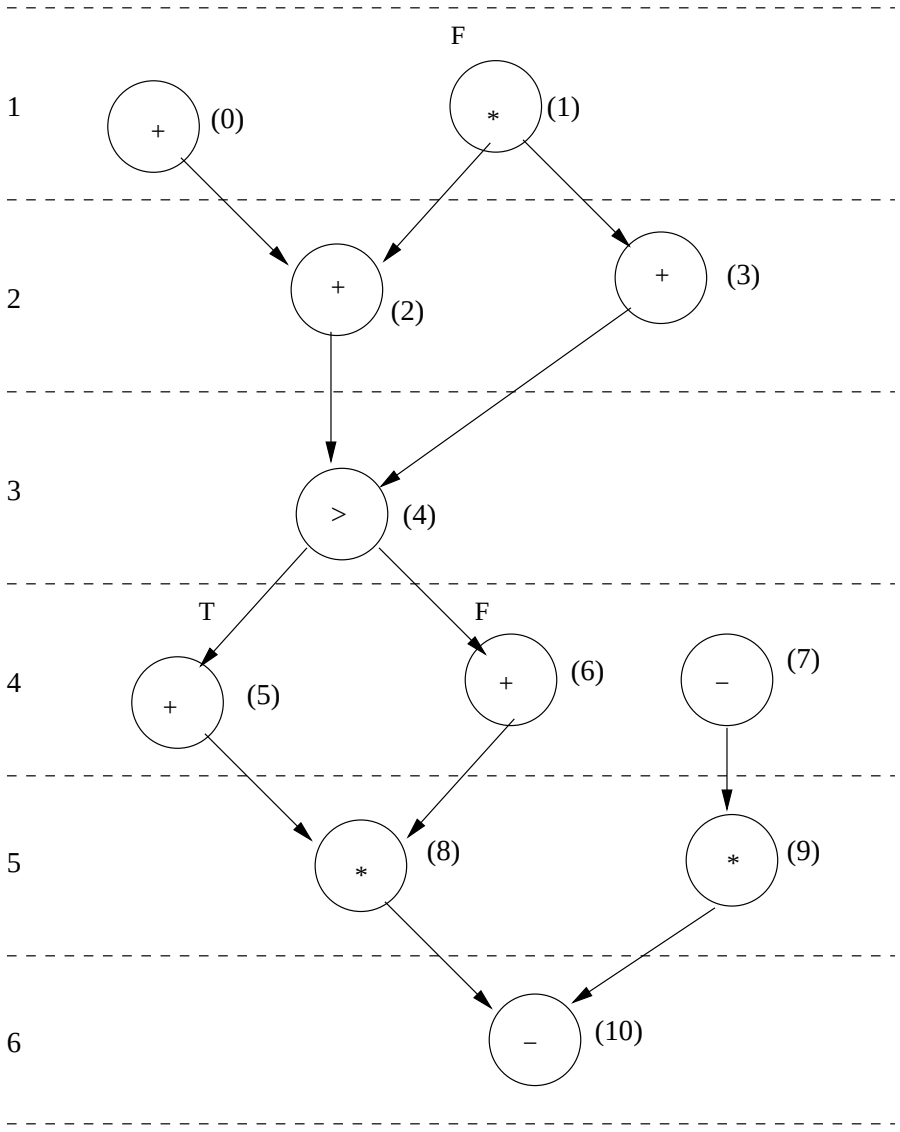


Figure 4.2.. Penalty Weights

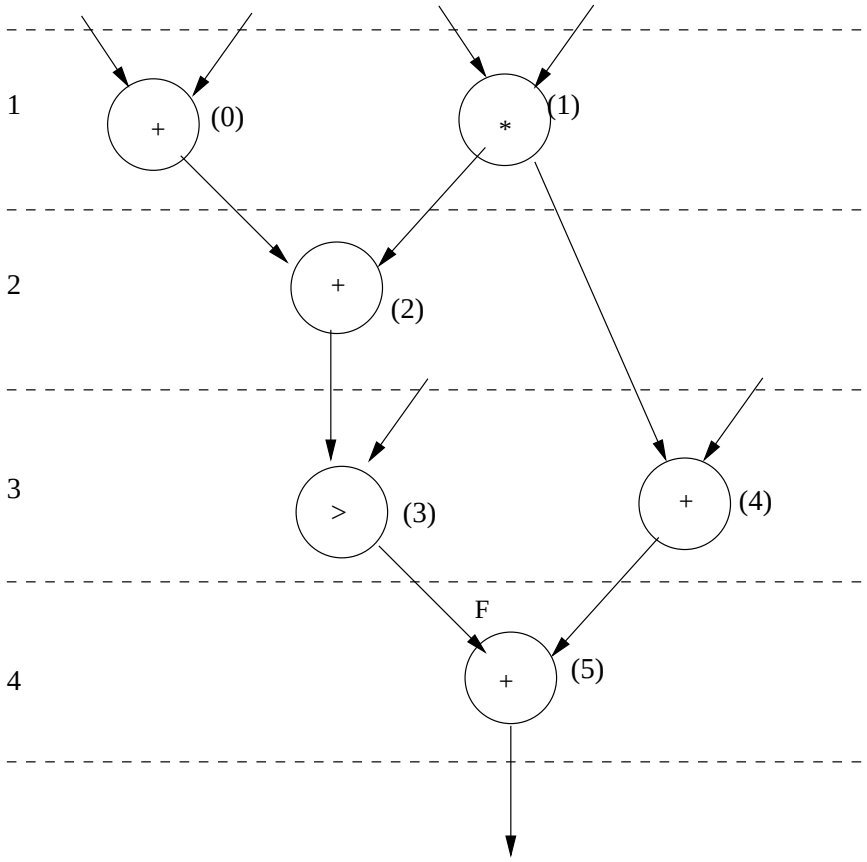


Figure 4.3.. Life-Time and Number of Buses From CDFG

(01) begin
(02) Determine ASAP and ALAP times for each node in CDFG
(03) S ← Generate initial solution
(04) $while$ fewer than $MAX_ITERATIONS$ have passed without
. any improvement on the best solution
. do
. $S_n \leftarrow$ choose best solution from neighborhood of S
. if S_n not visited in previous iterations
. then
. if S_n better than best solution found,
. then save S_n as the best solution
. $S \leftarrow S_n$
. else
. discard S_n
. $endwhile$
(05) end

Figure 4.4.. Scheduling Algorithm

CHAPTER 5

POWER-OPTIMIZED BINDING

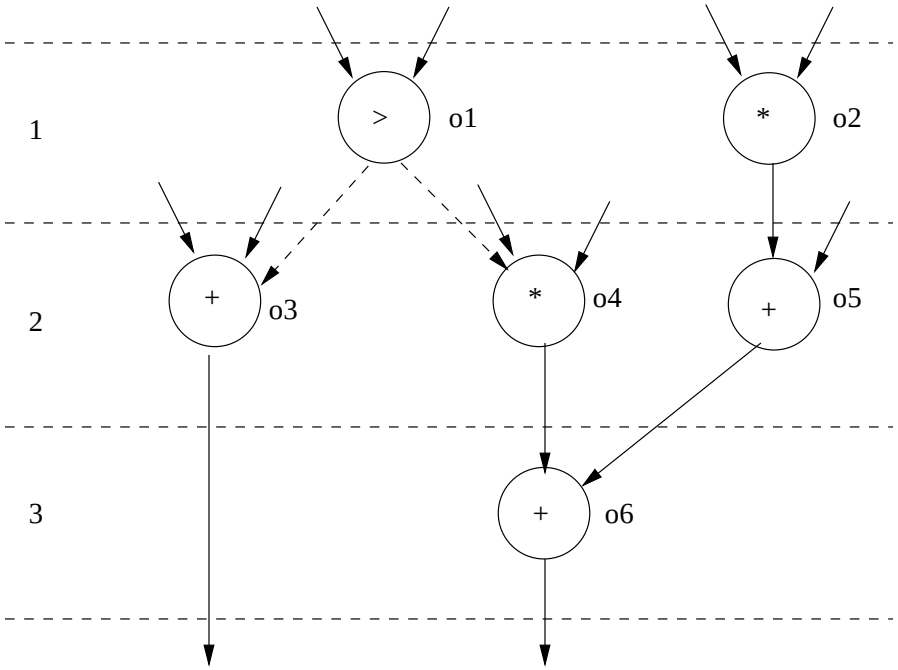
In this chapter, we describe a game-theoretic binding algorithm for power optimization. An approach, based on Game Theory and Auction Theory, was proposed by Murugavel et al. in [75] for minimizing the average power of a circuit during scheduling and binding in high-level synthesis. They formulated these two problems as auction-based non-cooperative games, the solutions to which were obtained using the theory of Nash Equilibrium. We have adopted this approach of binding and extended it to include control constructs in order to apply it to the scheduled CDFG (S-CDFG) obtained from the previous module.

5.1 Power Optimization During Binding

The concept of power reduction has been taken to higher levels of abstraction with the automation of the design process. In fact, power consumption is given special attention in the behavioral synthesis, because the decision made at this level can have significant effects on the final design. Also, previous research works indicate that designing at higher levels of abstraction leads to a greater potential for power reduction [99].

Power reduction could be considered at various levels of the high-level synthesis process. Specifically, the reduction of power due to switching activity could be addressed while binding the functional units to operations in a scheduled Control and Data Flow Graph (S-CDFG). Here, we formulate the problem of power-optimized binding using the Game Theoretic approach [76].

The binding algorithm is applied on the scheduled CDFG to obtain a *binding matrix*. The *binding matrix(B)* is a matrix ordered by the control steps and functional units with each entry being either a '0' or a '1'. An entry $B(i, j)$ takes the value '1' if functional unit j is bound to an operation in control step i. A scheduled CDFG and its corresponding binding matrix are shown in figure 5.1.. Power optimization is achieved through the concept of functional unit sharing, in which,



(a) Scheduled CDFG

Control step	Functional Units				
	comp1	mult1	add1	add2	add3
1	1	1	0	0	0
1	0	1	0	1	1
1	0	0	0	0	1

(b) Corresponding Binding-Matrix

Figure 5.1.. Scheduled CDFG and its Binding Matrix

neighborhood operations with at least one common input are assigned to the same functional unit so that the number of changing inputs are reduced [101].

5.2 Basic Concepts

We now give a brief introduction of the general game theory and the Auction Theory used for the formulation of binding problem.

5.2.1 Game Theory

Game theory was developed as a distinct approach by Neumann [76] to study the interaction among rational humans, each trying to maximize his or her profits in the given circumstances. Initially, the games were considered to be *non-cooperative*, in which each player maximizes his or her reward irrespective of the results of others. Nash [46] later extended the game theory to include *cooperative games*, in which the players agree upon a specific set of rules for the game and coordinate their strategies to obtain the best result for the whole group. The players in a non-cooperative game cannot collaborate and hence the profits are decentralized among them. The formal agreements among the players in cooperative games, which are enforced by an external entity, centralize the cost function among the players as opposed to the non-cooperative games in which the control function is decentralized among the players trying to achieve optimization at individual level.

In an n-player game, each player i is provided with a set of alternatives (or strategies) S_i, from which he can choose his move, and a payoff function p_i that determines the player's payoff for choosing a particular strategy. Each player i tries to apply a strategy $s_i \in S_i$ such that his or her payoff is maximized in the given circumstance, which is described by the set of strategies selected by the other $n-1$ players. The game is played until it reaches a stable point where the strategy of each player is optimal when compared to those of others.

Nash has proposed a stable point solution, called the Nash Equilibrium [46], at which no player can improve his payoff by deviating alone from that point. In other words, Nash equilibrium is a set of actions, $(s_1^*, s_2^*, ..., s_n^*), s_i^* \in S_i$ for each player, such that no player can gain more by choosing an action s_i, different from s_i^*, when it is known that each other player j (j $\neq$ i) is adhering to his or

her strategy s_j^*. Such an equilibrium can be calculated based on the number of players in the game and the strategies available to each player. The Nash equilibrium point indicates the most optimal solution for all players in the game.

5.2.2 Auction Theory

Auction theory [49] is the study of selling objects among different buyers in a way that is optimal for both the seller and the buyers. Each buyer *bids* a value for the object, which is the value that he is willing to pay for that object, and the seller selects the best bid from the available set of bids. Several methods have been proposed for the auction theory, some of which, are the English auction, the Dutch auction, the first-price sealed-bid auction and the second-price sealed-bid auction.

While applying the auction theory to the binding problem, the power consumption of each module is used as the bid value. The bid value is changed during the application of the English, Dutch or the second-price sealed-bid auction models. However, since the power consumed by a module for an operation is a constant value, none of these models can be applied for binding. Hence, the most suitable model is the first-price sealed-bid auction model, in which the bids are sealed and cannot be changed once submitted to the seller.

5.3 Problem Formulation

It is suggested in [46] that a very wide range of situations, like firms, action prices, etc., may be modeled as strategic games. All such situations are characterized by the presence of a fundamental conflict among the players. A conflict arises in the binding problem too, with the availability of more than one module for a single operation type. Hence, it is proposed to capture such a conflict using the game theory. The game theoretic model captures interactions among players by allowing each player to be affected by the actions of all players and not just by his/her action alone. This quality quite suits the problem of binding since the binding of an operation is affected by that of other operations of same type.

Applying Game theory and Auction theory, the low-power binding problem is transformed into a problem of finding the Nash equilibrium for the bidding strategies auction problem [75].

The available functional units are considered as the sellers and the operations as buyers in the auction problem. For an operation O_i, each functional units submits its power consumption for that operation as its sealed bid value and the lowest bidder is bound to that operation.

The game-theoretic algorithm described in [75] was aimed at data-flow descriptions, and hence, their approach is applicable for scheduled data flow graphs (S-DFGs). However, since the CHESS tool is intended to deal with both control and data flow descriptions, we extend the algorithm described in [75] to make it applicable to a scheduled CDFG (S-CDFG). The extension stems from the inclusion of control nodes in the graph and by considering the corresponding functional units as sellers in the bidding problem. Also, the approach for S-CDFGs includes the flexibility of binding the same functional unit to two mutually exclusive operations of same type in a control step. The cost functions for each functional unit are evaluated in the same way as described in [75].

The cost functions associated with each module are calculated initially and are assumed to be available to all the operations in the S-CDFG. The cost functions are chosen so as to represent the power consumption of that module for that operation. It is calculated as the difference between the switching cost (P_{sw}) and the cost due to changing inputs (P_{inp}). Since each module is considered to be a two-input module, the number of changing inputs could be one, two or none. Also, the switching cost (P_{sw}) is calculated as the power consumption due to two changing inputs. The values P_{inp} and P_{sw} of the available modules are determined through simulations. Thus, the cost of executing an operation o on module m is given as,

$$C(m, o) = P_{sw} - P_{inp}. \tag{5.1}$$

The binding strategy is applied individually to each control step independent of the other control steps. Moreover, it is assumed that a module can be assigned to only one operation in each control step. Given a set of modules $m_1, m_2, ..., m_{x_j}$, and the set of operators $o_1, o_2, ..., o_{y_j}$, a cost matrix is obtained initially, which gives the cost of all possible binding pairs among them. The cost matrix, CM is represented as,

$$CM = \begin{bmatrix} c(1,1) & ... & c(1,y_j) \\ ... & ... & ... \\ c(x_j,1) & ... & C(x_j,y_j) \end{bmatrix}$$

where, c(a,b) gives the cost of binding a module m_a to operation o_b.

Given a set of modules $M_i = (m_1, m_2, ..., m_{x_i})$ and the set of operations, $O_i = (o_1, o_2, ..., o_{y_i})$, where x_i is the number of modules compatible to operations of type i, and y_i is the number of operations of that type, we consider the feasible sets of module-operation pairs, and arrive at the most optimal one using the Nash equilibrium. If S is the set of all possible module-operation pairs, then tehre could be many feasible A sets, such that $A \in S$, like $(m_1, o_1), (m_2, o_2), ..., (m_{x_i}, o_{y_i})$. The total cost associated with such a set A, is given as, $C(A) = \sum_{k=1}^{y_i} C(m(o_k), o_k)$,

where $m(o_k)$ is the module assigned to operation o_k.

The problem of binding is now reduced to finding a feasible set A^*, such that, C(A^*) = min C(A), and, we claim that such a set A^* is the Nash equilibrium for the module-operation pairs, since, at that point, none of the modules can improve its cost with a different assignment, or, in other words, by deviating form the stable point.

If we assume that there are three adders, one multiplier and a comparator available for the S-CDFG shown in figure 5.1., then the bidding strategy could be applied to find the optimal binding for the three *add* operations. Since there is only one multiplier, it would be bound to the operations o_2 and o_4. Similarly, the comparator is assigned to the operation o_1. However, we have a choice of three adders for the two add operations o_3 and o_5 in control step 2. If A_1, A_2 and A_3 are the three adder modules, then the cost-matrix for these operations would be written as,

$$\begin{bmatrix} C(A_1, o_3) & C(A_1, o_5) \\ C(A_2, o_3) & C(A_2, o_5) \\ C(A_3, o_3) & C(A_3, o_5) \end{bmatrix}$$

Each of the modules A_1, A_2, and A_3 bids for the operations o_3 and o_5, and the Nash equilibrium of such a bidding strategy gives the optimal binding solution. When the same strategy is applied for control step 3, the Nash equilibrium gives the same module assigned to operation o_5 as the optimal one for o_6, since it reduces the number of input changes at that module.

The binding problem, once formulated as an auction based non-cooperative finite game, is solved for the Nash equilibrium with the aid of a computational game solver, named Gambit[71]. Gambit is a tool for solving finite games, which takes, as its inputs, a matrix representation of a strategic form game, and arrives at all the Nash equilibriums for that game through an iterative elimination of strongly dominated strategies.

The binding strategy at each control step is treated as a separate game in itself and hence, our algorithm needs to be applied individually at each control step in the appropriate sequence. The effects of the bindings from previous control steps are considered during the calculation of the pay-off matrix for the current step.

5.3.1 Algorithmic Description

The pseudo-code for the Game theoretic binding algorithm is given in figure 5.4.. The algorithm takes the S-CDFG and gives a binding matrix for that graph. Since the bidding strategy is applied separately for each control step, the Nash equilibrium has to be calculated for each set of applicable modules in each control step. Thus, the number of times the Nash equilibrium is applied depends on the number of control steps in the S-CDFG as well as the number of modules applicable for an operation in that control step.

The evaluation of the Nash equilibrium using the Gambit tool requires the set of all possible strategies in the form of a matrix. When applied to our problem, this corresponds to a matrix representation of the costs of all possible module-operation pairs in that step. A polynomial-time algorithm for obtaining such a matrix representation is described in figure 5.3.. It can be seen that the effective cost includes the cost due to changing inputs based on the previous bindings. The power and delay based cost matrix at each iteration is calculated by considering each operation o_i in a control step and all the compatible modules for the operation o_i. Each entry (i,j) in a the matrix lists the absolute and cumulative power values of assigning a module m_j to the operation o_i along with the cumulative delay of assigning that module to operation o_i. The power values are calculated based on the equation 5.1, taking into consideration the number of changing inputs for each module assignment.

Give the strategies available for each player and the number of alternatives a_i of each player, the Nash equilibrium can be obtained through the following procedure,

- determine all the possible outcomes for the game based on the set of alternatives, a_i, for each player i.
- for each player i, derive the inequality showing his/her equilibrium poitn, a deviation from which would not result in any increase of gain for that player. Such an inequality is given as,

input : Cost Matrix C, Number of players N, set of strategies, S for each player
output: Nash Equilibrium Solution NE

(01) begin
(02) $NE \leftarrow \{\}$;
(03) for each player i do
(04) for each set of strategies ($n_1 \in S_1$),...,($n_N \in S_N$) do
(05) Calculate $p^i_{(n_1,\ldots,n_N)} = C[i, n_i]$;
(06) end for
(07) for each strategy ($n_i \in S_i$) do
(08) $n_i^* \leftarrow$ Nash equilibrium for player i satisfying the inequality,
(09) $p^i_{(n_1,..,n_i,...n_N)} \geq p^i_{(n_1,...,n_i,...,n_N)}$;
(10) end for
(11) $NE \leftarrow NE \bigcup n_i^*$;
(12) end for
(13) end

Figure 5.2.. Algorithm for Finding the Nash Equilibrium

- the set of strategies satisfying all the inequalities from step (ii) gives the Nash equilibrium for the model.

The most time-intensive step of the algorithm lies in finding the Nash equilibrium, the algorithm for which is given in figure 5.2.. It is stated in [80] that there is a guaranteed existence of an equilibrium point when a number of alternative strategies are considered. Due to this, it is shown that the complexity of an algorithm for finding the Nash equilibrium is between P and NP, while, the problem of finding a feasible binding solution itself is NP-complete [48].

5.4 Summary

The problem of power-optimized binding could be successfully formulated as an auction-based game and the optimal solution obtained using the concept of Nash Equilibrium. The Nash equilibrium indicates a stable point for a set of players, with the optimization function distributed among them, and hence is suitable for finding a globally optimal binding solution for the given scheduled CDFG. The efficiency of such an algorithm is discussed in the next chapter.

input : Set of operations O, set of modules M in control step i
output: Payoff matrix C

(01) begin
(02) for each module m $\in$ M do
(03) for each operation o $\in$ O do
(04) $P_{mo_i} \leftarrow$ Power to execute operation o on module m in control step i;
(05) $D_{mo_i} \leftarrow$ time delay for executing operation o on module m in control step i;
(06) $P_t \leftarrow P_{mo_i} + \sum_{k=0}^{i-1}(P_{mo_k})$;
(07) $D_t \leftarrow D_{mo_i} + \sum_{k=0}^{i-1}(D_{mo_k})$;
(08) $C[o, m] \leftarrow (P_{mo_i}, P_t, D_t)$;
(09) end for
(10) end for
(11) end

Figure 5.3.. Algorithm for Finding the Cost Matrix

input : Scheduled Control Data Flow Graph(S-CDFG), Set of modules, Power and Delay Values
output: Binding Matrix B

(01) begin
(02) for each control step i do
% Conider all the modules that can execute that operation and select the best one
(03) for each set of compatible modules M in control step i do
% Use algorithm in figure to find the Cost matrix based on equation 5.1
(04) $C(O, M) \leftarrow$ Calculate the power and delay based cost matrix
% Use algorithm in figure to find the Nash Equilibrium solution
(05) $NE \leftarrow$ Determine the Nash Equilibrium based on M, O and CM
(06) $B \leftarrow$ Represent the Nash equilibrium solution as a binding matrix B
(07) end for
(08) end for
(09) end

Figure 5.4.. Binding Algorithm

CHAPTER 6

EXPERIMENTAL RESULTS

This chapter presents a set of experimental results to verify the accuracy and efficiency of each module in the CHESS tool. Each phase of the synthesis system was coded and tested separately, and later these codes were integrated to form a comprehensive tool for synthesis. For integration of the different codes, the output of each code was made compatible for the next step. However, other output formats are available at each step in a way suitable for the user. The user has the option of multiple entry and exit points in out tool. For example, the user can either begin with a behavioral VHDL code initially, or, with a CDFG representation, if already available. In the earlier case, the CDFG for the VHDL code would be extracted by the tool and used for further steps. The modularization of the synthesis tool, combined with the feature of multiple entry and exit points, has provided us with an opportunity to test extensively and individually, the efficiency of each part, the outcomes of which are presented here.

6.1 CDFG Extraction From Behavioral VHDL

The proposed conversion tool has been implemented and tested successfully with various VHDL source files. These source files were chosen carefully to cover all possible constructs of the VHDL that the tool can handle in different combinations. It was observed that the tool could extract the flow information from these files quite accurately.

The code for lexical analysis of the VHDL code was developed using the GNU Flex, and the code for the syntactic analysis using the GNU Bison tool. The output from these is then applied to a sequence of C++ codes which extract the parse-tree and trim it to obtain the final CDFG.

We tested our tool with some of the standard HLSynthesis VHDL benchmarks. Two different classes of examples were considered for testing : operation-dominated examples and control-dominated examples. From the first class of examples, we used standard VHDL programs like *the*

Table 6.1.. Experimental Results for CDFG Extraction From Behavioral VHDL Specification

Benchmark	No of nodes				Total execution. time
	operational	control	call	storage	
diffeq	10	1	0	15	$<$ 1 sec
ellipf	26	1	0	37	$<$ 2 sec
radix512	16	3	12	23	$<$ 2 sec
fft	21	8	0	48	$<$ 2 sec
dhrc	13	1	18	33	$<$ 2 sec
controller counter	17	12	0	19	3 sec
gcd	4	5	0	10	3 sec
kalman filter	17	11	0	30	5 sec
barcode	6	18	0	26	$<$ 4 sec
display	26	2	0	37	5 sec
falsepath	4	10	0	52	$<$ 4 sec
beamformer	15	10	0	8	$<$ 4 sec

differential equation (diffeq) and the elliptic wave filter (ellipf). From the second class of examples, we used algorithms like *the Greater Common Divider (GCD), Kalman filter, Fast Fourier Transform (fft), etc.*. The number of tasks extracted from these codes and the time required to extract them have been presented in table 6.1. The number of various operational, control and other nodes give an idea of the size of the problem. The corresponding CPU times shown in the other column depicts the runtimes required for the extraction of these CDFGs. It can be seen from the table that the runtimes are reasonably short, even when the model is used for large examples.

All the experiments have been run on a 40MHz Sun Sparc station running SunOS with 256MB RAM. The results show that CDFG extraction for VHDL descriptions that are more operation-intensive is faster than that for control-intensive descriptions. The graphical representations of the CDFGs obtained for some of these circuits are shown in figures 6.1. through 6.4.. In these figures, data dependencies are shown by solid arrows, and the control edges by dashed arrows. In most of the figures, the storage nodes were not shown for the sake of simplicity.

6.2 Scheduling

The scheduling algorithm was also implemented in C++ on a Sparc station. The algorithm takes, as its inputs, the CDFG, the types of functional units available, and the time constraints. For illustrating the efficiency of this algorithm, we concentrate on two of the most-widely used

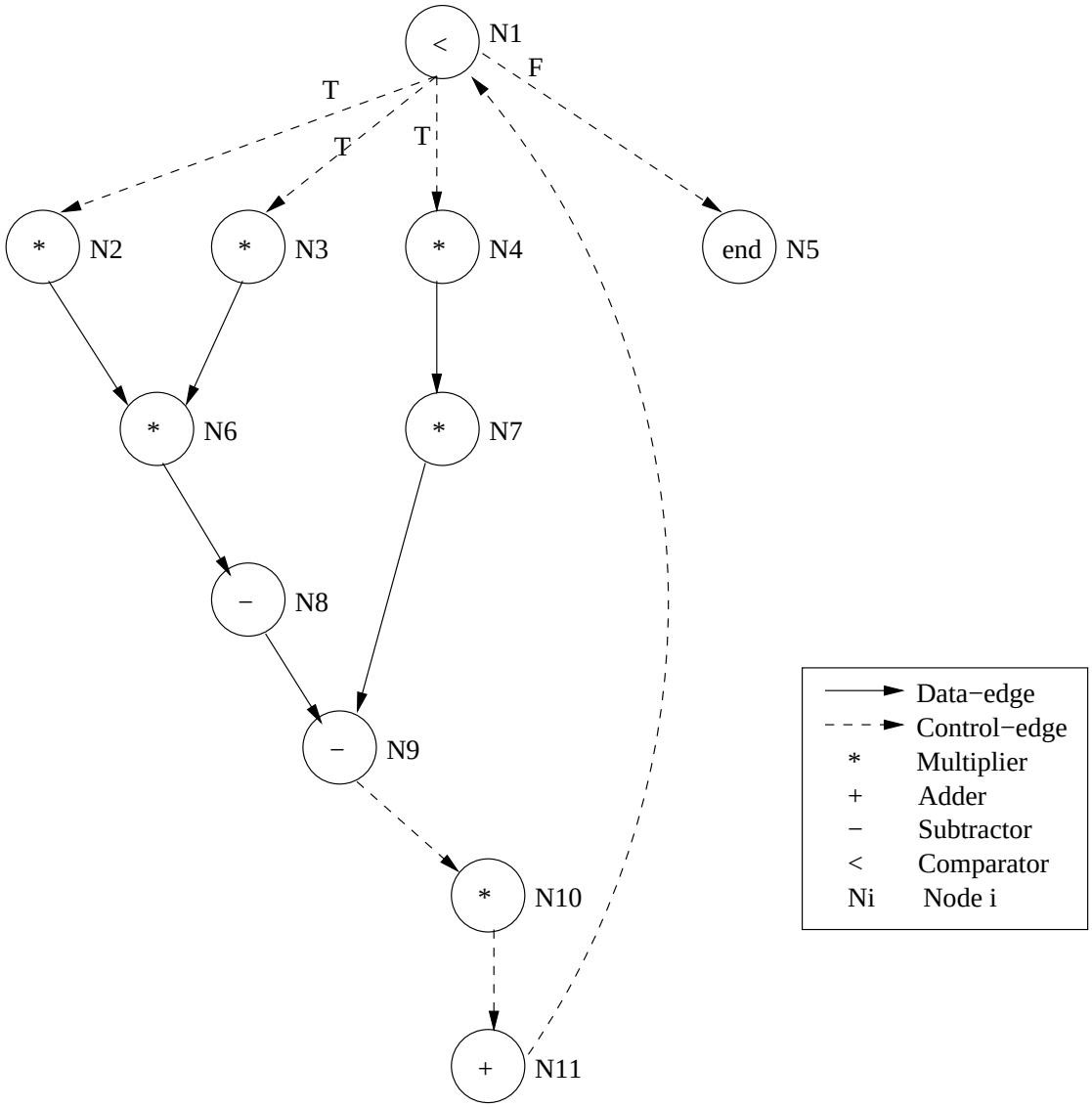


Figure 6.1.. CDFG Extracted for the Differential Equation Benchmark Circuit

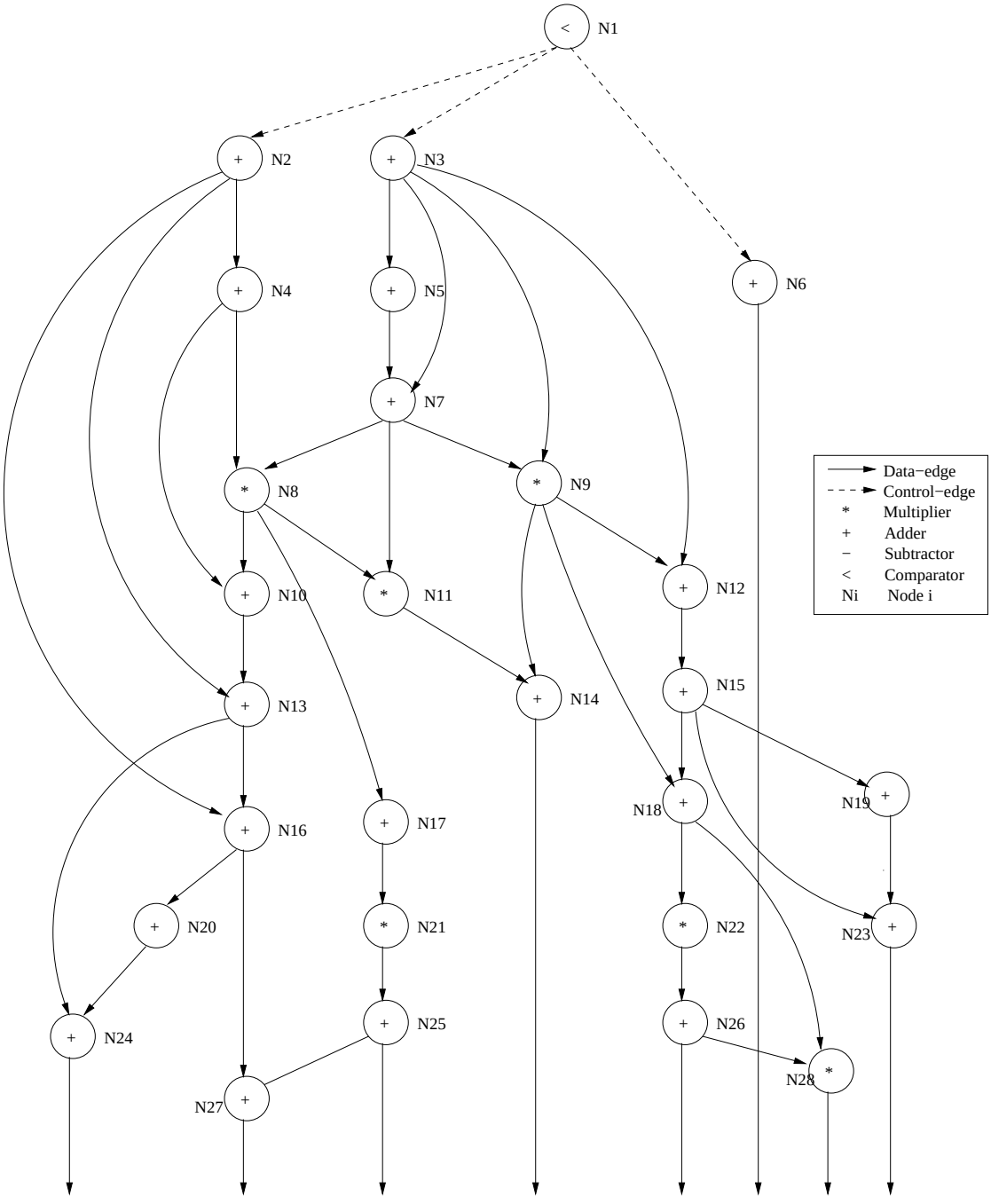


Figure 6.2.. CDFG Extracted for the Elliptic Filter Benchmark Circuit

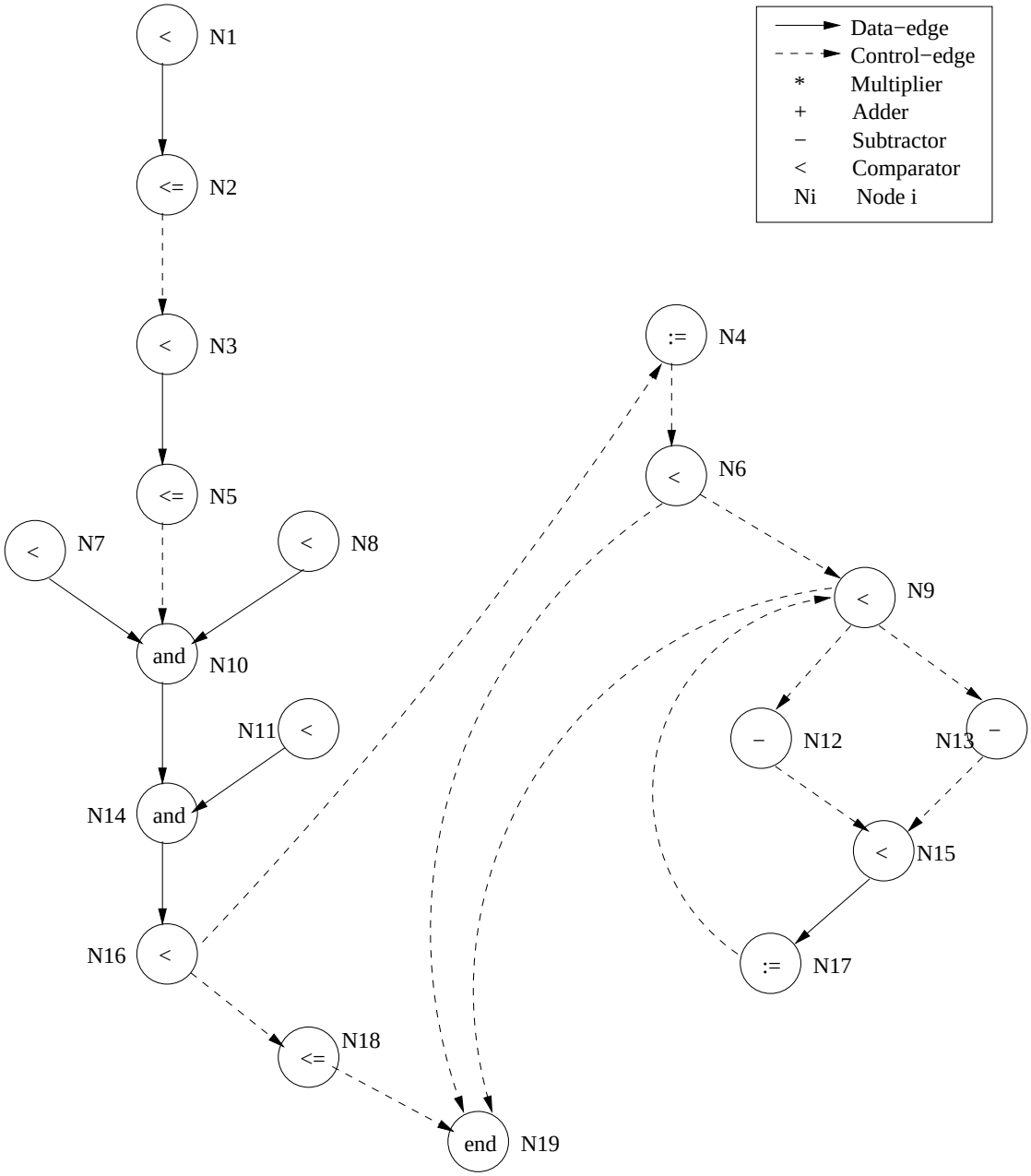


Figure 6.3.. CDFG Extracted for the Greatest Common Divisor Benchmark Circuit

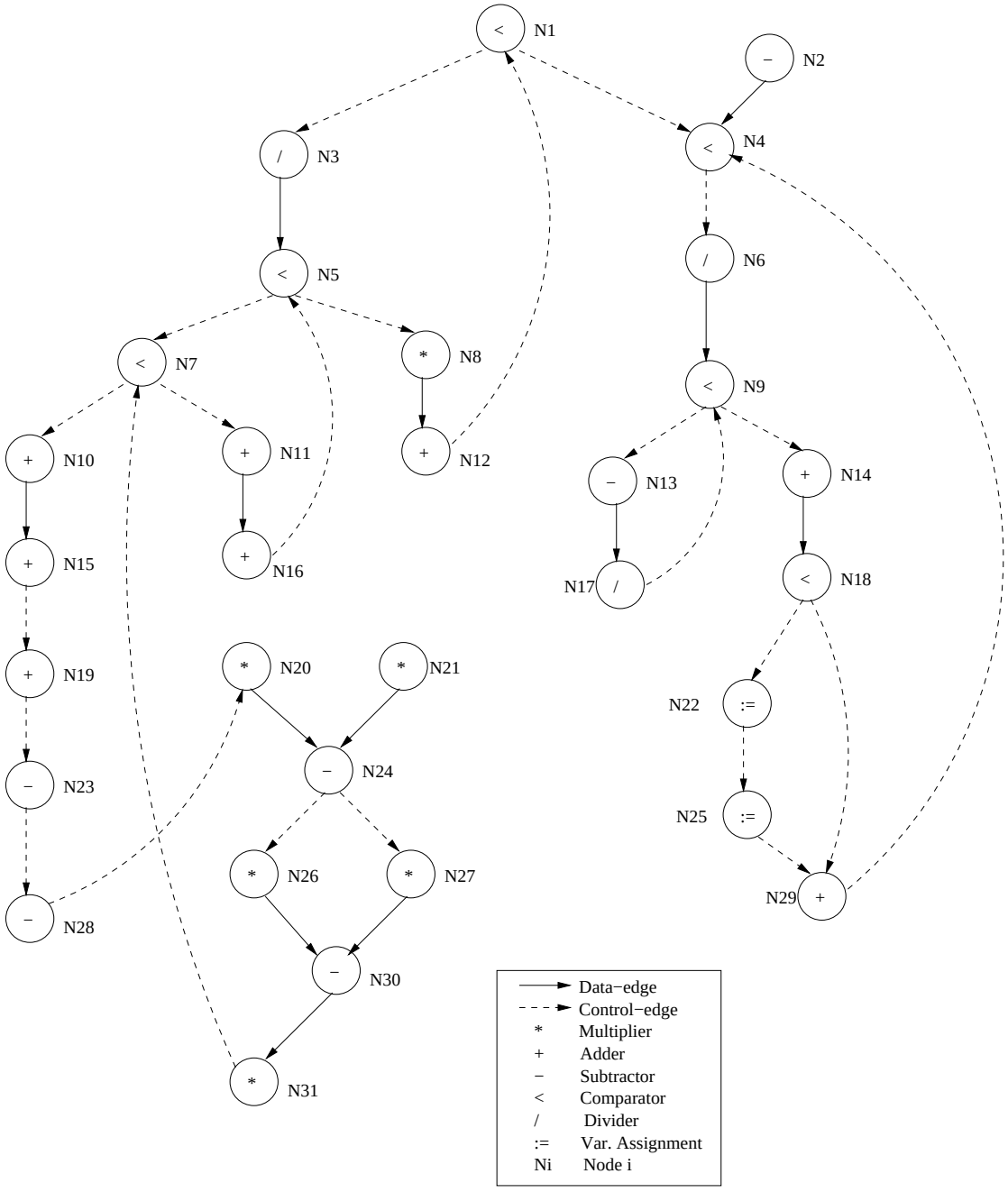


Figure 6.4.. CDFG Extracted for the Fast Fourier Transform Benchmark Circuit

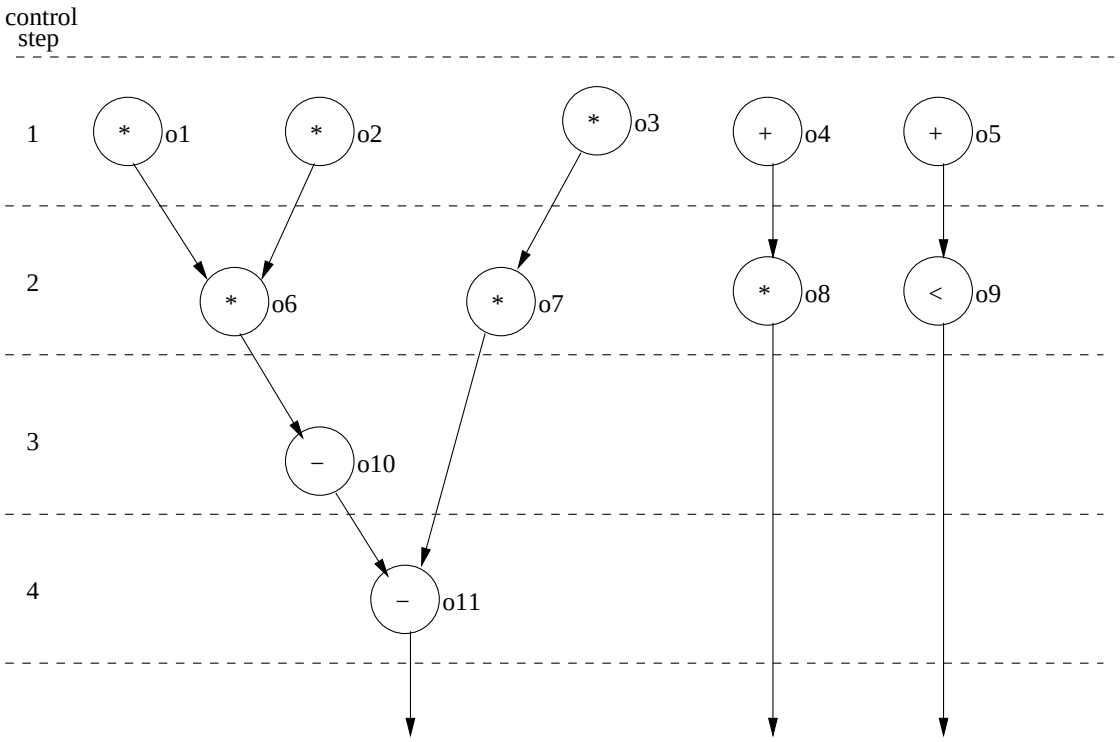


Figure 6.5.. ASAP Schedule for Differential Equation Benchmark

benchmark circuits, the diffeq and the elliptic filter, and compare the results to some of the existing systems. The following types of functional units were assumed to be available for scheduling the operations - adder, subtractor, adder/subtractor, multiplier, divider, and comparator.

6.2.1 The Differential Equation Benchmark

The ASAP and ALAP schedules for the diffeq are shown in figures 6.5. and 6.6. respectively. It can be seen that the CDFG can be scheduled in four control steps, but, requires more resources. Our algorithm tries to optimize the number of resources by using the functional unit that can perform both addition and subtraction instead of separate adder and subtractor modules (see section 5.4). Such a schedule obtained by our algorithm is shown in figure 6.7.. Also, alternative time constraints are tried to achieve further resource-optimization as shown in figure 6.8.. The number of components given by this algorithm is compared against that of HAL[85] and SPLICER [12] systems for the same benchmark circuit in table 6.2.

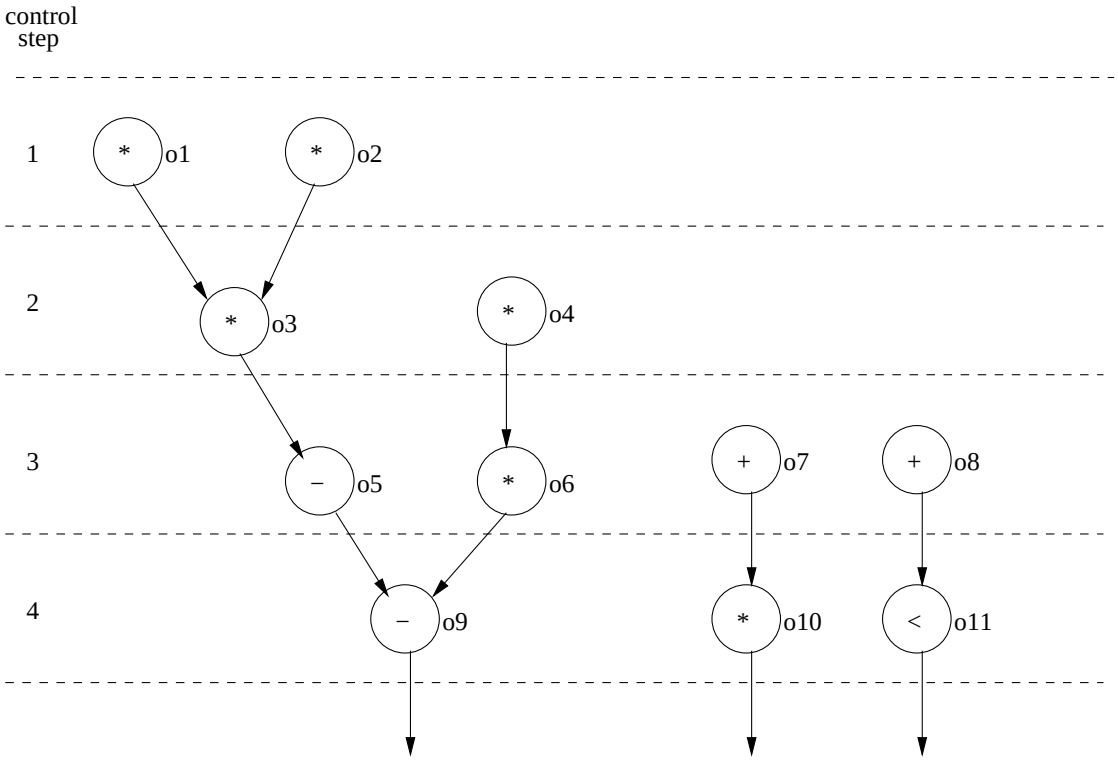


Figure 6.6.. ALAP Schedule for Differential Equation Benchmark

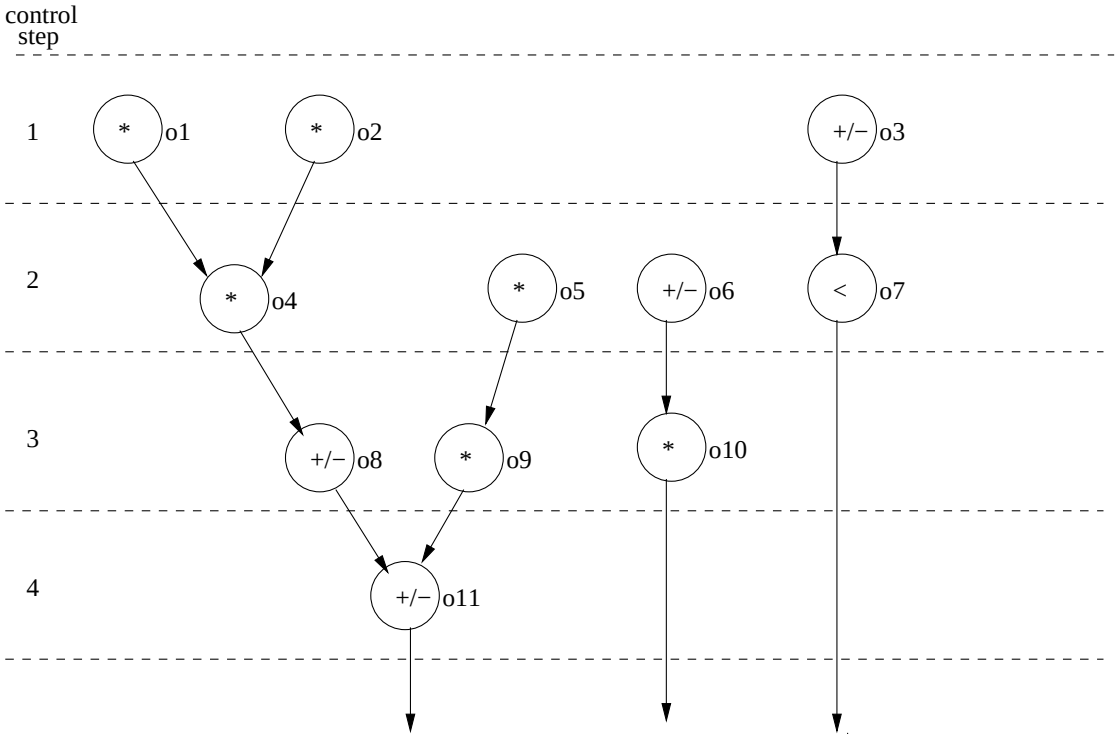


Figure 6.7.. Optimal Schedule for the Differential Equation Benchmark in 4 Control Steps

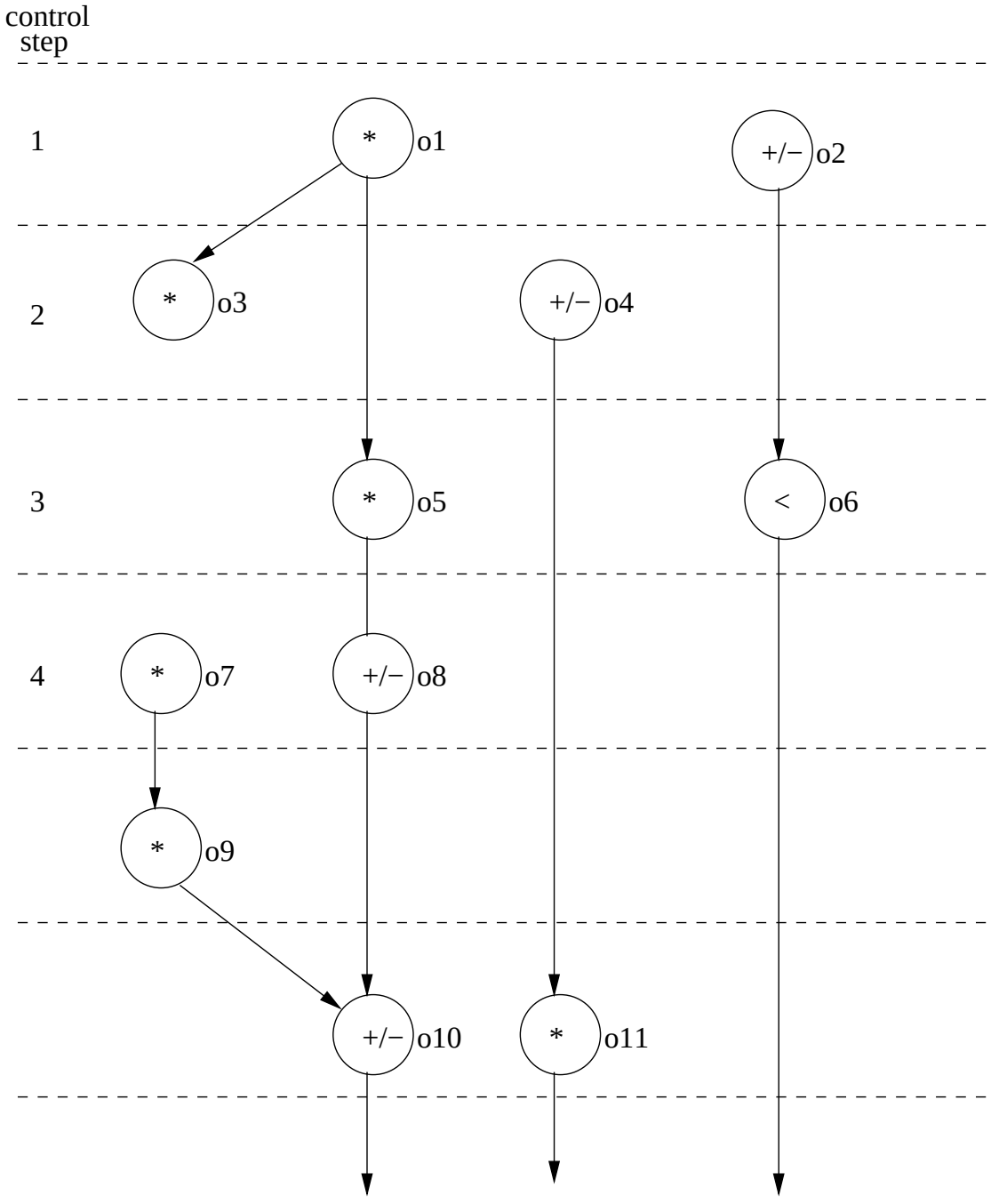


Figure 6.8.. Schedule for the Differential Equation Benchmark in 6 Control Steps

Table 6.2.. Comparison of Schedules for the Differential Equation Benchmark Circuit

Scheduling Algorithm	No. of control steps	No. of buses	No of FUs				
			(+)	(-)	(+/-)	(*)	(<)
ASAP	4	10	2	1	-	3	1
ALAP	4	8	1	1	-	2	1
HAL [85]	4	3	1	1	-	2	1
SPLICER [12]	4	6	1	1	-	2	1
CHESS (Solution 1)	4	6	-	-	1	2	1
CHESS (Solution 2)	6	4	-	-	1	1	1

HAL[85] and SPLICER[12] could schedule the diffeq benchmark with two multipliers, an adder, a subtractor and a comparator, but with CHESS, we were able to combine the adders and the subtractors by using a single ALU from the library that performs both the operations. This could be acheived because of the notion of the equivalence classes of operations used during the search for an optimized design. With six control steps, only one multiplier, one comparator and a single ALU performing addition and subtraction are required. Such a schedule was generated in 15 iterations in a CPU time of around 1 second.

6.2.2 Elliptic Filter

The elliptic filter involves more number of computations than the diffeq benchmark. The scheduled CDFG for the elliptic filter obtained by this algorithm is shown in figure 6.9.. Here, the number of control steps was constrained to 18. The outcomes with this and other time-constraints are compared to those of other systems in table 6.3.

Table 6.3.. Comparison of Schedules for the Elliptic Filter Benchmark Circuit

Scheduling Algorithm	No. of control steps	No. of buses	No of FUs	
			(+)	(*)
ASAP	17	8	4	2
ALAP	17	8	3	2
HAL [85]	19	6	2	2
FDLS [84]	17	-	3	2
CHESS (Solution 1)	17	8	3	3
CHESS (Solution 2)	18	6	2	2
CHESS (Solution 3)	19	5	2	1
CHESS (Solution 4)	28	4	1	1

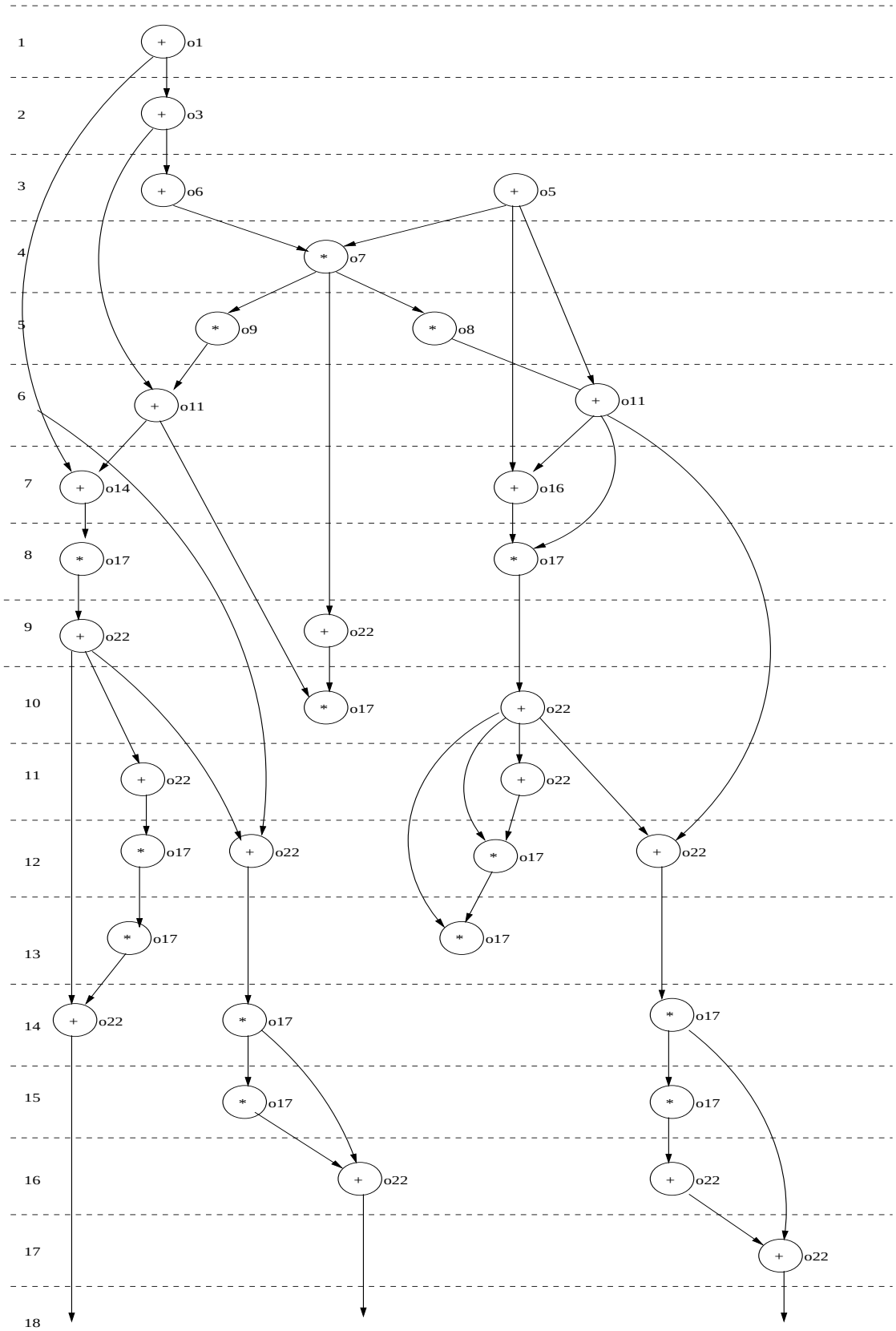


Figure 6.9.. Scheduled CDFG for the Elliptic Filter Benchmark Circuit

Several design alternatives have been generated for 17, 18, 19 and 28 control steps. Only two adders, two multipliers and six buses were required for the schedule in 18 control steps. The schedule with 17 control steps does not provide significant improvement over other approaches because of the limited solution space explored in this case, 17 being the critical path length. With increasing control steps, the solution space is extended, and after a number of trials, the best solution is obtained with 28 control steps, where only a single adder and a single multiplier are required. These solutions were produced in a CPU time of around 6 to 14 seconds.

The CDFG based scheduling algorithm, adopted from [6] was further improved for lesser execution times. The determination of mutual exclusiveness among the operations is now performed during the initial stages of extracting the CDFG from VHDL specification. This step, being removed from the scheduling algorithm has enhanced the speed of the algorithm, while adding little or no time-overhead to the CDFG extraction step. In the original work, a branch numbering approach, similar to the node coloring approach, was used for the mutual exclusion test. That approach requires the assignment of pairs of numbers to each node, depicting its branch level in the graph, followed by a simple comparision procedure determining the mutual exclusiveness between each pair of nodes. We avoid this step by performing a similar test simultaneously while extracting each node of the CDFG in the first phase of our tool.

One other aspect of this scheduling algorithm is the memory requirement. The algorithm is based on an iterative improvement scheme, and hence, requires the storage of the best set of solutions from the previous iterations, where each solution is given as a list of control step values assigned to the operations. This demands huge lists for designs involving more operations. To overcome this problem of excessive memory requirement, instead of saving a whole soluiton, we save only the modification characterizing the moves at each iteration. Such a modification could be stored as the node involved in the move, its previous control step and its new control step. Finally, the optimal solution is obtained by applying the stored sequence of changes upon the initial solution. The graph shown in figure 6.10. shows the improvement in memory utilization obtained with this optimization approach. It can seen that the memory requirement has nearly been reduced by 30% with our approach.

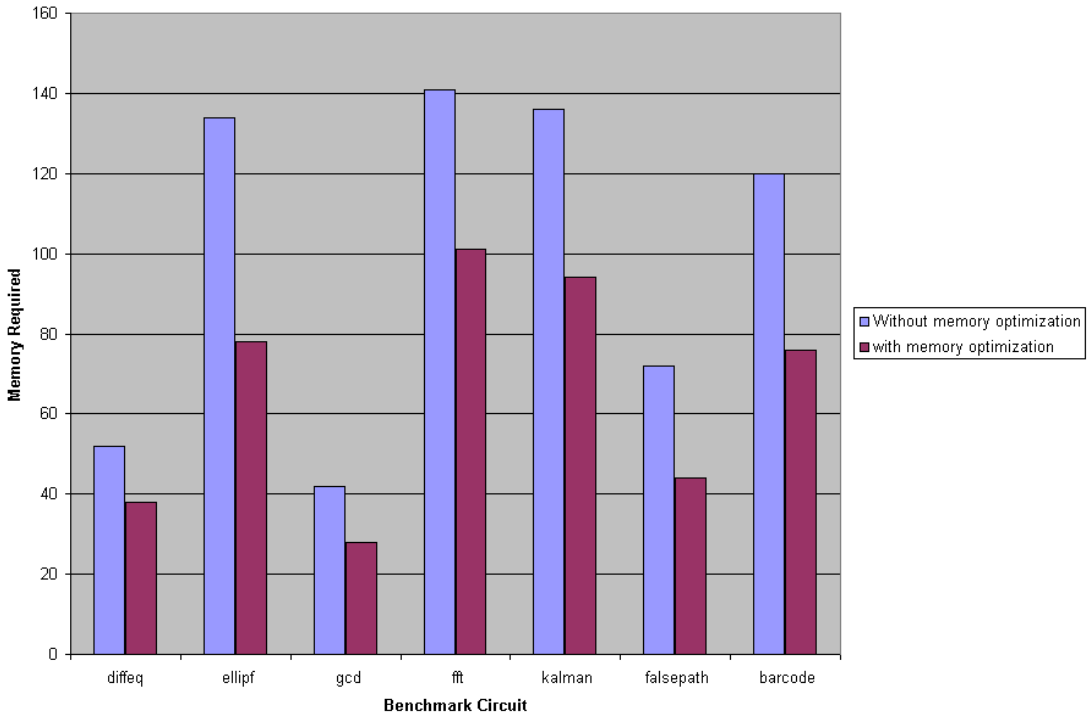


Figure 6.10.. Improvement in Memory Requirement

6.3 Binding

The algorithm described in section 5.4 was coded in C++ and tested upon the scheduled CDFGs obtained previously. The Nash equilibrium was calculated at every iteration of the algorithm using the *Gambit* tool for game theory [71]. To provide for alternative binding strategies for the operations, a library of FUs was developed using the Cadence design tool in 0.35 μ MOSIS SCN3M SCMOS technology. These cells were characterized for power and delay through simulations. Each simulation was performed using the *hspice* simulation tool upon 100,000 random input vectors that were generated using MATLAB. Some other values were borrowed from [87]. The library cells and their power and delay values are depicted in table 6.4.

The scheduled CDFGs of the benchmark circuits discussed in the previous section were subject to the game theoretic binding strategy. Before this, the total power for the S-CDFGs were calculated using a random binding strategy and a greedy approach. Later, the power values obtained through our binding strategy are compared against these, as illustrated in table 6.5. We can see that, on an

Table 6.4.. Power and Delay Values of the Library Cells

S.No.	Functional Unit	Power (mW)	Delay(ns)
1	Add1(ripplecarry)	0.260	10.71
2	Add2(carrylookahead)	0.475	9.52
3	Sub1(ripplecarry)	0.358	8.58
4	Sub2(carrylookahead)	0.549	7.02
5	Add_Sub(ripplecarry)	0.816	15.4
6	Mult1(Parallel)	1.238	43.7
7	Mult2(Wallace_tree)	5.275	21.71
8	Comp	0.187	17.0

Table 6.5.. Comparison of Binding Results

Benchmark Circuit	Random binding(mW)	Greedy binding(mW)	Our approach (mW)	%Red. over Random	%Red. over Greedy
FIR	24.005	18.300	15.005	37.49	18.005
IIR	16.290	15.304	10.290	36.83	32.76
diffeq	24.191	9.641	9.304	61.54	3.04
ellipf	34.595	28.890	25.595	26.02	11.41
fft	43.804	25.105	22.005	49.76	12.34
kalman	33.585	27.780	24.890	25.87	13.85
gcd	2.624	2.216	2.216	15.55	0

average, our tool could achieve a 18% reduction in the average power consumed over the greedy approach, for sufficiently large benchmark circuits.

The first four benchmark circuits are more data-dominated applications, while the last three involve more control structures. The improvement obtained with the control-dominated circuits is less compared to that of others because of the use of a single comparator for the control nodes. Also, it is observed that the reduction in power obtained through our approach is less for smaller circuits as compared to larger ones, because, the smaller circuits, when formulated with the game-theoretic approach, provide lesser number of strategies to be explored while optimizing the cost function, as in the case of the gcd benchmark circuit.

From these results, it can be concluded that the problem of binding for low-power can be successfully formulated using the Game-theoretic approach.

CHAPTER 7

CONCLUSIONS

We have presented a comprehensive tool for high-level synthesis with the flexibility of multiple entry and exit points. Our tool includes an automatic conversion of a behavioral VHDL description into its corresponding CDFG representation that captures all the flow information of the original description. We have demonstrated, through several examples, that the tool is quite efficient in converting various VHDL constructs into their corresponding flow-graph representations. A new technique using the Tabu search method was adapted to solve the scheduling problem for global resource optimization. The large design spaces explored through the tabu search result in producing better optimized design. A game-theoretic based power optimizing binding algorithm was proposed that takes advantage of the decentralized optimization features of non-cooperative games for obtaining an optimal assignment for each operation. The output files generated by the tool are expected to meet the requirements of various researchers across the world. With its accuracy and speed, our tool would be a significant aid to speed-up the initial steps in the VLSI design flow.

REFERENCES

[1] H. Achatz. "Extended 0/1 IP Formulation for the Scheduling Problem in High Level Synthesis". In *EURO-DAC'93*, pages 226 –231, 1993.

[2] V. Agarwal, A. Pande, and M. Mehendale. "High Level Synthesis of Multi-Precision Data Flow Graphs". In *Fourteenth Int. Conf. on VLSI Design*, pages 411 –416, 2001.

[3] A.Kumar and M. Bayoumi. "Low-power binding of function units in high-level synthesis". In *42nd Midwest Symp. on Circuits and Systems*, volume 1, pages 214 –217, 1999.

[4] M. A. Ali Shatnawi and M. Swamy. "Scheduling of DSP Data Flow Graphs onto Multiprocessors for Maximum Throughput". In *IEEE Int. Symposium on Circuits and Systems*, volume 6, pages 386–389, 1999.

[5] M. Aloqeely and C. Chen. "Sequencer-based Datapath Synthesis of Regular Iterative Algorithms". In *Design Automation Conference*, pages 155 –160, San Diego, CA, 1994.

[6] S. Amellal and B. Kaminska. "Functional Synthesis of Digital Systems with TASS". In *IEEE Transactions on Computer Aided Design of Integ. Circuits and Systems*, volume 13, pages 537–552, May 1994.

[7] R. A.V.Aho and J.D.Ullman. *"Compilers: principles, techniques and tools"*. Addison-Wesley, 1997.

[8] R. Bergamaschi. "Behavioral Network Graph Unifying the Domains of High-Level and Logic Synthesis". In *36th Design Automation Conference*, pages 213 –218, June 1999.

[9] R. Bergamaschi, S. Raje, I. Nair, and L. Trevillyan. "Control-Flow versus Data-Flow Scheduling". *IEEE Trans. on VLSI Systems*, pages 491 –496, June 1994.

[10] J. Bhasker and H. Lee. "An Optimizer for Hardware Synthesis". *IEEE Des. Test Comput.*, 7(5):20 –36, October 1990.

[11] S. Bhattacharya, S. Dey, and F. Brglez. "Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications". In *Design Automation Conference*, pages 491 –496, June 1994.

[12] B.M.Pangrle and D.D.Gajski. "Design tools for intelligent silicon compilation". In *IEEE Trans. on Computer Aided Design*, volume CDA-6, pages 1098 –1112, November 1987.

[13] R. Camposano. "Path-Based Scheduling for Synthesis". In *IEEE Trans. on Computer Aided Design of Integ. Circuits and Systems*, volume 10, pages 85 –93, January 1991.

[14] V. Chaiyakul, D. Gajski, and L. Ramachandran. ”High Level Transformation for Minimizing Syntactic Variances”. In *Design Automation Conference*, pages 413 –418, Dallas, TX, June 1993.

[15] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen. ”Optimizing Power Using Transformations”. *IEEE Trans. on CAD*, 14:12 –31, 1995.

[16] J. Chang and M. Pedram. ”Register Allocation and Binding for Low Power”. In *DAC*, 1995.

[17] J. Chang and M. Pedram. ”Module Assignment for Low Power”. In *EDAC*, 1996.

[18] S. Chaudhuri and R. Walker. ”Computing Lower Bounds on Functional Units before Scheduling”. In *IEEE Trans. on Very Large Scale Integ. of VLSI Systems*, volume 4, pages 273–279, June 1996.

[19] S. Chaudhuri, R. Walker, and J. Mitchell. ”Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem”. *IEEE Trans. on VLSI Systems*, 2(4):456 –471, 1994.

[20] K. R. Ching-Tang Chang and R. A. Walker. ”High-Level DSP Synthesis Using the COMET Design System”. In *Sixth Annual IEEE Int. ASIC Conference and Exhibit*, pages 408–411, September 1993.

[21] Y. Choi and T. Kim. ”An efficient low-power binding algorithm in high-level synthesis”. In *IEEE Int. Symp. on Circuits and Systems*, volume 4, pages IV–321 – IV–324, 2002.

[22] J. Crenshaw and M. Sarrafzadeh. ”Low-Power Driven Scheduling and Binding”. In *GLS-VLSI’98*, Lafayette, Lousiana, February 1998.

[23] J. C.T.Hwang and Y.C.Hsu. ”A formal approach to scheduling problem in data path synthesis”. In *IEEE Int. Conf. on CAD*, volume 10, pages 464–475, Apr. 1991.

[24] A. Dasgupta and R. Karri. ”High-Reliability, Low-Energy Microarchitecture Synthesis”. In *TCAD*, 1998.

[25] A. Dekkers and E. Aarts. ”Global Optimisation and Simulated Annealing”. *Math. Program*, 50:367–393, 1991.

[26] S. Devadas and A. R. Newton. ”Algorithms for Allocation in Datapath Synthesis”. In *IEEE Trans. on CAD on Integ. Cir. and Systems*, volume 8, pages 768–781, July 1989.

[27] S. Dey, A. Raghunathan, N. Jha, and K. Wakabayashi. ”Controller-based Power Management for Control-Flow Intensive Designs”. In *IEEE Trans. on Comptuer-Aided Design*, volume 18, pages 1496 –1508, October 1999.

[28] M. Dhodhi, F. Hielscher, R. Storer, and J. Bhasker. ”Datapath Synthesis Using a Problem-Space Genetic Algorithm”. In *IEEE Trans. on Computer Aided Design of Integ. Circuits and Systems*, volume 14, pages 934 –944, August 1995.

[29] J. Eijndhoven and L. Stok. ”A Data Flow Graph Exchange Standard”. In *European Design Automation Conference*, pages 193 –199, 1992.

[30] M. Elgamel and M. Bayoumi. ”On Low Power High Level Synthesis Using Genetic Algorithms”. In *9th Int. Conf. on Electronics, Circuits and Systems*, volume 2, pages 725 –728, September 2002.

[31] Y. Fann, M. Rim, and R. Jain. ”Global Scheduling for High-Level Synthesis Applications”. In *Design Automation Conference*, pages 542 –546, June 1994.

[32] C. Gebotys. ”An Optimization Approach to the Synthesis of MultiChip Architectures”. In *IEEE Trans. on VLSI Systems*, volume 2, pages 11 –20, 1994.

[33] E. Girczyc. ”Loop Winding - A Data Flow Approach to Functional Pipelining”. In *Int. Symp. on Circuits and Systems*, pages 382 –385, 1987.

[34] F. Glover. ”Tabu Search: A Tutorial”. *Interfaces*, 20(4):74–94, 1990.

[35] G. Goossens, J. Rabaey, J. Vandewalle, and H. D. Man. ”An Efficient Microcode Compiler for Application Specific DSP Processors”. *IEEE Trans. on Computer Aided Design Integ. Circuits and Systems*, 9(9):925 –937, September 1990.

[36] D. Grant and P. Denyer. ”Address Generation for Array Access Based on Modulus m Counter”. In *European Conference on Design Automation*, pages 118 –123, Paris, France, 1991.

[37] G.Sander. *”VCG: visualization of compiler graphs”*. Universitat des Saarlandes, Feb. 1995.

[38] S. Gupta and S. Katkoori. ”Force-Directed Scheduling for Dynamic Power Optimization”. In *IEEE Computer Society Annual Symposium on VLSI*, pages 68 –73, 2002.

[39] R. Hartley and A. Casavant. ”Tree-height Minimization in Pipelined Architectures”. In *Int. Conf. on Computer-Aided Design*, pages 112 –115, Santa Clara, CA, November 1989.

[40] P. Hilfinger. ”A High-Level Language and Silicon Compiler Digital Signal Processing”. In *International Symposium on Circuits and Systems*, pages 213–216, 1985.

[41] C. Hitchcock and D. Thomas. ”A Method of Automatic Data Path Synthesis”. In *Design Automation Conference*, pages 484–489, June 1983.

[42] S. Hong and T. Kim. ”Bus Optimization for Low Power Data Path Synthesis Based on Network Flow Method”. In *ICCAD*, 2000.

[43] C. Huang, Y. Chen, Y. Lin, and Y. Hsu. ”Data Path Allocation Based on Bipartite Weighted Matching”. In *Int. Conf. on Computer Aided Design*, pages 499 –504, Orlando, FL, June 1990.

[44] Institute of Electrical and Electronics Engineers, New York. *”IEEE Standard VHDL Language Reference Manual”*, ieee std. 1076-1987 edition, 1988.

[45] J.Bhaskar. *”A VHDL primer”*. Addison Wesley, 1999.

[46] J.F.Nash. ”Equilibrium Points in N-Person Games”. In *National Academy of Sciences of the United States of America*, volume 36, pages 48 –49, Jan 1950.

[47] G. D. Jong. ”Data Flow Graph: System Specification with the most Unrestricted Semantics”. In *European Design Automation Conference*, pages 401 –405, 1991.

[48] J.Teich, T. Blickle, and L.Thiele. "An Evolutionary Approach to System-Synthesis". In *First Online Workshop on Soft Computing*, 1996.

[49] J.W.Friedman. *"Game theory with applications to economics"*. Oxford University Press, 1986.

[50] T. Kawaguchi and T. Todaka. "Operation Scheduling by Annealed Neural Networks". In *IEICE Trans. Fund. Electr. Commun. Comput. Sci.*, pages 656 –663, June 1995.

[51] J. Kim, S. Park, Y. Seo, and D. Kim. "Pattern Generation for Verification of VHDL Behavioral-Level Design". In *The First IEEE Asia Pacific Conference on ASICs*, pages 332 –335, August 1999.

[52] T. Kim and C. Liu. "A New Approach to the Multiport Memory Allocation Problem in Datapath Synthesis". *VLSI Integration*, 19(3):133 –160, 1995.

[53] T. Kim, N. Yonezawa, J. Liu, and C. Liu. "A Scheduling Algorithm for Conditional Resource Sharing - A Hierarchical Reduction Approach". In *IEEE Trans. on Computer-Aided Design of Integ. Circuits and Systems*, volume 13, pages 425 –437, April 1994.

[54] P. Kollig and B. Al-Hashimi. "Simultaneous scheduling, allocation and binding in high level synthesis". In *Electronic Letters*, volume 33, pages 1516–1518. August 1997.

[55] D. Kolson, A. Nicolau, and N. Dutt. "Integrating Program Transformations in the Memory-based Syntehsis of Image and Video Algorithms". In *Int. Conf. on Computer-Aided Design*, pages 27 –30, San Jose, CA, 1994.

[56] T. Krol, J. Meerbergen, C. Niessen, W. Smits, and J. Huisken. "The Sprite Input Language: An Intermediate Format for High-Level Synthesis". In *European Design Automation Conference*, pages 186 –192, 1991.

[57] K.S.Vallerio and N.K.Jha. "Task graph extraction for embedded system synthesis".

[58] D. Ku and G. Micheli. "Relative Scheduling Under Timing Constraints". *IEEE Trans. on Computer-Aided Design*, 11:696 –718, June 1992.

[59] K. Kucukcakar and A. Parker. "Data Path Tradeoff using MABAL". In *Design Automation Conference*, pages 511 –516, Orlando, FL, June 1990.

[60] F. Kurdahi and A. Parker. "REAL: A Prgoram for Register Allocation". In *Design Automation Conference*, pages 210–215, Miami Beach, FL, June 1987.

[61] G. Lakshminarayana, K. Khouri, and N. Jha. "Wavesched: A Novel Scheduling Technique for Control-Flow Intensive Behavioral Descriptions". In *IEEE Trans. on Computer-Aided Design*, pages 244 –250, 1997.

[62] G. Lakshminarayana, A. Raghunathan, N. Jha, and S. Dey. "Transforming Control-FLow Intensive Designs to Facilitate Power Management". In *Int. Conf. on Computer-Aided Design*, pages 657 –664, November 1998.

[63] B. Landwehr, P. Marwedel, and R. Domer. "OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming". In *EURO-DAC'94*, pages 90 –95, 1994.

[64] M. Langevin and E.Cerny. ”A Recursive Technique for Computing Lower-Bound Performance of Schedules”. In *ACM Transactions on Design Automation of Electronic Systems*, volume 1, pages 443–456, October 1996.

[65] H. Lee and S. Hwang. ”A Scheduling Algorithm for Multiport Memory Minimization in Datapath Synthesis”. In *The Asia and South-Pacific Design Automation Conference*, pages 93 –100, 1995.

[66] J. Lee, Y.Hsu, and Y. Lin. ”A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis”. In *Int. Conf. on Computer-Aided Design*, pages 20–23, 1989.

[67] J. S. Lis and D. D. Gajski. ”Synthesis from VHDL”. In *IEEE Int. Conf. on Computer Design*, pages 378–381, 1988.

[68] T. Ly, D. Knapp, R. Miller, and D. MacMillen. ”Scheduling Using Behavioral Templates”. In *Design Automation Conference*, pages 101 – 106, 1995.

[69] T. Ly and J. Mowchenko. ”Applying Simulated Evolution to High-Level Synthesis”. In *IEEE Trans. on Computer Aided Design of Integ. Circuits and Systems*, volume 12, pages 389 –409, March 1993.

[70] S. D. M. Potkonjak and R. Roy. ”Synthesis-for-testability using tranformations”. In *Asia and South-Pacific Design Automation Conference*, pages 485–490, 1995.

[71] R. McKelvey, A. McLennan, and T. Turocy. *”Gambit: Software Tools for Game Theory”*. California Inst. of Tech. and Univ. of Minnesota and Texas A&M Univ., September 2002.

[72] G. Mekenkamp, P. Middelhoek, E. Molenkamp, J. Hofstede, and T. Krol. ”A Syntax based VHDL to CDFG Translation Model for High-Level Synthesis”. In *VHDL Int. Users Forum*, Santa Clara, March 1996.

[73] G. D. Micheli and D. C. Ku. ”HERCULES - A System for High-Level Synthesis”. In *Design Automation Conference*, pages 483–488, 1988.

[74] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar. ”Scheduling Techniques to Enable Power Management”. In *DAC*, 1996.

[75] A. K. Murugavel and N. Ranganathan. ”A Game-Theoretic Approach for Power Optimization during Behavioral Synthesis”. *16th Int. Conf. on VLSI Design*, pages 452 –458, January 2003.

[76] J. V. Neumann. ”Zur Theorie der Gesellschaftsspiele”, 1928.

[77] A. Nicolau and R. Potasman. ”Incremental Tree Height Reduction for High Level Synthesis”. In *Design Automation Conference*, pages 770 –774, San Fransisco, CA, June 1991.

[78] T. Nijhar and A. Brown. ”Appliation of Source Code Transformations in a High Level Synthesis Environment”. *Design Automation Group Research Journal*, 1995.

[79] A. Orailoglu and D. Gajski. ”Coactive Scheduling and Checkpoint Determination During High Level Synthesis of Slef-Recovering Microarchitectures”. In *IEEE Trans. on Very Large Scale Integration Systems*, volume 2, pages 304 –311, September 1994.

[80] C. Papadimitriou. ”Algorithms, Games and the Internet”. In *ACM Symp. on Theory of Computing*, pages 749 –753, 2001.

[81] I. Park and C. Kyung. ”Fast and Near Optimal Scheduling in Automatci Data Path Synthesis”. In *Design Automation Conference*, pages 680 –685, San Francisco, CA, June 1991.

[82] N. Park and A. Parker. ”Sehwa: A Software Package for Synthesis of Pipelined Data Path from Behavioral Specification”. In *IEEE Trans. on COmputer-Aided Design of Integ. Circuits and Systems*, volume 7, pages 356 –370, March 1988.

[83] S. Park. ”VDT: VHDL developer’s toolkit”. http://poppy.snu.ac.kr/vdt, 2002.

[84] P. G. Paulin and J. P. Knight. ”Force Directed Scheduling for the Behavioral Synthesis of ASIC’s”. In *IEEE Trans. Computer Aided Design*, volume 8, pages 661–679, June 1989.

[85] P. G. Paulin and J. P. Knight. ”Algorithms for High-Level Synthesis”. In *IEEE Design and Test of Compters*, volume 6, pages 18–31, December 1999.

[86] M. Potkonjak and M. Srivastava. ”Rephasing: A Transformation Technique for the Manipulation of Timing Constraints”. In *Design Automation Conference*, pages 107 –112, San Francisco, CA, 1995.

[87] V. Raghunathan, S. Ravi, A. Raghunathan, and G. Lakshminarayana. ”Transient Power Management through High-Level Synthesis”. In *IEEE Int. Conf. on Computer Aided Design*, pages 545 –552, 2001.

[88] S. Raje and M. Sarrafzadeh. ”GEM: A Geometric Algorithm for Scheduling”. In *IEEE Int. Symposium on Circuits and Systems*, volume 3, pages 1991–1994, May 1993.

[89] M. Rim and R. Jain. ”Lower-bound performance estimation for the high level synthesis scheduling problem”. In *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, volume 13, pages 451–458, April 1994.

[90] W. Rosenstiel. ”Optimizations in High Level Synthesis”, 1986.

[91] M. Rosien, G. Smit, and T. Krol. ”Generating a CDFG from CC++ Code”. In *3rd PROGRESS Workshop on Embedded Systems*, pages 200 –202. STW Technology Foundation, October 2002.

[92] S.Amellal and B.Kaminska. ”Scheduling of a control and data flow graph”. In *IEEE Int. Symp. on Circuits and Systems*, volume 3, pages 1666–1669. May 1993.

[93] S.Amellal and B.Kaminska. ”A CDFG model for synthesis from VHDL”. Technical report, Ecole Polytechnique de Montreal, August 1997.

[94] O. Sentieys, E. Martin, and J. Philippe. ”VLSI Architectural Synthesis for an Acoustic Echo Cancellation Application”. In *Workshop on VLSI Signal Processing*, pages 84 –92, October 1993.

[95] N. S.Govindarajan and R.Vemuri. ”Dependecny analysis and operation graph generation for high-level synthesis from behavioral VHDL”.

[96] A. Sharma and R. Jain. ”InSyn: Integrated Scheduling for DSP Applications”. In *IEEE Trans. on Signal Process.*, volume 43, pages 1966 –1977, August 1995.

[97] H. Shin and N. Woo. ”A Cost Function based Optimization Technique for Scheduling in Data Path Synthesis”. In *Int. Conf. on Computer Aided Design*, pages 424–427, 1989.

[98] W.-T. Shiue, J. Denison, and A. Horak. ”Low Power Binding using Linear Programming”. In *43rd IEEE Midwest Symp. on Circuits and Systems*, volume 2, pages 980 –983, 2000.

[99] W. T. Shiue, J. Denison, and A. Horak. ”Low Power Binding using Linear Programming”. In *IEEE Midwest Symp. on Circuits and Systems*, volume 2, pages 980 –983, 2000.

[100] A. Sllame and V. Drabek. ”An Efficient List-Based Scheduling Algorithm for High-Level Synthesis”. In *Euromicro Symp. on Digital System Design*, pages 316 –323, September 2002.

[101] V. Srikantam, N. Ranganathan, and S. Srinivasan. ”CREAM: combined register and module assignment with floor-planning for low power data-path synthesis”. In *IEEE Int. Conf. on VLSI Design*, pages 223 –228, 2000.

[102] N. Theypayasuwan, H. Tang, and A. Doboli. ”An Exploration-Based Binding and Scheduling Technique for Synthesis of Digital Blocks for Mixed-Signal Applications”. In *2003 Int. Symp. on Circuits and Systems*, volume 5, pages 629 –632, May 2003.

[103] F. Tsay and Y. Hsu. ”Data Path Construction and Refinement”. In *Int. Conf. on Computer-Aided Design*, pages 308 –311, Santa Clara, CA, November 1990.

[104] C. Tseng and D. Siewiorek. Automatic syntheis of data path on digital systems. In *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, volume 5, pages 379–395, July 1986.

[105] M. Unaltuna and V. Pitchumani. ”ANSA: A New Neural Net based Scheduling Algorithm for High-Level Synthesis”. In *IEEE Symp. on Circuits and Systems*, volume 1, pages 385 –388, 1995.

[106] K. Wakabayashi and T. Yoshimura. ”A Resource Sharing Control Synthesis Method for Conditional Branches”. In *Int. Conf. on Computer-Aided Design*, pages 62 –65, Santa Clara, CA, November 1989.

[107] R. Walker and D. Thomas. ”Behavioral Transformations for Algorithmic Level IC Design”. *IEEE Trans. on Computer Aided Design of Integ. Circuits and Systems*, 8(10):1115 –1128, October 1989.

[108] W. Wang, T. Tan, J. Luo, Y. Fei, L. Shang, K. Vallerio, L. Zhong, A. Raghunathan, and N. Jha. ”A Comprehensive High-Level Synthesis System for Control-Flow Intensive Behaviors”. In *GLVLSI’03*, 2003.

[109] X. Wang and S. Grainger. ”The Reduction of the Number of Equations in the ILP Formulation for the Scheduling Problem in High-Level Synthesis”. In *The Second International Conference on Concurrent Engineering and Electronic Design Automation*, pages 483 –487, 1994.

[110] T. Wilson, N. Mukherjee, M. Garg, and D. Benerji. ”An ILP Solution for Optimum Scheduling, Module and Register Allocation, and Operation Binding in Datapaht Synthesis”. *VLSI Design*, 3(1):21 –36, 1995.

[111] W. Wolf, A. Takach, C. Huang, and R. Mano. ”The Princeton University Behavioral Synthesis System”. In *Design Automation Conference*, pages 182 –187, June 1992.

[112] L. Zhong, J. Luo, Y. Fei, and N. Jha. ”Register Binding based Power Management for High-Level Synthesis of Control Flow Intensive Behaviors”. In *IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, 2002.

[113] J. Zhu and D. D. Gajski. ”Soft Scheduling in High Level Synthesis”. In *36th Design Automation Conference*, pages 219–224, 1999.

[114] G. Zimmermann. Mds: The mimola design method. *Journal of Digital Systems*, 4(3):337–369, 1980.