Applying integer programming in high level synthesis scheduling
study on where and how IP can be used in the process of scheduling Data Flow Graphs

van Leeuwen, J.C.

Award date:
1995

Eindhoven University of Technology
Department of Electrical Engineering
Design Automation Section (ES)

Applying Integer Programming in High Level Synthesis Scheduling

Study on where and how IP can be used
in the process of scheduling Data Flow Graphs

By J.C. van Leeuwen

Master Thesis

performed: October 1993 - May 1995
supervised by ir. A.H. Timmer

Abstract

A silicon compiler is a tool for the automated design of integrated circuits (ICs). It generates the layout of an IC starting from an algorithmic behavioural description provided by the designer.
High Level Synthesis is the part from a silicon compiler that generates an IC description in terms of modules, registers, interconnect and a controller description.
Scheduling is a major part of high level synthesis that decides in which cycle step operations start and finish their execution while holding designer constraints as the number of resources or the maximum execution time.
Scheduling is in general a NP-complete problem. Therefore all solving methods that find the optimal scheduling solution are not guaranteed to find the optimum within acceptable computation time.
One of the optimal methods is the Integer Linear Programming (**IP**) approach to the scheduling problem. In this approach the scheduling problem is described as a set of linear inequalities. A linear (object) function has to be optimised keeping linear constraints. All variables must have an integer value in the solution.
Recent research showed that applying the node packing (**NP**) model to the **IP** scheduling problem can lead to shorter run times.
This **IP:NP** model does not lead to all integer solutions in general cases. A branch-and-bound process is necessary to come to an integer solution. Especially in cases of detecting infeasibilities in problems this is a very inefficient process. So it is interesting to extend the **IP:NP** model as far as possible to come to an all integer solution.
This thesis shows the process to handle the **IP:NP** model as efficient as possible. It also describes some possible extensions to the model. In general the number of possible additions to complete the model will grow exponential with the size of the problem.
This thesis shows that the application of the **IP:NP** model will be of limited use to the scheduling problem due to its incompleteness or the amount of extra constraints.
Still the extended **IP:NP** model can be applied in certain steps of the scheduling process. Mostly as support tool in special cases for more run time efficient methods.

Contents

Chapter 1

Introduction

At the Design Automation Section of the Eindhoven University of Technology software tools are being developed for the automated design of Integrated Circuits (ICs). A Silicon Compiler is one of these tools. It generates the layout of a chip starting from an algorithmic behavioural description, provided by the designer. The synthesis process of the silicon compiler consists of three main parts (see figure 1.1).

- **High level synthesis** translates the behavioural description into a data path (a network of modules (like adders and multipliers), registers and interconnect) and a controller description. This is a translation with given goals and constraints (like cycle time, area, power consumption etc.).
- **Logic synthesis** translates the controller description into a gate network and can optimise the data path.
- **Layout synthesis** generates the integrated circuit layout from the gate network supplied after the previous steps.

See [Mich92] for a more thorough approach to these topics.

The assignment that this master thesis is based on is a small part of high level synthesis. Chapter 2 consists of a brief introduction to high level synthesis and a description of the high level synthesis development environment NEAT. The research for my master degree assignment is performed in that environment.

The subject of this report is scheduling (described in chapter 3). Scheduling is a main part of high level synthesis. A scheduler assigns cycle steps to each operation of the behavioural description without violating given constraints: the precedence relation between 2 operations is the order of their execution. Resource constraints

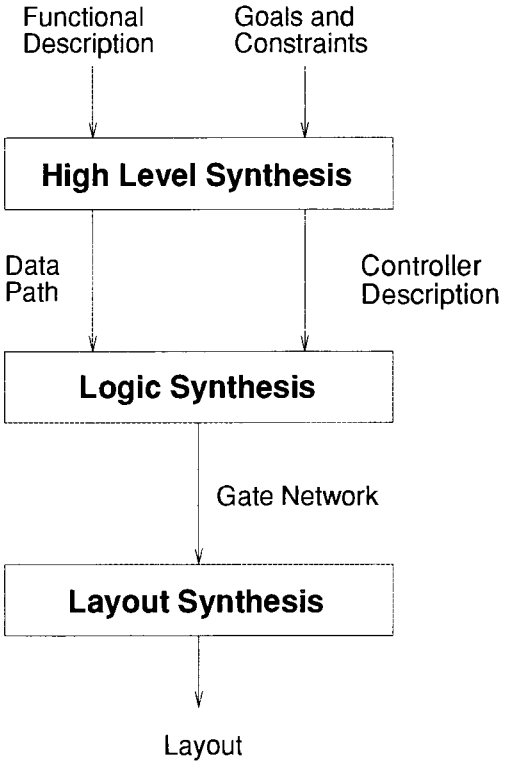

Figure 1.1: Silicon Compiler Overview

restrict the number of modules which are used to implement all operations. Time constraints restrict the number of cycle steps in which all operations must be executed.

My assignment concerned the research towards the possible use of Integer Linear Programming (**IP**) in the high level synthesis scheduling process. This **IP** scheduling method states the scheduling problem and its constraints as a set of linear inequalities. The solution of the set of linear inequalities is also the solution of the scheduling problem. All operations are assigned to specific cycle steps at the time the **IP**-solver is finished.
The research aims to check the claims that are made about the efficiency of **IP** in the scheduling process. These efficiency claims are made about run time efficiency (i.e. the computing time it takes to get a schedule) and the way the scheduling problem is modeled in this solution method. The **IP** solution to a schedule problem is optimal but it can take unacceptable time to compute this solution. Therefore it is tried to model the **IP** scheduling problem in a way that is better suited to certain properties of scheduling. The model that is used here is the Node Packing (**NP**) model. A sequence of processing steps is suggested to make most use of the **NP**

model, to try to keep computation time of the **IP** schedule acceptable, described in chapter 6. Apart from research on the already existing method of **IP** scheduling this thesis contains possible enhancements to that model, they are described in chapter 7. In the implementation of the **IP** scheduler only basic operations and modules are implemented. The implementation is described and evaluated in chapter 8.
The goal of this thesis is to evaluate the practical future use of **IP** in the high level synthesis scheduling traject. Conclusions are drawn after the research that has been done on that subject. Some possibilities have been left open for further research, these possibilities are stated in section 8.4.

Chapter 2

High Level Synthesis

High level synthesis is the transition between the behavioural domain and the structural domain of a design. The behavioural domain gives a functional description of the behaviour of the IC. The structural domain gives a description in terms of functional modules. There is also the physical domain that describes the design of the IC in terms of its hardware components. High Level is an abstraction level of a domain. Each domain is divided in levels of decreasing abstraction: System-, High-, Logic- and Circuit level. A lower abstraction level gives a more detailed IC description in the same domain. High level synthesis is the process where an algorithmic description of an IC is translated into a digital network which performs the functionality of the algorithmic specification. In figure 2.1 this process is put in the context of the complete process of automated IC creation. The domains form the axes of the Y-chart and the abstraction levels are the nested shells. The bold arrow shows the high level synthesis translation.

A high-level synthesis system should be able to accept a high-level specification language and some user constraints. In high level synthesis the algorithmic behaviour is represented by a data flow graph (DFG) (see fig 2.2)

The specification is translated using some module library and must regard some specific objectives, like minimising area or power dissipation. An introduction to high level synthesis can be found in [Arts91] and [McFa90].

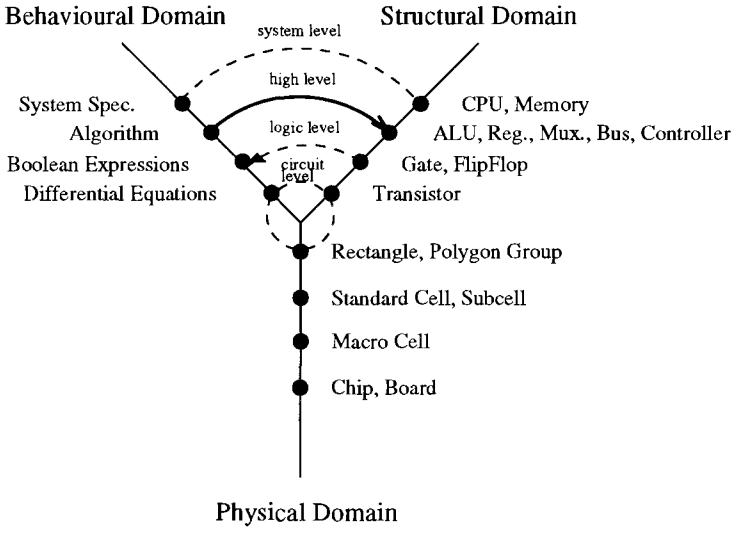

Figure 2.1: Y-chart Transitions of High Level Synthesis

2.1 The NEAT System

In high level synthesis several steps are performed subsequently, to achieve the desired result. At the Design Automation Section (ES) of the Eindhoven University of Technology these steps are performed on the NEAT database using the available functions from the NEAT system. The acronym NEAT is derived from New Eindhoven Architectural synthesis Toolbox and is the successor of the EASY system as described in [Stok91]. In the first step of high level synthesis, the behavioural description of the algorithm which has to be implemented is translated into an ASCIS DFG [Eijn91]. This graph combined with the goals and constraints, can be processed by all parts of the NEAT system.

One of the inputs for the high level synthesis process is a set of module-types called a "library". The information which module types are available and what operations they can support is also stored in a database file of the NEAT system. At an abstract level the ASD-file, in which the contents of the NEAT database are stored, contains a collection of graphs belonging to three distinct categories:

- Data Flow Graphs
- Network Graphs
- Control Graphs

```
process example(b, c)
  in boolean b[8];
  out boolean c[8];
  {
    a = 1;
    while (a < 3)
    do {
      a++;
      b = b + a;
    }
    c = b;
  }
```

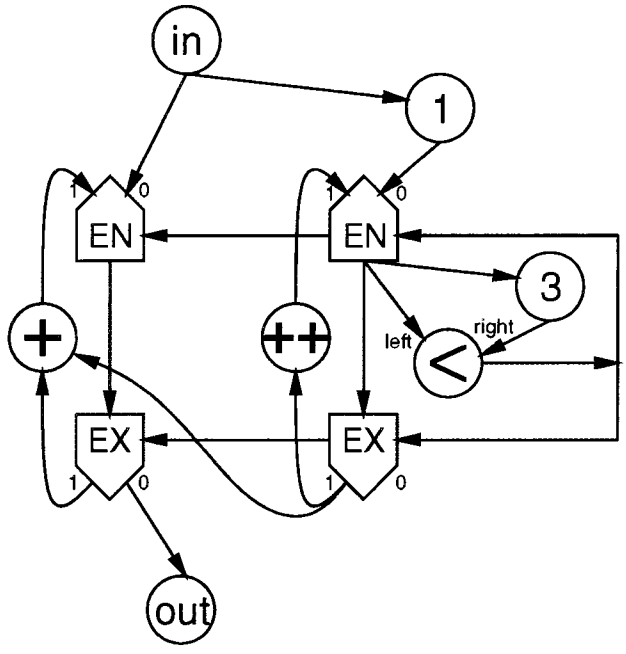

Figure 2.2: An example of an algorithm with corresponding DFG

The network and control graphs will both be constructed and added to the NEAT database during the high level synthesis process. The network graph contains the data path consisting of modules that implement the operations in the DFG, registers to store intermediate data values and the interconnection units. The control graph contains the controller description and in this description the control flow is specified.

The NEAT system features a modular mechanism that makes it possible to split the high level synthesis problem in several manageable parts. Those parts work in a certain order (depending on the designer demands) on the same database and use the same predefined functions. Each part can be constructed and maintained independent from the others. (See figure 2.3).

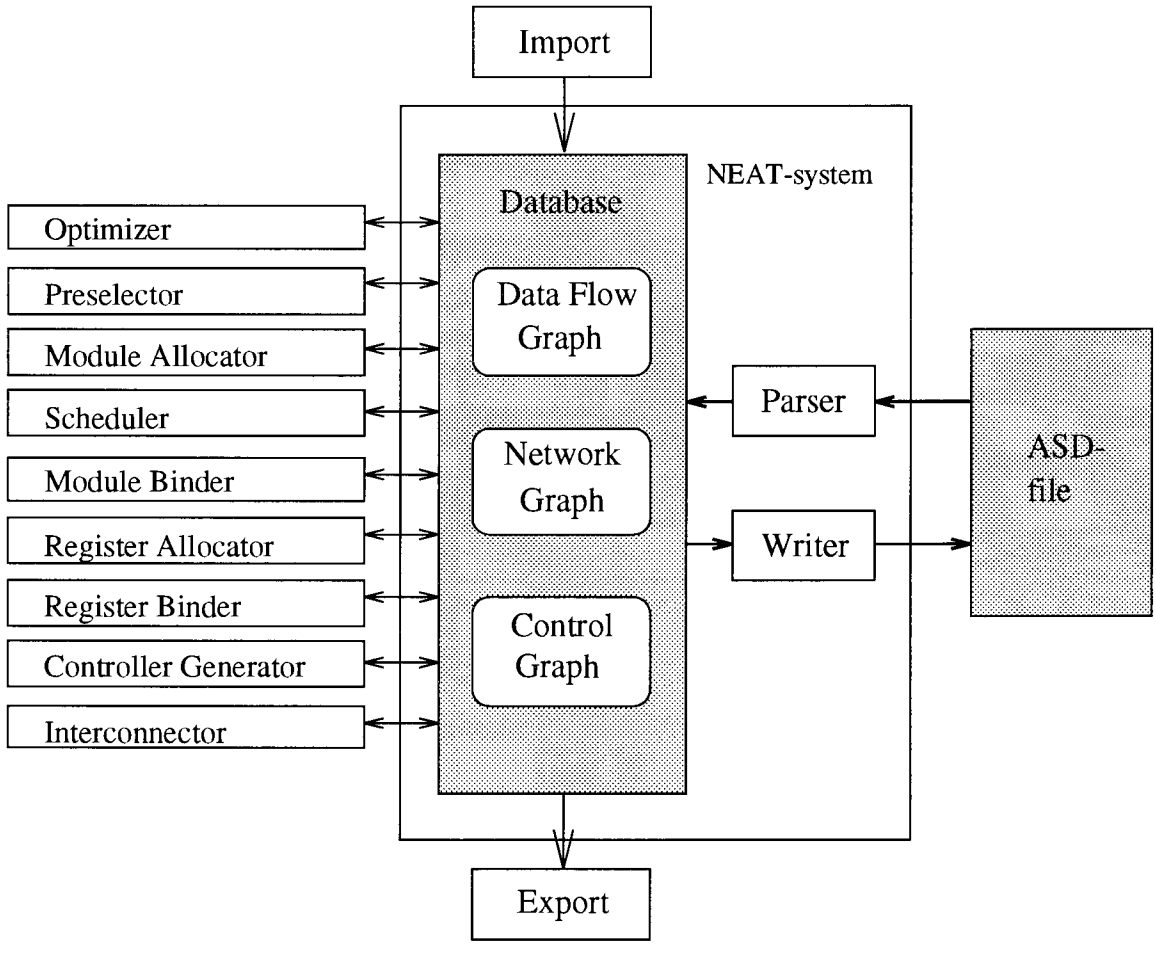

Figure 2.3: The NEAT System

A brief description of the modules:

- optimiser: Some optimisations can be applied to the initially created data flow graph like dead code elimination, constant propagation, common subexpression elimination, in-line expansion of procedures, tree height reduction, loop optimisation and memory access optimisations.
- preselector: From a certain library of module types, the preselector tries to determine a subset of module types that is optimal for the current DFG.
- module allocator: The exact number of modules of each type that are needed is determined.
- scheduler: The scheduler assigns a range of cycle steps to every operation that occurs in the DFG. A cycle step is the fundamental sequencing unit in a synchronous system.

- module binder: Assigns the operations in the DFG to modules.
- register allocator: The register allocator allocates the necessary registers to hold intermediate data values.
- register binder: Assigns necessary registers to hold intermediate data values.
- controller generator: Modules like registers, multiplexers and ALUs that can perform a variety of actions are managed by the controller to perform the right actions at the right moment.
- interconnector: The interconnections between all the resulting modules are allocated by the interconnector. It also binds the data transfers and creates the multiplexers that switch the right data values to the right nodes.

For more information about the NEAT system see [NEAT92].

Chapter 3

Scheduling

The process that decides when operations of the DFG start and finish their execution is called scheduling. Scheduling is one of the major parts in high level synthesis. During scheduling a trade-off can be made between execution time and module area. If an integrated circuit needs to perform fast (perform operations parallel) it needs a large module area. If the IC can perform relatively slow (sequential performance) it can be designed with much smaller module area. The relation between time and area can be visualised by a so-called design space. The general shape of a design space is fig 3.1. The bound is formed by the time-area curve. Every solution above that curve is valid. A solution on the curve is desired.

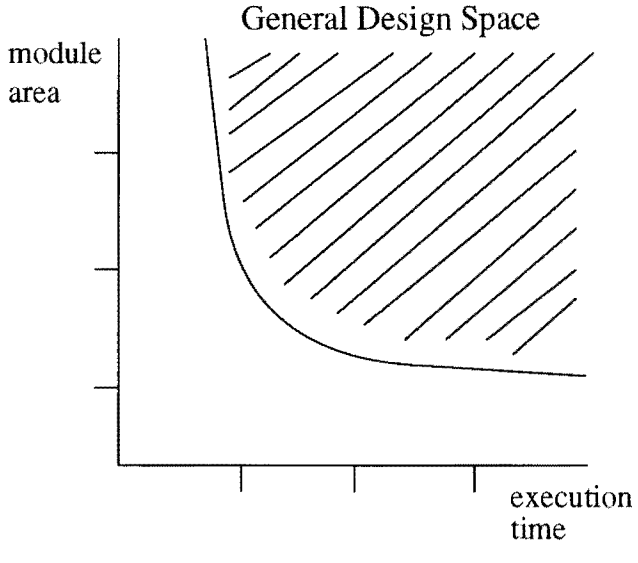

Figure 3.1: General Design Space

For fast schedules more resources will be needed to implement these schedules. The goal of scheduling is to find the point in the design space that best meets the requirements of the designer.

The scheduling problem is in general a Non Deterministic Polynomial complete (NP-complete) problem (see [Gare79]) (i.e. the computing time solving the problem lengthens exponentially with the growing of the amount of parameters in the computation). Therefore all solving methods that find the optimal solution are not guaranteed to find that optimum within acceptable computation time. Simple and special examples will perform excellent but in general there will be an exponential rise in computation time when the number of variables increases.

Therefore two approaches for solving the scheduling problem are possible. The first approach is to solve the problem with a method that finds the optimal solution. The disadvantage is inherent to the NP-complete nature of scheduling: finding the optimal solution is in general not run time efficient.

The second approach is to apply a heuristic solution method. The solutions found with this method are however not guaranteed to be optimal. The process of finding (near) optimal solutions using heuristics is in most cases much faster than using the optimal solution methods. Compared to schedulers using the optimal solution method, heuristic methods find a solution after computation time that is usually acceptable in practice.

3.1 Scheduling Constraints

The designer can restrict the schedules which he considers to be valid. These restrictions are often called schedule constraints. The constraints limit the search space of the scheduler.

Some constraints that are used are:

Precedence constraints:
A data flow node can start its execution after all its predecessors have finished their execution. The partial execution order is reflected by the precedence relations of the DFG.

Time constraints:
The DFG has to be scheduled within a limited number of cycles, denoted by (Tmax).

This means for all operations $v \in V : end(v) \leq T_{max}$, with $end(v)$ the time an operation has finished its execution.

Resource constraints:
The number of resources that may be used are determined beforehand. Often these resources are modules, but one may also constrain the number of registers, multiplexers and interconnect.

Other constraints:
Other constraints that can be used are throughput, power dissipation or combinations of several constraints. Combinations can lead to the situation that no valid schedule exists.

The task of the scheduler can be described as to find the optimal schedule within the design space, with respect to the goals and constraints given. The types of schedule problems that schedulers are mostly based upon are:

Precedence constrained scheduling:
These schedule algorithms try to minimise the execution time while preserving precedence constraints. Algorithms to solve precedence constrained scheduling are known as ASAP or ALAP schedulers.

Resource constrained scheduling:
The schedule algorithm tries to minimise the execution time while preserving both precedence and resource constraints.

Time constrained scheduling:
The schedule algorithm tries to minimise the area while preserving both precedence and time constraints.

Feasible constrained scheduling
The schedule algorithm tries to find a schedule solution that fits a combination of constraints (e.g. time and resource constraints) next to the precedence constraints.

3.2 ASAP and ALAP scheduling:

These methods are precedence constrained. As Soon As Possible (ASAP) and As Late As Possible (ALAP) are often used because of their $O(n + e)$ complexity (with n the number of operations and e the number of edges in the DFG) and

the information they give about a DFG. Both the ASAP and ALAP scheduler do not take any further constraints into account. They both generate a schedule with minimum execution time without looking at efficient use of hardware.

Let pred(v) = the set of predecessors of operation v, succ(v) = the set of successors. $d(v_i)$ = delay of the fastest module type available that can execute operation v_i. For each $v \in V$ the ASAP schedule $\varphi(v)$ can be defined recursively as:

$asap : V \rightarrow \mathbb{R} :$

$$asap(v) = \begin{cases} 0 & \text{if pred(v)} = \emptyset \\ max_{v_i \in pred(v)} asap(v_i) + d(v_i) & \text{otherwise} \end{cases}$$

$$\varphi(v) = [asap(v), asap(v) + d(v)]$$

For each $v \in V$ the ALAP schedule $\varphi(v)$ can be defined recursively as:

$alap : V \rightarrow \mathbb{R} :$

$$alap(v) = \begin{cases} t_{max} & \text{if succ(v)} = \emptyset \\ min_{v_i \in succ(v)} alap(v_i) + d(v_i) & \text{otherwise} \end{cases}$$

$$\varphi(v) = [alap(v) - d(v), alap(v)]$$

The ASAP and ALAP schedulers are usually run as an initiating procedure prior to some other scheduling method that does take more constraints into account. The resulting [ASAP,ALAP] interval gives for each operation a lower bound (ASAP value) and an upper bound (ALAP value). The operation has to be scheduled within this interval. Other schedule methods as in section 5 and section 5.2 make use of this initiating limitation.

Chapter 4

Integer Linear Programming

A linear programming (**LP**) problem is an optimising problem where a linear (object) function has to be optimised keeping a number of linear constraints. In practice many optimising problems demand integer values for their variables. With that extra constraint on the **LP** problem, the problem has become an Integer Linear Programming (**IP**) problem. An integer linear programming problem has the following formulation:

$$\begin{aligned} MAX\ & \{w = \underline{c}\underline{x}\} \\ & A\underline{x} = \underline{b} \\ & \underline{x} > \underline{0} \\ & x_j\ integer, \qquad j \in I \subset \{1,2,3,\ldots,n\} \end{aligned} \tag{4.1}$$

In (4.1) is $\underline{x} \in R^n, \underline{c} \in R^n, \underline{b} \in R^m, A$ is a $m \times n$ matrix and I is the index set of integer variables.

There are two obvious ways for solving an **IP** problem: the first is to search the optimum by checking all possible combinations. This method is practically not possible because the number of combinations grows exponentially with the increasing number of variables. The second way is to solve the **IP** problem by first dropping the integer demands called relaxation (i.e. it becomes a continuous **LP** problem). For that problem the optimal solution can be found in polynomial time, but it is in general cases not an all integer solution. **IP** solvers based on the second approach often use the Branch-and-Bound method to come to an all integer solution. Branching divides a continuous **LP** problem (with a non integer solution) into 2 continuous **LP** subproblems by adding an inequality that restricts

a non integer variable to be smaller or equal than its floor value or an inequality restricting the variable to be greater or equal than the ceiling value (i.e. these are the closest integer values). Then these subproblems are solved. This process continues until the subproblems give an integer solution or find no solution. Bounds are determined from subproblems to find an optimal solution and to stop the creation of subproblems if the solution is not existing or already below a temporary optimum. The Branch-and-Bound method searches all possible integer solutions and the efficiency of this search process is enlarged by the use of bounds. Theory of **IP** problems can be found in [Garf72], [Hend91], [Kovά80] and [Taha75].

4.1 The Node Packing Model

The Node Packing (**NP** model is introduced because structural properties of data flow graph scheduling can be profitable in this model to obtain more run time efficient solving of the problem.

According to [Nemh88] the general **NP** problem is defined as follows:
In a given graph $G = (N, E)$ with N the set of nodes and E the set of edges. A node packing or independent set in G is a subset of nodes such that no pair in the subset is joined by an edge. The node packing problem is to find a maximum packing.

In other words: find a maximum set $N' \subset N$ such that for each edge $e \in E$ at most one end node is in N'.
The node packing problem is NP-hard for general graphs, see [Gare79].

The most elementary node packing description is that from all edges of G one node is in the packing.

$$Variables \quad x_n \begin{cases} 1 & \text{the node } n \text{ is chosen in the packing} \\ 0 & \text{otherwise} \end{cases}$$

IP Edge Formulation of Node Packing:

$$MAX \sum_{n \in N} x_n \quad such\ that \quad x_n + x_w \leq 1 \quad (\forall_{\{n,w\} \in E}) \tag{4.2}$$

$$x_n \in \{0, 1\} \quad (\forall_{n \in N})$$

The edge formulation of node packing gives poor bounds on the search space of the problem. The node packing problem can be transformed into a clique formulation.

A clique of G is a subset of the nodes with the property that each pair of nodes in that subset is linked by an edge. A packing contains no more than one node from each clique of G. Maximal cliques are known to be possible integral facets for the node packing problem. Facets are restrictions to the solution space.

The problem description now becomes:

Clique Formulation of Node Packing:

$$MAX \sum_{n \in N} x_n \quad such\ that \quad \sum_{n \in C} x_n \leq 1 \quad for\ all\ Cliques\ C \tag{4.3}$$
$$x_n \in \{0,1\} \quad (\forall_{n \in N})$$

This problem description provides a least as good and generally much tighter bound on the problem since every edge is contained in some clique. There is a linear inequality for each clique in the graph. Unfortunately the number of cliques is possibly exponential. It is NP-hard to determine all maximum cliques, so it can not be expected to solve the problem exactly, but we can get an upper bound on its optimal value by finding a solution that satisfies a subset of all clique inequalities.

4.2 Odd Holes

Other known possible facets are odd holes without cords. A hole in a graph is a subset of nodes (of at least 4 nodes) such that the subgraph induced by the subset is a cycle without chords. Without cords means that no 2 nodes of the subset can share an edge that does not belong to the cycle. A hole is said to be an odd hole if it contains an odd number of nodes.
If P is a node packing and the set of nodes H forms an odd hole of size 2k + 1, then $|P \cap H| \leq k$:

Odd Hole inequality:

$$\sum_{n \in H} x_n \leq k \ \ where \ \ |H| = 2k + 1 \tag{4.4}$$

The odd hole inequalities are satisfied by all incidence vectors (the nodes $n \subset P \cap H$) of packings P. An interesting property connected to odd holes is that they cannot be obtained from linear combinations of the clique inequalities and non-negativity

so they can be used to tighten the relaxation of the node packing model more to an all integer solution.

Unfortunately the odd hole inequalities generally do not define facets of the convex hull of packings (i.e. the solution space of the problem). To obtain facets the odd hole inequalities have to be lifted. Lifting is adding variables that are not in the odd hole set H into the odd hole inequality. The coefficients α_n are as large as possible while validity of the equation is preserved.

Lifted Odd Hole inequality:

$$\alpha_n \geq 1 \ and \ \alpha_n \ \text{ is integer.}$$
$$\sum_{n \in H} x_n + \sum_{n \notin H} \alpha_n x_n \leq k \ \ where \ \ |H| = 2k + 1 \qquad (4.5)$$

Figure 4.1 is an example of a graph representing an odd hole (as stated in formula (4.4)) that can be lifted (according to formula (4.5)).

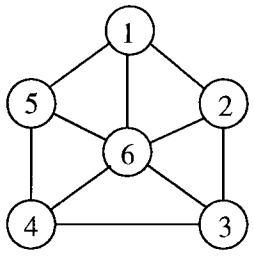

Figure 4.1: Odd Hole formation by nodes 1 to 5. Possible lifting with node 6 with $\alpha_n = 5$.

The total number of all lifted odd hole inequalities is possibly exponential. So the finding of all of the lifted odd hole inequalities is in general cases not run time efficient. Again the application of a subset of all lifted odd hole inequalities gives an upperbound on the exact value of the solution.

Chapter 5

Integer Programming Scheduling

The set of scheduling algorithms contains not many optimal solution methods because of their run time inefficiency. Heuristics give better run time results in many practical cases. This does not mean that optimal solution methods should not be the subject of research. They can very well be used as a basic approach that can be refined with additional algorithms. In that way they become practically useful. One of the optimal solution methods is scheduling using Integer Linear Programming (**IP**).

The general idea behind the **IP** approach to scheduling is to describe the scheduling problem as a set of linear inequalities. This set consists of the constraints which the schedule is limited by and the object function that has to be optimised. Solving the system of linear inequalities with an **IP** solver leads to an optimal solution but not within a certain predictable time limit because of the NP-complete character of the scheduling problem. So it is important to see that **IP** scheduling can only be of practical use if special properties of high level synthesis scheduling are used in the modeling to limit the complexity of the **IP** problem.

The scheduling problem is modeled as an assignment problem, where the variables represent a placement of DFG operations in 2-dimensional space. The 2-dimensional space is defined by time (control steps) and area (functional modules). From the graph initially a variable is generated for each possible scheduling combination of operation, module (or module type in case of time constrained scheduling) and execution cycle step.

A general **IP** solver is used to optimise the object function while preserving the constraints. The **IP** solver tries to find a solution for the schedule by assigning

values to the variables that are feasible to all linear equations. It is exactly known through these variables which operation is scheduled in which cycle step onto which module or module type. The set of linear inequalities of the **IP** formulation of the schedule problem consists of three kinds of constraints following the formulations in section 6.2.

After the generation of the constraints according to the **IP** model for scheduling it is solved as a standard **IP** problem. First the **LP** relaxation of the problem is solved. If the solution is not all integer the branch-and-bound algorithm is used to achieve that. Special models as node packing can be used to try to limit the necessary branch-and-bound influence. In the following chapter the possibility of further improvements through applying the node packing model to the **IP** scheduling problem is described (chapter 6).

5.1 Time vs Resource Constrained IP Scheduling

The major difference between the resource and time constrained implementation of **IP** scheduling is the way modules are handled. Resource constrained scheduling uses modules as resources. Time constrained scheduling applies module types as resources. This different approach to resources has consequences:
Resource constrained **IP** schedulers sometimes have short run times due to the application of efficient models like node packing. Time constrained scheduling cannot be modeled as a strict node packing problem because of the module constraints which do not represent cliques [Gebo91]. Therefore time constrained **IP** schedulers are hardly very efficient, good models are not known for this scheduling problem. It is clear that improvements have to be made before it is possible to use **IP** scheduling in practical cases and the time constrained version needs adaptations if to be used at all. The test results (See Appendix A) from the **IP** scheduler support these conclusions. The implementation of the resource constrained scheduler is already extended with a procedure for reducing execution intervals of operations. It gives the upper bound on the execution intervals and that makes the scheduler feasible constrained (find the best fit) rather than resource constrained. In this fashion the scheduler is modeled as pure node packing. In the resource constrained version there is possibly still corruption of the model due to the object function that is not according the node packing model. By finding the lower bound on necessary resources this is transformed into a feasible scheduling problem that is certain to be a node packing formulation.

5.2 A Competetive Exact Scheduling Scheme

The **IP** method is not the only exact method for scheduling. For example: the branch-and-bound algorithm can also be used under types of schedulers that do not apply integer programming. The task of **IP** and additional modeling is *pruning* the search space of the problem to reduce search time. In Binary Schedule Graph (BSG) scheduling this can efficiently be taken care of by bipartite graph matching. Many of the common high level synthesis systems schedule operations by assigning their begin values directly to specific cycle steps. In [Timm95] it is proven that for feasible scheduling (with time and resource constraints applied in the model) the existence of a schedule can be decided more efficiently by finding a correct ordering of the operations. A correct ordering means a linear ordering that corresponds to a feasible schedule.

This leads to the following schedule approach: An ordering is imposed by matching for each module type the operations with their Operation Execution Interval (OEI) one-by-one to a specific Module Execution Interval (MEI). Edges that can not be part of a complete matching are removed and the corresponding execution intervals are reduced by this.

In the branch-and-bound process it is also interesting to try to start the process with the best suited variables. The use of bipartite graph matchings give the possibility to find those bottlenecks in the problem. Analysis of the matchings is an effective way of steering the process.
More on run time efficient branch-and-bound scheduling with use of bipartite graph matching can be found in [Timm95].

The reason that BSG scheduling is introduced here is that the process of solving the scheduling problem can benefit from the properties of both the BSG- and **IP**-model if they are applied in a concurrent way.

Chapter 6

IP Model Optimisation

6.1 Introduction

In this chapter improvements are given on the **IP** model for resource constrained scheduling. They are the result of exploitation of the node packing model and using a structural approach toward the solving process. In [Timm94] it is shown that the only possible efficient way to use the **IP** scheduling model is resource constrained in combination with lower bound estimated time constraints. So the total constraint set results in a feasible type of scheduling. With feasible scheduling the goal is to get a fit solution for the set of constraints without further regard of an object function.

In general scheduling cases the node packing model leads to a fractional solution of the **IP** relaxation. This imperfection raises the question whether modeling the scheduling problem as an **IP** problem can result in run time efficiency improvements at all. [Nemh92] introduces a standard procedure structure to solve node packing problems in the most efficient run time possible:

- Preprocessing: basic **IP** modeling (node packing constraints).
- Applied heuristics (if available).
- **LP** relaxation.
- Constraint generation: processing non integer variables.
- Branch-and-Bound.

The standard techniques in the procedure are the **LP** relaxation and branch-and-bound process. They have already been described in the chapter 5. Heuristics are not used in our model because the goal of the assignment was to test the value of the optimal method of integer programming to scheduling. Constraint generation considers the results of the relaxation and tries to bring that closer to the desired all integer solution before applying the inefficient branch-and-bound process.

In practice the solving of an **IP** scheduling problem depends heavily on the branch-and-bound process. It is therefore of importance that the model results in an as tight as possible search space before the branch-and-bound is started. In this chapter the features in the suggested procedure and their application to the **IP** scheduling problem are described.
The node packing model is the basis for this process.

6.2 The Applied Node Packing Model

Recent research [Gebo92] showed that applying the node packing model [Nemh88] to the **IP** scheduling problem (**IP:NP**) sometimes can lead to short run times. The node packing model is introduced through an adaptation of the linear inequality formulation. The constraints are all clique inequalities. In [Gebo92] proof is given that this formulation compared to earlier formulations leads to a tighter solution space for the **IP** solver. According to [Gebo92] solving the precedence constraints together with module- and operation assignment constraints solves the following problem: It produces a schedule, by simultaniously mapping each code operation to a time cycle (maintaining the partial order among operations), and binding each operation to a functional module.

In this section the feasible constrained **IP** scheduling formulation is presented. The following notations are used:

- A data flow graph DFG is represented by a tuple (V, E), where V is the set of nodes (operations) and E the set of directed edges representing dependencies between operations.
- C is the list of available cycles, $c \in C$.
- $EI(v)$ is the list of cycles (execution interval) in which operation $v \in V$ could be performed.

- L is the set of module types, $n(l) \in N \backslash \{0\}$ is the number and $area(l)$ the area of modules of type $l \in L$.
- M is the set of available modules, $m \in M$.
- $\xi(v)$ is the mapping from an operation $v \in V$ to a module type in the set L.
- $\mu(v)$ is the set of modules that operation $v \in V$ can be mapped on.
- $d(v)$ is the delay of an operation $v \in V$ mapped on module type $\xi(v) \in L$.
- $dii(v)$ is the data introduction interval of an operation $v \in V$ mapped on module type $\xi(v) \in L$ (the dii is the minimal number of cycles required between the data arrivals for two executions).
- $x(v,c,m) = 1$ $(x(v,c,m) \in \{0,1\})$ represents that operation $v \in V$, binded to module $m \in M$ starts its execution in cycle $c \in C$.

For the rest of this thesis the applied module library is trivial , i.e. operations can be assigned to only one module type. The model is applicable on non-trivial module libraries as well.

The **IP:NP** scheduling constraints are:

The operation assignment constraints ensure that the start of each operation is assigned to one cycle step and to one module.

$$\forall_{v \in V} : \sum_{c \in EI(v)} \sum_{m \in \mu(v)} x(v,c,m) = 1, \ \forall_{c \notin EI(v)} : x(v,c,m) = 0. \qquad (6.1)$$

The module constraints prevent that more than one operation is assigned to a module in the same cycle step.

$$\forall_{m \in M} \forall_{c \in C} : \sum_{v \in V | m \in \mu(v)} \sum_{s=c}^{c+dii(v)-1} x(v,s,m) \leq 1. \qquad (6.2)$$

The precedence constraints ensure that the precedence relations between operations in the DFG are maintained:

$$\forall_{(v,w) \in E} \forall_{c \in EI(v) \cap EI(w)} :$$
$$\sum_{m \in \mu(w)} \sum_{s \leq c+d(v)-1} x(w,s,m) + \sum_{m \in \mu(v)} \sum_{c \leq s} x(v,s,m) \leq 1. \qquad (6.3)$$

The application of node packing can be modeled by a node packing graph defined by the constraints: nodes for all variables $x_i \in \{0,1\}$, and edges for pairs of variables that can not have value 1 at the same time. The gain is found through cliques in the node packing graph that form possible facets of the solution space of the problem.

An example of a node packing graph is shown in fig 6.1. The edges in the graph are formed using the equations of this section, they are all illegal scheduling combinations. That kind of graph is called a conflict-graph.

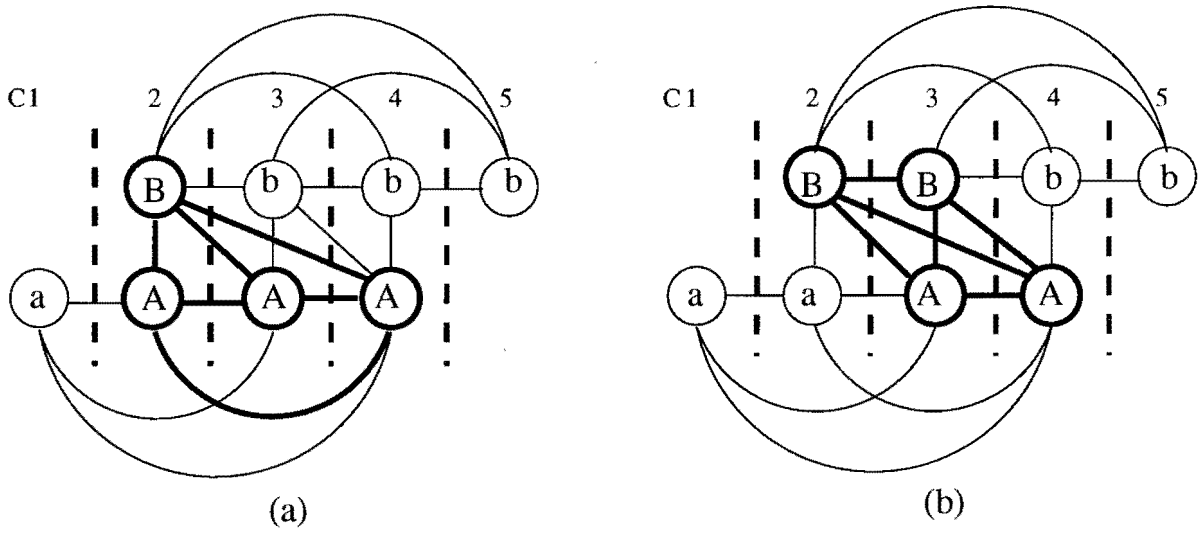

Figure 6.1: Node packing graph for 2 operations with precedence relation: a before b, showing in bold a clique facet for cycle step c=2 in (a) and c=3 in (b) of inequality 6.3

6.3 Constraint Generation

Constraint generation is the step in the procedure that reviews the **LP:NP** relaxation of the standard constraint set that is generated out of a given problem in the first steps. In most cases the **LP:NP** relaxation results in a not-all-integer solution. If the relaxation is not all integer it means that the search polyhedron is not tight enough. It can be tried to formulate extra constraints within the model to constrain the variables that are not integer. This should lead to a **IP:NP** formulation with more facets of the solution space of the problem. The search is for constraints that can describe such facets, like: maximum cliques or odd holes without cords.

The constraints have to be built with the non integer variables that were violated in the relaxation of the problem. Then those constraints could be added to the original constraint list. This would again restrict the search polyhedron. The effort is to find so many violated constraints that the search polyhedron is restricted to the integer solution which is implied by the description with all constraints of facets. If this can not be achieved it is already satisfactory if the search space of the problem is restricted so the underlying branch-and-bound procedure can solve it with acceptable run time efficiency.

The process of constraint generation can be visualised in the following way:
The **NP** model of the scheduling problem can be seen as a conflict model so the **NP** graph can be seen as a conflict graph. This means that all edges in the graph connect the operations that can not be scheduled in the same time cycle at the same module. The **NP** graph is a construction of all impossibilities in the schedule.

- The search for extra constraints is a search for more scheduling impossibilities.
- Another possible step forward in the model is the search for combinations of edges that are already in the conflict graph (from different constraints) but are not explicitly stated in the constraint list. This means that they are not directly considered in the relaxation and that they have no effect as bound on the search space.

Chapter 7

Model Enhancement Constraints

7.1 Clique Maximisation

The easiest enhancement of the model is the search for cliques that can be extended (because they were not maximal). The only possibility that was found which can appear in general cases of scheduling DFGs, is the case that 3 or more operations are all precedence related: operation A before B, B before C, A before C and also their execution intervals overlap. This results in a clique formulation involving A,B and C. An illustration of such a situation is shown in figure 7.1.

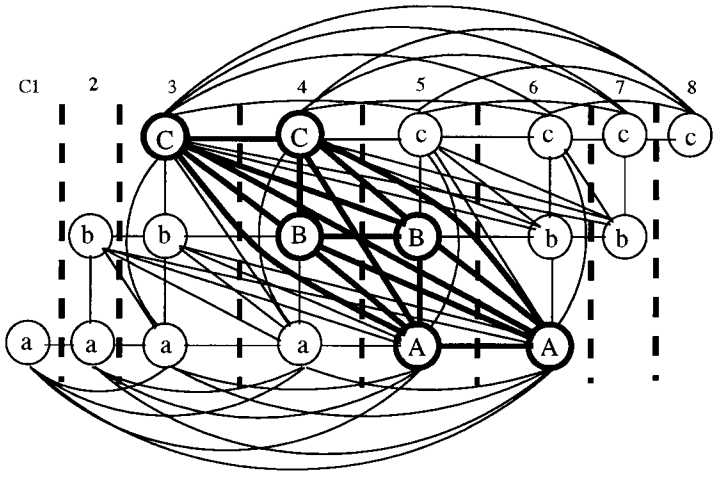

Figure 7.1: Node packing graph for 3 operations with precedence relation: a before b, b before c, a before c, showing in bold a clique for cycle step c=5 of inequality 7.1. Most edges between all variables of operation b have been omitted for clarity.

Multi Operation Precedence Clique Constraints (MOPC):

$$\forall_{(v_1,v_2),..,(v_1,v_i),(v_2,v_3),..,(v_2,v_i),..,(v_{i-1},v_i)\in E} \; \forall_{c\in EI(v_1)\cap EI(v_2)..\cap EI(v_i)} \; :$$

$$\sum_{w=2}^{i} \sum_{m\in\mu(v_w)} \sum_{s=c_w=c_{w-1}-d(v_w)+1}^{c_{w-1}+d(v_{w-1})-1} x(v_w,s,m) + \sum_{m\in\mu(v_1)} \sum_{c1=c\le s} x(v_1,s,m) \le 1 \qquad (7.1)$$

(7.1) is the general formal representation of possible qlique maximisation as in the situation that is mentioned in the previous paragraph. Since this situation is very limited in existence it is not the unique extension that is needed to enhance the model.

7.2 Indirect Precedencies

An addition seemed to be the use of so called indirect precedencies. These are precedence relations between operations that are predecessors or successors but not in the direct sense. They are connected by a path in the DFG. A part of their respective execution intervals is not valid if they are both scheduled in that part. The minimal length of execution time of the interconnecting DFG path is to long to obtain a valid schedule. This is illustrated in figure 7.2.

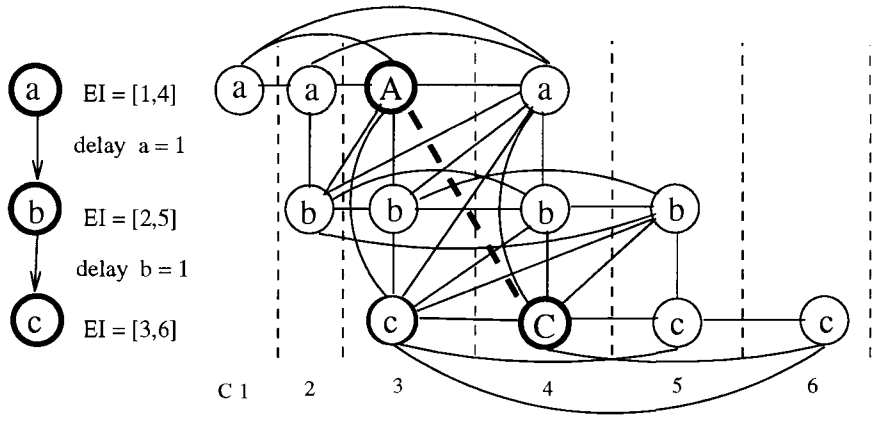

Figure 7.2: (a) a DFG example with operations a,b,c with their execution interval between brackets, (b) The node packing graph equivalent of (a). The bold dotted edge is an example of an indirect precedence relation, between a and c. If operation a is scheduled in cycle 3 then c can not be scheduled in cycle 4, because b has to be executed in between. This edge is not present in the standard constraints.

Unfortunately this extra set of precedence relations is not of direct importance since

in [Chau94] is proven that the resulting indirect precedence clique constraints are linear dependent on the direct precedence constraint set. Still they can be of use in other sets of violated constraints on top of existing precedence relations as we will mention later.

7.3 Violated Clique Constraints

A set of clique constraints that is not linear dependent on one of the known **IP:NP** model constraints is a crossover between resource and precedence relations.
The proof of linear independence consists of actual violation of these constraints after relaxation of the **LP:NP** formulation. Precedence relations are defined between pairs of operations (see 6.2). If an operation maintains that relationship with more than one operation that can be scheduled on the same module, we speak of multiple precedencies. Extra cliques exist, caused by the combination of those multiple precedencies and resource demands. An example of a clique of that kind is shown in figure 7.3.

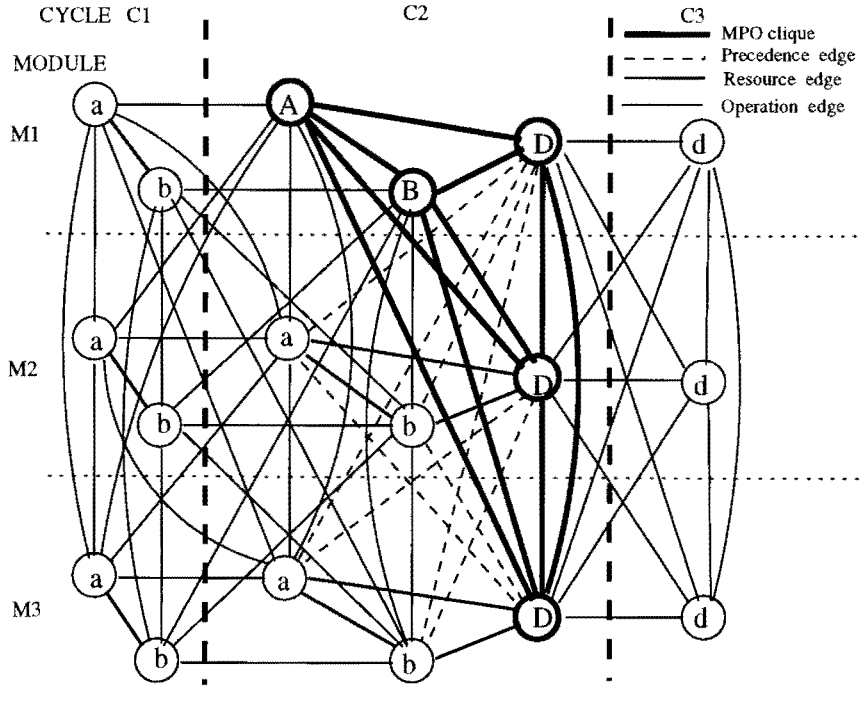

Figure 7.3: Node packing graph for three operations with 2 precedence relations: a before d, and b before d, showing in bold a clique for cycle step c=2 of inequality 5.2. A **MPO** clique exists also between the other operation sets {a,b} and d in c=2

Formula (7.2) represents the set.

Multi Precedent Operations Constraints (MPO):

$$\forall_{\{(v_1,w),..,(v_i,w)|\mu(v)=\mu(v_1)\cap..\cap\mu(v_i)\neq\emptyset\}}\ \forall_{c\in EI(v_1)..\cap EI(v_i)\cap EI(w)}\ \forall_{m\in\mu(v)}:$$
$$\sum_{n\in\mu(w)}\sum_{s\leq c+d(u)-1} x(w,s,n) + \sum_{u=v_1}^{v_i}\sum_{s=c-d(u)+1}^{c} x(u,s,m) \leq 1 \qquad (7.2)$$

These multiple precedencies can exist between an operation and sets of operations through direct precedence relationship (successors, predecessors) and also with sets through the indirect precedence relationship, mentioned in the previous section.

The second set of clique constraints is a consequent development of the previous. It exists if there are several operations, that can be scheduled from the same time cycle and on the same module, that have a multiple precedence relationship with another group of operations with the same property (not necessarily the same module type). The result of such a relation between two groups of operations is a set of cliques that exists across modules. An example of this set of constraints is shown in figure 7.4. Formula (7.3) represents the clique constraint set.

Multi Precedent Sets Constraints (MPS):

$$U = \{v_1..v_i\} \subset V \ \ with\ \ \mu(U) = \mu(v_1)\cap..\cap\mu(v_i) \neq \emptyset,$$
$$T = \{w_1..w_i\} \subset V \ \ with\ \ \mu(T) = \mu(w_1)\cap..\cap\mu(w_i) \neq \emptyset:$$
$$For\ \ all\ \ u \in U, t \in T \rightarrow (u,t) \in E$$
$$\forall_{U\subset V}\ \forall_{T\subset V}\ \forall_{c_1,c_2\in EI(v_1)..\cap EI(v_i)\cap EI(w_1)..\cap EI(w_i)}\ \forall_{m\in\mu(T)}\ \forall_{n\in\mu(U)}:$$
$$\sum_{t=w_1}^{w_i}\sum_{s=c_2-d(t)+1}^{c_2} x(t,s,m) + \sum_{u=v_1}^{v_i}\sum_{s=c_1-d(u)+1}^{c_1} x(u,s,n) \leq 1 \qquad (7.3)$$

Again this relation exists between groups of operations with direct or indirect precedence relations between them.

The main interest in these 3 clique constraint sets is to obtain more control on the use of resources in the enhanced **NP** model. In the standard **NP** model (of section 6.2) the control is limited to single modules, not on the number of modules of a certain type. Because the extra multiple precedence constraints relate different modules, they can help to enhance the model if they are added to the list of constraints.

It is proven in [Timm94] that the **MOPC** and **MPO** clique constraints form a

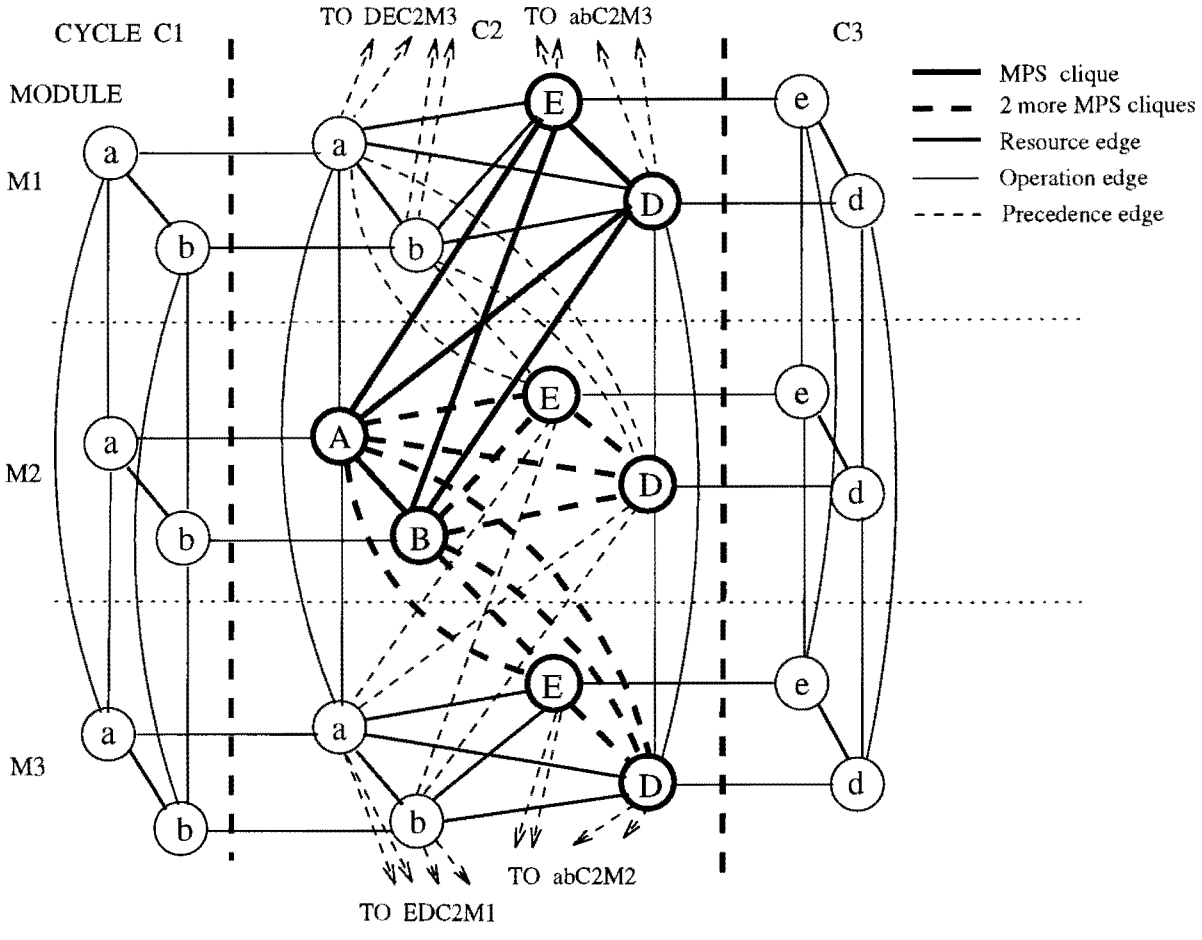

Figure 7.4: Node packing graph for 2 operation sets with precedence relations: {a,b} before {e,d}, showing in bold a clique for cycle step $c_1 = 2$ and $c_2 = 2$ of inequality 5.3. For clarity reasons operation edges have been omitted where necessary, but all variables of the same operation are fully interconnected over cycles and modules

closer cut of the search polytope than the constraint enhancement mentioned in [Chau94]. Also these constraint sets are formed as possible facets according to the node packing model. However if these clique sets do not appear in the **IP:NP** model, the formulation in [Chau94] is still valid. The latter is more universal but it is not a node packing formulation so it does not represent facets of the **NP** polytope in the strict node packing scheduling model.

7.4 Odd Holes Constraints

After the discovery of two sets of violated clique constraints the relaxation of exemplary scheduling problems still had no complete integer solution. Literature on the theory of node packing problem optimising (see [Nemh92]) states apart from maximal cliques also odd holes as possible facets of the polyhedron. Research continued in the direction of finding odd holes in the **IP:NP** scheduling model.

In the case of the **IP:NP** scheduling model the found odd holes are a consequence of the earlier mentioned multiple precedence and resource constraints.
A set of operations U, that can all be scheduled on the same modules, has a multiple precedence relationship with one operation (**MPO**). The odd hole is formed with variables from operations in that set, with modules of one type (l). The number of variables in the odd hole must be $2 * n(l) + 1$ so the odd hole equation limit (k in formula (4.5)) is $n(l)$, the number of available modules of the module type. This will constrain the schedule in cycles that the odd holes are situated with the number of modules of that type.

The odd hole can safely be lifted with all other variables within the same cycle of the operations that have variables in the odd hole. These variables must have lifting factor $\alpha_n = 1$, the odd hole equation is preserved because their sum will always be smaller than the number of available modules of the type l.

Next the odd hole is lifted with a variable of the operation that holds the multiple precedence relation with the set U in the **MPO**. From now on this is called the lifting variable. Because this operation is precedence related with all operations with variables in the hole, its lifting factor can be $\alpha_n = n(l)$. This can be done with all variables of this operation that are related to the odd hole through the **MPO**.

The odd hole constraints as they can be used in the **IP:NP** scheduling model are described in formula (7.4).An example is worked out in figure 7.5.

Resource Precedence Odd Hole Constraints (RPOH):

$$U = \{v_1..v_i\} \subset V \ \ with \ \ \xi(U) = \xi(v_1) \cap .. \cap \xi(v_i) \neq \emptyset,$$
$$For \ \ all \ \ sets \ \mathbf{MPO}(U, w) \ \ with \ \ i \geq 3 \ \ \forall_{c \in EI(v_1)..\cap EI(v_i) \cap EI(w)} \ \ \forall_{l \in \xi(U)} :$$

$$\sum_{u=v_1}^{v_i} \sum_{n \in M | type=ly} \sum_{s=c-d(u)+1}^{c} x(u,s,n) + \alpha_w \sum_{m \in \mu(w)} \sum_{s \leq c+d(u)-1} x(w,s,m) \leq n(l) \qquad (7.4)$$

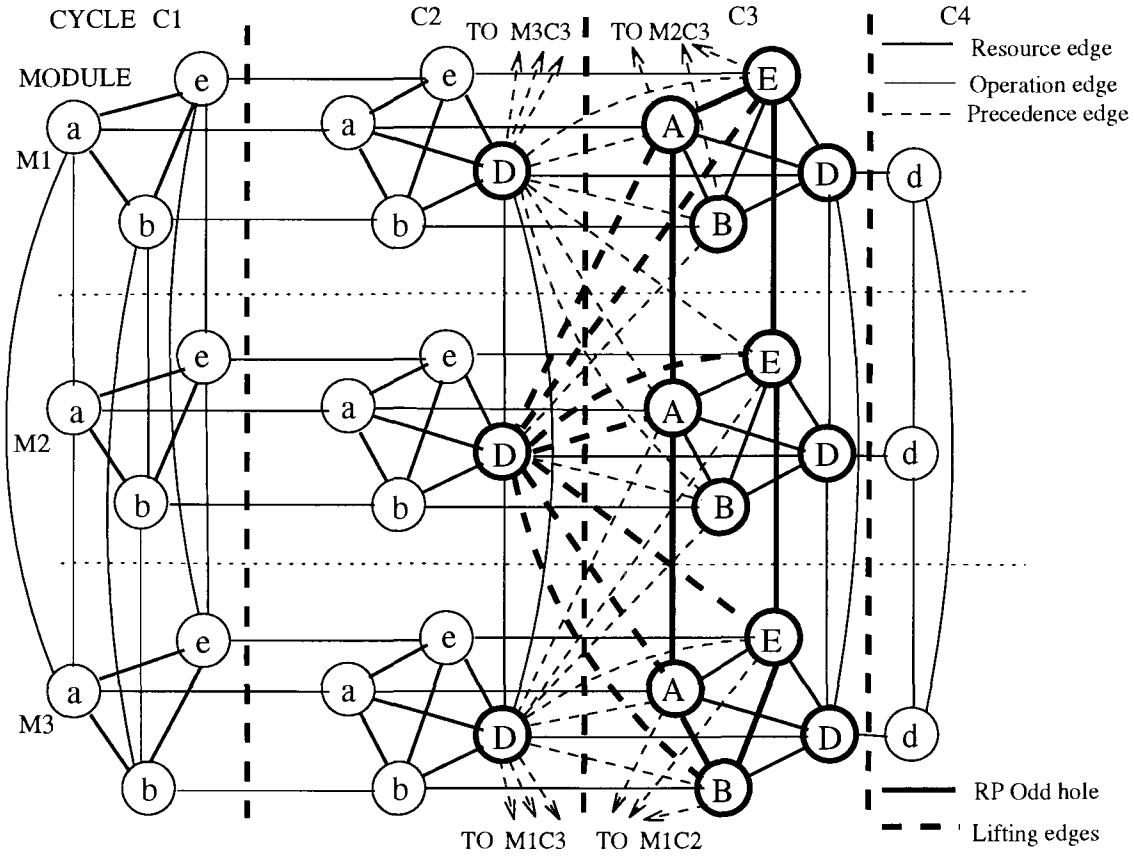

Figure 7.5: Node packing graph for operations with resource odd hole relation: showing in bold an odd hole in cycle step c=3 with possible lifting with operation d in cycle step c=2 due to precedence relations between (a,d), (b,d) and (e,d). Here $\alpha_w = n(l) = 3$, all other bold faced operations can be added according to inequality 5.6. For clarity some operation edges have been omitted, all variables of the same operation are fully interconnected over cycles and modules. For the same reason the precedence edges between d and a,b,e in cycle c=2 have been omitted.

It can be interesting for the result of the **RPOH** to have a trade off (if possible) in the odd hole between the multiplication factor $\alpha_w - 1$ of the lifting variable and another variable. This can be done with variables from operations in the set R. The operations in R can be scheduled on modules of the same type and in the same time cycle as the operations with variables in the odd hole. However they are not in the related **MPO** because they have no precedence relation with the operation of the lifting variable. Of course this trade off can only be done as long as $\alpha_w > 1$. The possibility for trade off occurs if the value of the new variable (after relaxation) is larger than the value of the lifting variable. It can also be of interest to involve as many operations as possible in the constraint of the same module type to get

more global constraining. The trade off is formulated in(7.5). α_{w2} is the new multiplication factor of the lifting variable w after the trade off.

Enhanced Odd Hole (EOH):

$$O = \{v_1..v_i\} \subset V \ \ with \ \ l = \xi(O) \in \xi(v_1) \cap .. \cap \xi(v_i) \neq \emptyset,$$

$$For\ all\ sets\ \mathbf{RPOH}(O,w) \ \ with \ \ \alpha_w > 1 \ \ :$$

$$\{\mathbf{RPOH}(O,w)\ with\ \alpha_{w2} = \alpha_w - |R| \ \wedge \ \alpha_{w2} \geq 1\}$$

$$+ \sum_{\{r \in R | (r,w) \notin E \ \wedge \ \xi(O) \in \xi(r)\}} \ \sum_{n \in M | type = \xi(O)} \ \sum_{s=c-d(r)+1}^{c} x(r,s,n) \leq n(l) \qquad (7.5)$$

Another possible improvement of the effect of the odd hole constraint is the lifting with variables of an indirect precedent operation (as mentioned in section 7.2) instead of a direct precedent operation. The set of indirect precedence related operations in the odd hole can be larger than a direct related set. This way more operations are involved in the constraint, creating larger impact on the **IP:NP** search space. This improvement represents a more global constraint on resources. A formulation is given in (7.7).

Multiple Indirect Precedence Set (MIS):

A path P of precedent related edges exists in a DFG G between: node $v \rightarrow$ node w then $d(P)$ is the sum of delays of operation nodes in P including v up to w.

$$\mathbf{MIS}(U,w) = \{(v,w) | v \in U \ \wedge \ \exists \ P : v \rightarrow w \wedge \ EI(v) \cap (EI(w) - d(P) + 1) \neq \emptyset\} \qquad (7.6)$$

Indirect Precedence Odd Hole (IOH):

$$EI(U) = EI(v_1) \cap .. \cap EI(v_i)$$

$$For\ all\ sets\ \mathbf{MIS}(U,w) \ \ with \ \ i \geq 3 \ \ \forall_{c \in EI(U) \cap (EI(w) - d(P(U)))} \ \ \forall_{l \in \xi(U)} :$$

$$\sum_{u=v_1}^{v_i} \ \sum_{n \in M | type = l} \ \sum_{s=c-d(u)+1}^{c} x(u,s,n) + \alpha_w \sum_{m \in \mu(w)} \ \sum_{s \leq c+d(u)-1} x(w,s,m) \leq n(l) \qquad (7.7)$$

The **IOH** constraints can be refined according to (6.5) in the same way as **RPOH** constraints with operations from U that are not in $\mathbf{MIS}(U,w)$.

Chapter 8

Applied Optimisation Results

All suggested model optimisations from the previous chapter (7) have been tested on several standard exemplary graphs. A very good test case was the Fast Discrete Cosine Transformation graph (FDCT from [Deny90]) with different upper bound time restrictions. The different optimising possibilities will be treated in this chapter. The first section describes the way that they can be applied in the **IP:NP** model. The results of the different optimisations will be described in the second section as well as an evaluation of their effects.

8.1 Implementation of Optimisations

The first step in the procedure of using the **IP** model optimally as stated in chapter 6 is the **IP:NP** clique formulation. As described in section 6.2 this basic modeling is implemented for the scheduling problem after [Gebo92] . This model has been implemented and tested thoroughly in previous research (see [Leeu93] for the implementation and results in Appendix A).

Additional constraints for this model as described in the previous chapter 7 were implemented in 2 different ways.
The first way follows the procedural structure of optimising the use of the **NP** model as stated in chapter 6: First **IP:NP** modeling and a **LP:NP** relaxation step, then find constraints that are violated by non integer variables in that relaxation. These violated constraints are added to the constraint list of the **IP:NP** model and again **LP:NP** relaxation follows. This procedure can be repeated to try to make all

variables in the relaxation integer and a solution is reached.
The second way profits of the relaxation analysis of the first way. The additional constraints are put into a general description and they are implemented into the basic **IP:NP** model. This way there is immediate profit of the extra constraints and relaxation can be closer to the solution.

8.2 Evaluation

The results of the **IP:NP** model application on scheduling FDCT with the 3 basic constraint sets according to [Gebo92] looked promising but could not convince in all cases. FDCT with an upper bound of 9 or 10 time cycles with the used (wrong) lower bound estimation of necessary resources is not feasible, the **IP:NP** modeling could not find that out in acceptable time (see Appendix A).

Analysis of the **LP:NP** relaxation during tests learned that the solution seldom is all integer at once. This means that the **IP:NP** model of the problem is not as tight as should be. Mostly the underlying branch-and-bound procedure has to be used to come to an integer solution. Infeasible problems take unacceptable time to solve because all (impossible) solutions have to be checked before they can definitively be rejected.

Analysis of the relaxation has uncovered already three kinds of constraints that are violated by non integer variables. The method of adding violated constraints after the relaxation step gives good insight in the consequences of the applied **IP:NP** model to scheduling problems.
The addition of violated clique constraints results in a change in the relaxation that is more toward a solution of the involved variables. The search space of the problem shrinks by adding previously violated constraints.

A direct negative result is a shift of non integer values to other variables that give cause for other violated cliques in the next solution step. This can reset variables that were integer in the previous relaxation to non-integer values. This happens if there exists parallelism in the graph that is not completely covered by specific constraints.
From this observation can be concluded that the solution will not be all integer if not all possible violated clique constraints are added in the model. It is efficient to put the complete set of potential violated constraints already in the basic **IP:NP** model. Then of course a formal description must be available.

Unfortunately the number of violated constraints can become too large to be efficiently solved if they are linearly independent. An example of the large numbers of violated constraints can be found in Appendix B.

Final tests have been done to find an efficient applicable set of extra constraints with as strong as possible effect on the process. These tests were especially run to check their effect on the possibility of finding infeasibilities sooner than the original model or the bipartite scheduling graph model ([Timm95]) that is used for initial module selection and execution interval analysis. The tests resulted in further application of a restricted number of additional constraints only. Only odd hole constraints are applied in the model restricted to precedence relations between a group of operations that can all be scheduled on the same module type and an indirect predecessor or successor with only one predecessor or successor in between (see section 7.4). This limits the number of constraints that are added to the model considerably.

The tests that compared the models concentrated on detecting schedule configurations that are not feasible. Specifically FDCT with upper time bound 9 and 10 cycles with a lower bound estimation on necessary resources that is too low (i.e. not enough resources are reserved for a feasible schedule). The tests were run in the situation that the module selection is being performed with the bipartite schedule graph (BSG) method. Each step this method takes, the intermediate result can be verified on feasibility by using it in the **IP:NP** model and by computing a relaxation. If the relaxation solution from one of the models to the problem can conclude that it is not feasible then this method performs better then the others that fail.

The effect on the solution of the odd hole model enhancement in the cases of FDCT 9 and 10 time cycles, results in better performance than the standard model and the BSG model. It immediately detects the infeasibility of FDCT 9 cycles with estimated resources. The standard model fails to do that and the BSG model takes more time to come to that conclusion. In the case of FDCT 10 cycles with the estimated resources not one of the models comes to early detection of infeasibility. Still the odd hole model enhancement leads to better performance than the standard model or the BSG model. The detection of infeasibility of local parts in the search tree is in some cases earlier than the BSG model. Nevertheless global infeasibility of the problem is not detected. Results of verification tests on FDCT with upper time bound 10 cycles in the process of module selection can be found in Appendix D.

The application of more additional constraints to the model did not improve the results of the odd hole enhanced model. With this argumentation the limitation on extra constraints is justified. The execution time is still influenced by the generation of this limited extra set of constraints because of the number of constraints that is considerably higher than in the standard model. See the results of the relaxation tests on FDCT in Appendix C.

8.3 Other Applications

Due to the problems with exponential growth of the number of constraints and the inherent non integral result of the modeling if their number is limited, the application possibilities of the **IP:NP** model will be limited.
In case the search space is not all integer, the **IP** process is depending on a branch-and-bound process to come to an integer solution. Already a method is known for more efficient scheduling through a branch-and-bound method based on bipartite scheduling graphs (see [Timm94]). So the **IP** process is as a general stand alone scheduler inferior even in its best known possible form.

This does not mean that the **IP** model is of no value to the scheduling process. **IP** modeling gives a global view on the scheduling problem where for instance the BSG model works at a local level. Detecting infeasibility is a typical global problem. Tests showed that in some cases the extended **IP:NP** model comes to an earlier infeasibility detection of suggested schedules then other models. The application of the **IP:NP** model could be an extra verification tool on top of other models, as in module selection with the BSG model. The cases that the BSG model fails may be corrected by an **IP:NP** modeling.

Another possibility of applying the extended **IP:NP** model is the use of intermediate relaxations as a steering mechanism for the branch-and-bound process. Because the local attention of the branch-and-bound process it can not detect possible bottlenecks in graphs that must be scheduled. In many cases it is of interest to start branching in parts of the graph where there are not so many scheduling possibilities. This way the search tree will be less wide because each schedule decision can influence the rest of the process and limit other choices that still have to be made. Heuristics can be derived from the **LP:NP** relaxation to find the right starting variables for branching in the graph.

Both applications do not use the extended **IP:NP** model stand alone because that

would not be run time efficient. The **IP:NP** model is of use as a support tool to other possible models. Especially to the BSG model because they are directly related through the branch-and-bound process.

8.4 Future Work

In [Nemh92] is stated that if the problem model is pure node packing, variables can be fixed on integer values if they are produced by a **LP** relaxation because they will hold this integer value in an all integer solution. If the **IP:NP** model is applied to the scheduling problem, time constrained scheduling (due to resource constraints) and resource constrained scheduling (due to the object function) are not pure node packing problems. So this would only count for scheduling with a given upper time bound and an estimated lower bound on the number of resources (feasible scheduling). The consequences of this possibility could be very interesting for further application of **IP:NP** to the scheduling problem.

If all variables that are 0 or 1 in the relaxation can be fixed on there value, the problem can be reduced for following processing to a problem with less variables.It also gives a chance to generate specific additional constraints that are violated in the relaxation by variables that are not yet integer. It prevents variables that are already integer to participate in the relaxation again to satisfy the additional constraints so the effect of extra constraints is maximised. The amount of additional constraints would be restricted to those that constrain the non integer variables. Until now all possible violated constraints have to be added to get an all integer polytope and their number can grow exponentially.

Future research can be done on the consequences of this possibility. The applicability of **IP:NP** in the scheduling process would be greatly enlarged. For example together with the BSG model in the process of Execution Interval Analysis (EIA) (see [Timm94]) were execution intervals are minimised. After an EIA step using the BSG model the **LP** relaxation can be used to fix a number of integer variables. The integer variables influence the original EIA problem because it means that some operations are assigned or can not be assigned to modules and time cycles. That information can be added in the EIA which can be performed again on this new instance of the problem. Because the search space has changed this can give new, more restrictive results. EIA and **IP:NP** relaxation can be executed repeatedly until no more improvement occurs in the number of integer variables, then the strictest execution intervals possible through this method are available. This

improvement can be very helpful in large problems with great parallelism in the graph, to detect if there are possibly not enough resources reserved to come to a feasible schedule. Restriction of execution intervals leads to smaller scheduling problems because less variables are needed.

Due to large numbers of constraints and variables the **IP:NP** scheduling problem becomes execution time inefficient. Therefore it is also of importance for future research to try to decrease the number of variables in the problem. A possible method is the assignment of operations to the first possible module without regarding binding information (which modules are connected to which). This prevents that variables from that operation with all other possible modules are taken into account as well. This is a simple procedure if a trivial library of modules is used because all possible modules are of the same module type. In cases of non-trivial libraries it is necessary to know on what module type an operation has to be scheduled to perform this procedure. Tests have been performed with this procedure while using a trivial library, results can be found in Appendix E. The preliminary conclusion from these tests must be that there is no real improvement for general graphs using this method.

Chapter 9

Conclusions

Because of the exponential character of the necessary constraint generation it is not efficient to use the **IP:NP** scheduling model on general graphs. The underlying branch-and-bound mechanism supporting the Bipartite Scheduling Graph (BSG) modeling already has a far better performance. The use of the **IP:NP** model will be limited to the function of verification tool in module selection, steering mechanism to another branch-and-bound based scheduler and global infeasibility detection (in the process of execution interval analysis).

To achieve this goal the **IP:NP** model should be enhanced as far as possible. This makes the search space polyhedron of the problem as small as possible.
The enhancement is best done according to a standard procedure:

1. Basic modeling
2. **LP** relaxation
3. Violated constraint generation (repeated cycle with step 2. is possible)
4. Branch-and-bound (if requested).

Violated constraint sets can be put into the basic model or violated constraints can be added to the constraint list for next relaxation steps after they have been detected. An evaluation has to be done on the method of adding extra constraints on top of the standard **IP:NP** model. Because the first relaxation step has to have efficient execution time and the growth of the number of constraints is possibly exponential with growing number of variables, not all clique constraints can be

added in the first step. After the analysis of the relaxation still not all violated extra constraints can be added because of their large number which makes the solver less run time efficient.

Experiments showed that the extra odd hole constraints give the best results in our exemplary schedule problems. To get immediate profit of this extra set, they were added as constraint set in the basic **IP:NP** model. The result of this model enhancement was not as strong as was hoped.
The exponential growth of the number of violated constraints in growing problems confirmed the suspicion that there is no complete modeling possible in general graphs. Complete modeling means that the problem can be described with all possible facets of the solution polytope and that an all integer solution is immediately available. If modeling is complete then the number of constraints is too large, if the number of constraints leads to acceptable run time efficiency then the modeling is not as tight as possible. So not all facets of the solution polyhedron are modeled.

The positive result of the enhanced **IP:NP** model is achieved in combination with a BSG branch-and-bound scheduling model. In different cases that combination comes to a faster infeasibility detection result than the stand alone BSG branch-and-bound model.

There are still test problems that were not directly detected as infeasible as should have been. Therefore it should be stated that the modeling, even after total addition of the found extra constraint sets, is not as tight as possible. Large problems still have a search space polyhedron that is too big to solve efficiently. Even if more violated constraint sets can be found it would not be efficient to add them all to obtain a complete integer solution. Heuristics should be found to come to a satisfying and efficient solving method.

A possible improvement might be the fixation of integer variables after the **LP** relaxation in case of feasibility scheduling. If that process is repeatedly interchanged with execution interval analysis this could prune the search space of the problem in a fast way. This works especially well in cases of infeasibility or on graphs with a lot of parallelism. This improvement is based on the property of strictly formed node packing problems: integer valued variables in the relaxation stay integer in the resulting solution. This is an interesting future research subject.

Bibliography

[Alva90] R. Alvarez-Valdés, J.M. Tamarit, **"The Project Scheduling Polyhedron Dimension, Facets and Lifting Theorems"** European Journal of Operations Research 67, pp 204-220, North Holland, Amsterdam, 1990.

[Arts91] H.M.A.M. Arts, M.J.M. Heijligers, H.A. Hilderink, A.H. Timmer, **"High Level Synthesis of Digital Systems"** International Tutorial Paper, Sep. 1991.

[Berk94] M.R.C.M. Berkelaar, **"lp_solve"** a public domain MILP solver available by anonymous ftp from ftp.es.ele.tue.nl.

[Chau94] S. Chaudhuri, R.A. Walker, J. Mitchell, **"Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem"** Technical Report, Rensselaer Polytechnic Institute, Department of Computer Science, Troy, New York, July 1994.

[Deny90] P. Denyer, **"SAGE - A User Directed Synthesis Tool"** Proc. of the ASCIS Open Workshop on Synthesis Techniques for (lowly) multiplexed Datapaths, Leuven, Belgium, August 1990.

[DeSi90] C. DeSimone, **"Lifting Facets of the Cut Polytope"** Operations Research Letters 9, pp 341-344, North Holland, Amsterdam, 1990.

[Eijn91] J.T.J. van Eijndhoven, G.G. de Jong, L. Stok, **"The ASCIS Data Flow Graph: Semantics and Textual Format"** Eindhoven University of Technology EUT Report 91-E-251 June 1991.

[Gare79] M.R. Garey, D.S. Johnson, **"Computers and Interactability: A Guide to the Theory of NP-Completeness"** Freeman, 1979.

[Garf72] R.S. Garfinkel, G.L. Nemhauser, **"Integer Programming"** John Wiley & Sons,New York, 1972.

[Gebo91] C.H. Gebotys, M.I. Elmasry, **"Simultaneous Scheduling and Allocation for Cost Constrained Optimal Architectural Synthesis"** Proc. 28th DAC, pp. 2-7, 1991.

[Gebo92] C.H. Gebotys, M.I. Elmasry, **"Optimal VLSI Architectural Synthesis"** Kluwer, The Netherlands, 1992, ISBN 0-7923-9223-X.

[Hend91] TT.H.B. Hendriks, P. Van Beek **"Optimaliserings Technieken: Principes en Toepassingen"** Bohn Staflue Van Loghum, Houten,1991, ISBN 90-313-1241-X.

[Hoes93] C.P.M. van Hoesel, **"Introduction to Polyhedral Combinatorics, Cutting Plane Methods, and Branch-and-Cut Algorithms"** Eindhoven University of Technology, June 1993.

[Ková80] L.B. Kovács, **"Combinatorial Methods of Discrete Programming"** Mathematical Methods of Operations Research, Vol 2. Akadémiai Kiadó, Budapest, 1980.

[Leeu93] J.C. van Leeuwen, **"High Level Synthesis Scheduling Using Integer Linear Programming"** Training Report, Eindhoven University of Technology, The Netherlands, May 1993.

[McFa90] M.C. McFarland, A.C. Parker, R. Camposano, **"The High-Level Synthesis of Digital Systems"** Proc. of the IEEE, Vol. 78, No. 2, pp. 301-318, 1990.

[Mich92] P.Michel, U. Lauther, P. Duzy, **"The Synthesis Approach to Digital System Design"** Kluwer Academic Publishers Boston/Dordrecht/London 1992, ISBN 0-923-9199-3.

[Moré93] J.J. Moré, S.J. Wright, **"Optimization Softwareguide"** Frontiers in Applied Mathematics, vol. 14, SIAM, Philadelphia, 1993, ISBN 0-89871-322-6.

[NEAT92] M. Heijligers, H. Arts, R. Hilderink, A.H. Timmer, W. Philipsen, **"The NEAT System"** Private Communications, 1992.

[Nemh88] G.L. Nemhauser, L.A. Wolsey, **"Integer and Combinatorial Optimization"** Wiley Interscience, New York, 1988, ISBN 0-471-82819-X.

[Nemh92] G.L. Nemhauser, G. Sigismondi, **"A Strong Cutting Plane/Branch-and-Bound Algorithm for Node Packing"** Journal of the Operations Research Society, vol. 43, pp. 443-457, U.K., May 1992.

[Padb79] M.W. Padberg, **"Covering Packing and Knapsackproblems"** Annals of Discrete Mathematics, vol. 4, Discrete Optimization I, North Holland, Amsterdam, 1979.

[Rama91] L. Ramachandran, D.D. Gajski, **"An Algorithm for Component Selection in Performance Optimized Scheduling"** Proc. ICCAD-91, pp 92-95, 1991.

[Stok91] L. Stok, **"Archtectural Synthesis and Optimization of Digital Systems"** PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, July 1991.

[Taha75] H.A. Taha, **"Integer Programming, Theory, Applications and Computations"** Academic Press, New York, 1975.

[Timm93a] A.H. Timmer, M.J.M. Heijligers, L. Stok, J.A.G. Jess, **"Module Selection and Scheduling using Unrestricted Libraries"**, Proc. EDAC/EuroASIC '93, pp. 547-551, 1993.

[Timm93b] A.H. Timmer, J.A.G. Jess, **"Execution Interval Analysis under Resource Constraints"** Digest of technical papers of ICCAD '93,pp. 454-459, 1993.

[Timm94] A.H. Timmer, PhD thesis, preliminary version, Eindhoven University of Technology, Eindhoven, The Netherlands, 1994.

[Timm95] A.H. Timmer, J.A.G. Jess, **"Exact Scheduling Strategies based on Bipartite Graph Matching"** To appear in proc. of the European Design & Test Conference, Paris, France, March 1995.

[DeWi85] P. DeWilde, E. Deprettere and R. Nouta, **" Parallel and Pipelined VLSI Implementations of Signal Processing Algorithms"**, in S.Y. Kung, H.J. Whitehouse and T. Kailath, **"VLSI and Modern Signal Processing"**, Prentice Hall, pp. 258-264, 1985.

[Wols80] L.A. Wolsey, **"Heuristic Analysis, Linear Programming and Branch-and-Bound"** Mathematical Programming Study 13, Combinatorial Optimization II, North Holland, Amsterdam, Aug. 1980.

Appendix A

Standard Model Test Results

IP schedule results for the fifth wave digital filter from [DeWi85] WDELF (table 1) and the fast discrete cosine transform from [Deny90] FDCT (table 2). The test schedules are all pre-processed with execution interval analysis. The first column with CPU times represents the CPU times of the **IP** solver for the resource constrained method with upper time bound and lower bound area estimation. The second column with CPU times represents the time contrained method without lower bound area estimation. The CPU-times of the **IP** solver lp_solve (see [Berk94]) on a HP9000/755 (in seconds):

Table 1: results for WDELF.

cycles	mult d=2	add d=1	CPU times **IP** solver with LB estimation (sec) Resource constrained	CPU times **IP** solver without LB estimation (sec) Time Constrained
17	3	3	0.05	0.11
18	2	2	0.08	0.16
19	.	.	0.11	12.36
20	.	.	0.16	631.68
21	1	2	0.04	73.22
22	.	.	0.69	5416.80
23	.	.	0.49	stopped after 51607.3 sec
24	.	.	1.57	*
25	.	.	1.67	*
26	.	.	2.02	*
27	.	.	2.75	*
28	1	1	12.84	*

Table 2: results for FDCT.

cycles	mult d=2	add d=1	CPU times **IP** solver with LB estimation (sec) Resource constrained	CPU times **IP** solver without LB estimation (sec) Time Constrained
8	8	4	0.53	0.09
9	.	.	13988	*
10	5	4	*	*
11	4	3	266	*
12	.	.	78	*
13	4	2	851	*
14	3	2	268	*
15	.	.	801	*
16	.	.	507	*
17	.	.	419	*
18	2	2	578	*
19	.	.	306	*
20	.	.	409	*
21	.	.	662	*
22	.	.	1600	*
23	.	.	330	*
24	.	.	466	*
25	.	.	453	*
26	2	1	3093	*
27	.	.	*	*
28	.	.	2105	*
29	.	.	4765	*
30	.	.	4757	*
31	.	.	8738	*
32	.	.	5954	*
33	.	.	4467	*
34	1	1	*	*

* Stopped after several hours of CPU time.

Appendix B

Violated Constraints Search Results

Test cases of FDCT with feasible scheduling in 8-25 time cycles.
A comparative table with quantative results of 3 levels of search for violated constraints: No search, limited search to direct precedence and search for all violated constraints of the sets described in this thesis.

Table: Violated Constraint Generation for FDCT.

max cycle	initial number of constraints without search for violation	number of additional violated constraints after limited search	number of additional violated constraints after complete search
fdct	(lines)	(lines)	(lines)
8	142	6	234
9	189	342	1393
10	288	586	4974
11	250	527	3647
12	397	874	9770
13	444	670	9658
14	470	561	8999
15	525	797	11956
16	584	695	14601
17	639	1130	17675
18	602	499	11180
19	716	457	17713
20	772	677	20990
21	828	882	24211
22	885	889	27547
23	940	984	31582
24	996	1166	35721
25	1052	1274	39558

The number of violated constraints is counted after 1 lp-relaxation step.

Appendix C

Relaxation Test Results

Test cases of FDCT with feasible scheduling in 8-25 time cycles.
A comparative table of execution times. Showing the effect of additional odd hole constraints on the execution time of the relaxation. The **LP:NP** relaxation is computed by lp_solve v1.4 on a HP9000/735.

Table: Relaxation Times for FDCT.

max cycle fdct	CPU times of the relaxation of the **NP** model without extra constraints (sec)	CPU times of the relaxation of the **NP** model with extra odd hole constraints (sec)
8	0.1	0.2
9	0.4	1.85 (infeasibility detected!)
10	1.3	7.6
11	0.6	7.7
12	2.1	29.7
13	3.1	38.7
14	3.1	24.1
15	4.8	60.6
16	5.2	139.6
17	6.7	158.5
18	5.3	18.7
19	5.1	39.9
20	6.0	74.7
21	6.5	75.1
22	7.1	101.1
23	8.9	112.2
24	10.1	99.3
25	11.8	116.9

Appendix D

Module Selection Test Results

Verification of the initial BSG module selection process for FDCT with upper time bound 10 cycles and with an infeasible module set of 3 adders and 5 multipliers. The table shows the points in the search tree where the **IP:NP** model verification on top of the BSG model is successful in finding non feasibilities where the BSG model does not. The steps where the odd hole enhanced model is more effective than the standard model are bold faced. The verification is done by relaxation of the intermediate problem.

Table: Verfication Results for FDCT 10 cycles

(a): Part of search tree, with numbers 1/2/3:
1 = the number of vertices in the branch-and-bound search tree
2 = the number of branches , i.e. the number of of times an operation is matched without detecting immediately that the matching leads to an incorrect ordering.
3 = the number of times a matching was immediately detected to be incorrect, (so the branch-and-bound process does not follow such a branch)
(b): detected infeasible by standard model relaxation
(c): CPU times of the relaxation in seconds
(d): detected infeasible by odd hole enhanced model relaxation
(e): CPU times of the relaxation in seconds

(a)	(b)	(c)	(d)	(e)	(a)	(b)	(c)	(d)	(e)
6/1/27	**no**	**0.58**	**yes**	**1.19**	6/1/28	yes	0.67	yes	1.22
6/1/29	yes	0.38	yes	1.22	6/1/30	yes	0.61	yes	1.26
6/1/31	yes	0.4	yes	1.62	6/1/39	yes	0.61	yes	1.28
6/1/41	yes	0.65	yes	1.67	6/1/49	yes	0.38	yes	1.25
6/1/55	yes	0.4	yes	1.22	6/1/62	yes	0.42	yes	1.54
7/2/68	**no**	**0.6**	**yes**	**1.21**	7/2/69	yes	0.42	yes	1.93
7/2/70	yes	0.42	yes	1.51	7/2/71	yes	0.41	yes	1.78
7/2/72	yes	0.44	yes	1.35	7/2/80	yes	0.43	yes	1.78
7/2/82	**no**	**0.68**	**yes**	**1.41**	7/2/90	yes	0.39	yes	1.48

continued on the next page

(a)	(b)	(c)	(d)	(e)	(a)	(b)	(c)	(d)	(e)
7/2/96	yes	0.42	yes	1.49	7/2/103	yes	0.48	yes	1.39
8/3/104	yes	0.39	yes	0.95	8/3/106	yes	0.36	yes	1.18
8/3/107	yes	0.34	yes	0.92	8/3/109	yes	0.59	yes	1.29
8/3/112	yes	0.36	yes	0.94	8/3/113	yes	0.59	yes	0.99
8/3/115	yes	0.41	yes	1.32	8/3/117	yes	0.34	yes	1.21
8/4/119	**no**	**0.72**	**yes**	**1.57**	9/4/122	yes	0.34	yes	1.11
9/4/126	yes	0.34	yes	1.22	9/4/129	yes	0.34	yes	1.22
9/4/134	yes	0.35	yes	1.11	9/5/140	yes	0.73	yes	1.43
12/7/164	yes	0.65	yes	1.18	12/7/165	yes	0.42	yes	1.86
12/7/166	yes	0.42	yes	1.3	12/7/167	yes	0.44	yes	1.94
12/7/168	yes	0.41	yes	1.62	12/7/176	yes	0.43	yes	1.53
12/7/178	**no**	**0.66**	**yes**	**1.39**	12/7/186	yes	0.42	yes	1.57
12/7/192	yes	0.64	yes	1.31	12/7/199	yes	0.67	yes	1.68
13/8/205	**no**	**0.61**	**yes**	**1.2**	13/8/206	yes	0.42	yes	1.55
13/8/207	yes	0.66	yes	1.58	13/8/208	yes	0.44	yes	1.29
13/8/209	yes	0.45	yes	1.61	13/8/217	yes	0.69	yes	1.31
13/8/219	**no**	**0.71**	**yes**	**1.77**	13/8/227	yes	0.45	yes	1.51
13/8/233	yes	0.43	yes	1.75	13/8/240	yes	0.74	yes	1.44
13/9/241	**no**	**0.46**	**yes**	**1.44**	14/9/242	yes	0.41	yes	1.18
14/9/244	yes	0.42	yes	1.11	14/9/245	yes	0.39	yes	0.98
14/9/247	yes	0.42	yes	1.0	14/9/250	yes	0.58	yes	1.22
14/9/251	yes	0.4	yes	1.03	14/9/253	yes	0.43	yes	1.34
14/9/255	yes	0.39	yes	1.03	**14/10/257**	**no**	**0.64**	**yes**	**1.71**

Module selection search tree continues, verification test aborted after reaching this point.

Appendix E
Module Assignment Test Results

Test cases of FDCT with feasible scheduling in 8-25 time cycles.
A comparative table of execution times. Showing the execution times of the **IP:NP** scheduler with standard model versus the scheduler that assigns operations to modules without regarding binding information and this way restricting the number of variables in the **IP:NP** model.
Tests have been run with lp_solve v1.4 on a HP9000/735.

Execution Times for FDCT.

max cycle fdct	CPU times **IP** solver without module assignment (sec)	CPU times **IP** solver with module assignment (sec)
8	0.9	1.08
9	*	*
10	*	*
11	128.36	92.65
12	101.77	67.98
13	501.37	*
14	230.33	242.13
15	242.84	183.98
16	589.56	727.28
17	852.97	1339.7
18	239.94	81.88
19	449.21	425.08
20	495.88	529.76
21	435.74	434.01
22	2637.5	2496.26
23	853.51	854.26
24	1588.27	1589.82

* Stopped after several hours of CPU time.