Scheduling Algorithms for High Level Synthesis using Integer Linear Programming

Master Thesis
By J.M.W. Hansen

Performed from June 1991 - February 1992
by order of prof. dr. ing. J.A.G. Jess
supervised by ir. M.J.M. Heijligers

The Eindhoven University of Technology is not responsible for the contents of training and thesis reports
Abstract

In high level synthesis the behavioral description of a digital system is transformed to a register transfer level description. One of the tasks in this transformation is scheduling: operations are assigned to controlsteps. For different schedules a different number of resources is needed to implement the system. The problem is to find a schedule that minimizes an objective function, which takes into account the number of resources, the area and power consumption.

One method to perform scheduling is to describe the problem as an integer linear programming (ILP) problem. By solving this ILP problem, the optimal schedule is found according to the objective function.

Three different ILP formulations of the scheduling problem are developed. The difference lies in the interpretation of the binary variables that are used in the ILP formulation and in the tightness of the constraints.

The three formulations were solved with a general (mixed) integer linear program solver based on a simplex method. One of the formulations was solved by an enumeration algorithm. In this algorithm equations were combined to make the constraints as tight as possible.

The algorithms and formulations were tested for a couple of data flow graphs, module sets and time constraints. Only for small scheduling problems the ILP-scheduling method works fast. For bigger problems the computation time rises to hours and days.

The advantage of the ILP method to solve the scheduling problem is the fact that all constraints can be formulated in a very simple way. Extra constraints can simply be added without needing to change the general solving method.

Clearly, this scheduling method is not suitable for interactive use. It can however be used when heuristic scheduling algorithms do not give acceptable results. Further, the results of this scheduler can be used as reference for the quality of heuristic schedulers.
## CONTENTS

1. Introduction ........................................................................................................... 1
   1.1 The design of digital systems ............................................................................. 1
   1.2 High level synthesis ......................................................................................... 2
   1.3 Scheduling ...................................................................................................... 3

2. Formal definition of scheduling ............................................................................. 4
   2.1 Input for the scheduler .................................................................................... 4
      2.1.1 The data flow graph ................................................................................. 4
      2.1.2 The module type set ................................................................................ 5
      2.1.3 The assignment function ........................................................................ 5
      2.1.4 Time constraints ..................................................................................... 6
   2.2 Supplementary definitions ............................................................................... 6
   2.3 The output of the scheduler ............................................................................ 8
   2.4 Scheduling Methods ....................................................................................... 8
   2.5 Extensions to the scheduling problem ............................................................ 10
      2.5.1 Conditional operations and loops ............................................................. 10
      2.5.2 Timing edges .......................................................................................... 11
      2.5.3 Other constraints ................................................................................... 11

3. ILP formulations ................................................................................................... 13
   3.1 Simple ILP model .......................................................................................... 14
   3.2 Binary model .................................................................................................. 15
   3.3 Tighter constraints ......................................................................................... 17
   3.4 Alternative binary model ............................................................................... 20
   3.5 Extensions to the ILP model .......................................................................... 24
      3.5.1 Conditional operations and loops ............................................................. 24
      3.5.2 Timing edges .......................................................................................... 24
      3.5.3 Other constraints ................................................................................... 25

4. ILP solving methods ............................................................................................ 27
   4.1 The enumeration algorithm ............................................................................ 27
   4.2 Module counts ................................................................................................ 29
   4.3 Combining equations ..................................................................................... 32
      4.3.1 Precedence relations .............................................................................. 32
      4.3.2 Module equations ................................................................................... 34

5. Results .................................................................................................................. 36
   5.1 Test conditions ............................................................................................... 36
   5.2 Test results ...................................................................................................... 37
1. Introduction

High level synthesis is an important phase in the design of digital systems. The decisions that are made in this phase have a great impact on the final implementation. Therefore the tasks of high level synthesis have to be performed very accurately. One of these tasks is scheduling which is the subject of this report.

1.1 The design of digital systems

Digital systems can be described at six different levels of abstraction [McFarland88/90]. Going from one level to a level below, is called synthesis. Summarized, the following levels of abstraction can be distinguished:

1. System level,
2. Algorithmic level,
3. Register transfer level,
4. Logic level,
5. Circuit level,
6. Device level

The design of a digital system starts with a system level description. From this description an algorithmic level description is synthesized. Then the register transfer level description is made and so on.

At the design automation section of the Eindhoven University of Technology, tools are developed for the automation of the synthesis of digital systems, in particular for Application Specific Integrated Circuits. One of these tools is a silicon compiler [Stok91], which transforms an abstract behavioral description into a hardware realization. The silicon compiler supports three levels of synthesis: high level synthesis, logic synthesis and layout synthesis. In figure 1 an overview of the silicon compiler is given.

The task of the high level synthesis part is to transform a functional description into a register transfer description. This means going from the algorithmic level to the register transfer level. Therefore, first a data flow graph is derived from the functional description. This graph is then transformed to a global network (a data path) and a controller (finite state machine).

The logic synthesis part is concerned with state assignment of the controller and the production of a network of gates and flipflops. It transforms the register transfer level description to a logic level description.
Finally the layout synthesis part maps the gates and flipflops to the given technology and produces a layout of the chip, the circuit level.

1.2 High level synthesis

We will now concern ourselves with the high level synthesis part. In figure 2 an overview of the tasks of this part is given.

The behavioral description of the system, which means the way the system communicates with the environment, is given in a hardware description language like Lisp, Hardware C, or VHDL. To obtain the data path description and the controller, this description is parsed and transformed to a data flow graph.

The data flow graph is optimized in the sense that dead code is eliminated, constants are propagated, common subexpressions are eliminated, procedures are expanded inline, tree height is reduced, loops are optimized and memory access is optimized.

Further, modules are generated and assigned to operation types. In the next step, operations in the data flow graph are scheduled. This means that operations are assigned to a control step. Finally operations are binded to modules by the allocator.
In this report scheduling is described.

1.3 Scheduling

Scheduling means determining when operations will be executed. When the schedule times of operations are chosen carefully, the number of resources that is needed to implement the system can be minimized.

Because the scheduling problem is known to be NP-hard, many heuristic methods have been developed. These methods generally do not give an optimal solution in terms of a given objective function; only an approximate solution is given. Scheduling, however, is an important task in high level synthesis, so it may be worth while to find an optimal solution.

Therefore a scheduling methods have been developed based on Integer Linear Programming (ILP) ([Gebotys90/91], [Hafer83], [Hwang90/91], [Lee89], [Papachristou90]). With these methods, the optimal schedule in terms of the objective function can be found. Still, the time that is needed to find the optimal schedule, increases exponentially with, for example, the number of operations that have to be scheduled. The problem is now to find an ILP-solving method that is fast for most scheduling problems.
2. Formal definition of scheduling

The task of a scheduler will be described in this chapter by giving the output as function of the input.

2.1 Input for the scheduler

The input of the scheduler consists of a data flow graph, a module type set and an assignment function relating them. Further, the cycle time and the maximum number of cycle steps is specified. All these inputs will be specified in this section.

2.1.1 The data flow graph

The data flow graph describes the behavior of the system that is specified. It specifies which operations have to be executed and in which order this has to be done. The nodes in the graph correspond to operations and directed edges give the relative ordering of the operations: the precedence relations.

**Definition 1:**
A data flow graph is a directed acyclic graph $D_{fg} = (V, E)$ where $V$ is a set of nodes that represent operations and $E$ a set of edges that describe the precedence relations; $E \subseteq V \times V$.

There are a couple of types of operations. The types are defined by the operation type function $\tau_V : V \rightarrow T_V$, where $T_V$ is a set of operation types.

$T_V$ contains the following categories:
1. input/output (for connections with the environment)
2. put/get (for communication with the environment)
3. constant
4. operation (e.g. $+, -, *, /$, user defined operations)
5. array-access (declaration, retrieve, update)

It is possible to extend $T_V$ with operation types (see [Eijndhoven91]), for example to represent conditional statements or loops. However, these operation types are not accounted for in this report.

An important difference between operation types, is that types like ‘input’ and ‘output’ do not represent transformations of data. They only indicate where data comes from or where it is going to. Their impact on the scheduling problem will be explained in § 2.2.
Precedence relations can be divided in categories as well. The categories are defined by the function \( \tau_E : E \rightarrow T_E \), where \( T_E \) is a set of precedence relation types.

\( T_E \) contains the following categories:
1. data (for data transfer)
2. source (for constant enable)
3. chain (ordering of put/get nodes and array-access)

In [Eijndhoven91] also more precedence types are given. They are however not included in this report.

The main difference between the precedence types, is that the types ‘source’ and ‘chain’ only give ordering constraints. The type ‘data’ also represents transfer of data.

### 2.1.2 The module type set

All operations have to be executed by a module. A module is a block that can perform one or more functions. With the implementation of a module, parameters are associated that refer to chip area, power consumption and the number of clock cycles which the module needs to perform certain functions. For scheduling only the type of the modules that are used is interesting.

**Definition 2**: The module type set \( L \) contains module types.

The operation range of a module type is the collection of operation types that can be executed by a module of that type. \( \rho : L \rightarrow \Pi(T_V) \). (\( \Pi(\cdot) \) means the power set of \( \cdot \)).

For every operation type that the module type supports, a delay is determined. This denotes the time a module of that type needs to execute a certain operation type. For every \( m \in L \), the delay is a function \( \delta_m : \rho(m) \rightarrow \mathbb{R} \).

Furthermore, with every module type a cost is associated corresponding to the area that a module of that type occupies and the power it dissipates. The cost is given by the function \( c : L \rightarrow \mathbb{R} \).

### 2.1.3 The assignment function

For each operation it is determined by what kind of module it will be executed. The actual modules that will be used to execute the operations are specified by the allocator. This is called module binding and this is not a task of the scheduler.
The assignment function relating the operations (nodes of the data flow graph) and the module type set is a function \( \alpha : V \to L \), with the property: \( \tau_V(v) \in \rho(\alpha(v)) \).

Note: In the rest of this report it is assumed that all \( m \in L \) are needed to implement the system; the function \( \alpha \) is surjective.

### 2.1.4 Time constraints

For scheduling purposes, the real time is divided in 'timeslots' separated by 'timeslot boundaries' (see figure 3). The length of a timeslot is given by \( T_{cycle} \in \mathbb{R} \). There are \( T_{max} \in \mathbb{N} \) timeslots for scheduling.

The timeslot boundaries are determined by the times: \( k \cdot T_{cycle} \), for all \( k \in \{ 0, 1, \ldots, T_{max} \} \)

The timeslots are determined by the open time interval: \( < k \cdot T_{cycle} , (k + 1) \cdot T_{cycle} > \), for all \( k \in \{ 0, 1, \ldots, T_{max} - 1 \} \)

From now on the timeslots and timeslot boundaries are described with an integer \( k \).

![Figure 3. Numbering of timeslots and timeslot boundaries](image)

### 2.2 Supplementary definitions

In order to describe the output of the scheduler first the following definitions are necessary.
The next short notation will be used to describe the number of cycle steps that the execution of an operation needs. The delay of an operation is a function $\delta_v : V \rightarrow \mathbb{N}$ with 

$$\delta_v(v) = \left[ \frac{\delta_{c(v)}(\pi(v))}{T_{\text{cycle}}} \right] \text{ (} \lceil . \rceil \text{ means round up to the nearest integer.)}$$

The task of a scheduler is to determine at what time operations are executed. Difference is made between operations with positive delay and operations with delay zero:

a. An operation with positive delay is assigned to a timeslot. This timeslot indicates when the operation starts executing. The time that the operation finishes can then easily be determined by using the delay function.

b. An operation with delay zero is assigned to a timeslot boundary. These operations do not transform data; they only indicate where data comes from or where it is going to (e.g. 'input' and 'output')

For both types of operations it is sufficient to give the number of the timeslot or timeslot boundary to which the operation is assigned. The following definition can be used to describe the schedule time for both types of operations:

**Definition 3:**
A schedule is a function $\sigma : V \rightarrow \mathbb{N}$ with the following properties:

a. $\forall v \in V \ (0 \leq \sigma(v) \wedge \sigma(v) + \delta_v(v) \leq T_{\text{max}})$

b. $\forall e \in E \ | \ e=(v_1,v_2) \ \sigma(v_1) + \delta_v(v_1) \leq \sigma(v_2)$

This definition defines a number of functions. These functions form a set of schedule functions.

**Definition 4:**
Define $S$ to be the set of all schedule functions for the given data flow graph, module set, assignment function and time constraints.

The number of resources that is needed to implement the system is given by the distribution function of the modules. The value that the function returns, is the number of modules of a certain type that is busy with the execution of an operation at a certain time.

**Definition 5:**
The number of active modules at a timeslot or timeslot boundary is defined by the function:

$DF_{\sigma} : L \times \mathbb{N} \rightarrow \mathbb{N}$, with

$$DF_{\sigma}(m,t) = \# \{ v \ | \ v \in V \wedge \alpha(v) = m \wedge (\delta_v(v) = 0 \wedge \sigma(v) = t \vee \delta_v(v) > 0 \wedge \sigma(v) \leq t < \sigma(v) + \delta_v(v) ) \}$$
By using this function, it can be calculated how many resources are needed to implement the system that is to be designed. The global cost for implementing the system can then be calculated.

**Definition 6:**
The cost of a schedule is a function $C: S \rightarrow \mathbb{R}$, with

$$C(\sigma) = \sum_{m \in L} c(m) \cdot \max_{0 \leq t \leq T_{\text{max}}} DF_\sigma(m, t).$$

This function is called the objective function.

### 2.3 The output of the scheduler

The output of the scheduler can now be formulated:

Output of the scheduler is the schedule $\sigma \in S$ for which the cost $C(\sigma)$ is minimal.

### 2.4 Scheduling Methods

The general scheduling problem is known to be NP-complete. A lot of heuristic methods have been developed to find solutions close to the optimum.

Basically there are two different scheduling methods: time constrained scheduling and resource constrained scheduling. In time constrained scheduling the maximum number of timeslots is given and the number of modules that are needed has to be minimized according to some weighted sum. In resource constrained scheduling the maximum number of modules is fixed and the number of timeslots that is needed has to be minimized. The scheduling problem as described in the previous sections is time constrained.

Some existing heuristic scheduling methods are the following ([Stok91]):

1. **Critical path scheduling:**
   The critical path is scheduled first. Then the other operations are selected to be scheduled. The selection is based on the execution interval of each operation. This time constrained scheduling method can be found in [Parker86].

2. **List scheduling:**
   For each module type an upper bound is given. All operations for which all predecessors have been scheduled, are put in a list. From this list, operations are selected according to some priority function. This priority function depends on the number of available modules, the number of operations that are executed by a module of the same type and the amount of time that is left. This is a resource
constrained scheduling method. In [Heijligers91] this method is transformed to a
time constrained scheduling method.

3. Force directed scheduling:
For each operation, the impact of scheduling the operation to a certain timeslot on
the objective function is determined. Operations are scheduled iteratively in such a
way that tries to minimize the objective function. In this way time constrained
scheduling is performed. Details can be found in [Heijligers91] and [Paulin89].

4. Transformational scheduling:
Starting with a maximal parallel or serial schedule, operations are shifted to try to
satisfy the constraints and to optimize the objective function. [Nestor90] describes
this method.

5. Neural network scheduling:
With a neural network, the effect of scheduling an operation on the schedule of its
neighbours is modeled, in order to optimize the objective function. See [Hemani90].

All the previous mentioned schedule methods do not guarantee to give the optimum
solution for the problem. However, they generally give good solutions in polynomial
time.

ILP scheduling is a solution method that guarantees that the optimum value is found for
the objective function (but, of course, generally not in polynomial time). The schedule
problem is then formulated as an integer linear programming problem. By solving the
equations, an optimal solution to the scheduling problem is found.

In this report we will consider this scheduling method and try to find an ILP solving
method that is fast for most problems.

Before going further, first a special form of scheduling is described: asap/alap
scheduling. With this form of scheduling no attention is paid to the number of modules
that is needed to implement the system. In asap scheduling, the first time, that an
operation may be active, is calculated. In alap scheduling, the last time, that an operation
may finish, is calculated.

\textbf{Definition 7}:
\textit{asap} and \textit{alap} both are functions $: V \rightarrow \mathbb{N}$, with
\begin{itemize}
  \item[a.] $asap(v) = \min_{\sigma \in S} \sigma(v)$
  \item[b.] $alap(v) = \max_{\sigma \in S} (\sigma(v) + \delta_{\sigma}(v))$
\end{itemize}
Using these definitions it is easily proved that the function \textit{asap} is a schedule. For reasons of notation, instead of the function \textit{asap}, in the rest of this report, the function \textit{first}: \( V \rightarrow \mathbb{N} \) will be used with \( \text{first}(v) = \text{asap}(v) \).

The function \textit{alap} is not a schedule. There is however a schedule that is closely related to \textit{alap}: the function \textit{last}: \( V \rightarrow \mathbb{N} \), with \( \text{last}(v) = \text{alap}(v) - \delta_v(v) \). This function defines the last time that an operation can be scheduled.

The following 2 notations will be used:

1. \( F_v \) is the first timeslot that an operation can be scheduled; \( F_v = \text{first}(v) \).
2. \( L_v \) is the last timeslot that an operation can be scheduled; \( L_v = \text{last}(v) \).

\section*{2.5 Extensions to the scheduling problem}

As mentioned in § 2.1.1, not all possible operation types and precedence relation types that are possible in data flow graphs (according to [Eijndhoven91]) have been considered yet. The scheduling function is not defined yet for conditional operations and loops. Also edges of type ‘timing’ are not accounted for. Further it is possible to account for other constraints and costs.

\subsection*{2.5.1 Conditional operations and loops}

When operations of type branch/merge (for conditional branches) or entry/exit (for loops) are introduced plus edges of type ‘control’, the data flow graph can be divided in basic blocks of operations. These blocks are formed by operations that occur in the same branches of conditional statements and loops.

In particular, operations in different blocks can be ‘mutual exclusive’. For example, a pair of operations is said to be ‘mutual exclusive’, when they occur in different branches of a conditional statement. These operations can never be executed simultaneously. This implies that these operations can be scheduled at the same time and allocated to the same module without increasing hardware cost.

Loops can be regarded as nested conditional operations. Therefore, entry/exit operations can be treated largely in the same way as branch/merge operations.

For more types of mutual exclusion see [Hwang91] and [Stok91]. They also describe ways to include conditional operations and loops in the scheduling problem formulation. Other methods to perform hierarchical scheduling can be found in [Goossens89], [Hwang88/89], [Potkonjak89] and [Wakabayashi89].
2.5.2 Timing edges

Precedence relations of type 'timing' can be basically of three forms: maximum, exact and minimum timing. Assume that \( e \in E \) with \( e = (v_1, v_2) \) and that \( t_E(e) \) = 'timing', then the following constraints should be satisfied:

1. **Maximum timing:**
   Operation \( v_2 \) must start executing maximum \( t_{\text{max}} \) after \( v_1 \) has started:
   \[
   \sigma(v_2) \leq \sigma(v_1) + t_{\text{max}}
   \]

2. **Exact timing:**
   Operation \( v_2 \) must start executing exactly \( t_{\text{ex}} \) after \( v_1 \) has started:
   \[
   \sigma(v_2) = \sigma(v_1) + t_{\text{ex}}
   \]

3. **Minimum timing:**
   Operation \( v_2 \) must start executing minimal \( t_{\text{min}} \) after \( v_1 \) has started:
   \[
   \sigma(v_2) \geq \sigma(v_1) + t_{\text{min}}
   \]

In these three cases, timing edges were defined from the 'finish time' of one operation to the 'start time' of the other. It is also possible to define them from the start time of the first operation to the finish time of the other, or to define them for other combinations. The definitions are easily adapted to these combinations.

2.5.3 Other constraints

There are a number of extra constraints that can be added to the scheduling problem. Examples are:

1. **Hardware constraints:**
   For a certain module type \( m \in L \), a maximum can be given, \( M_{m, \text{max}} \). This constraint can be given by \( DF_{\sigma}(m, t) \leq M_{m, \text{max}} \) for all \( 0 \leq t \leq T_{\text{max}} \).

2. **Time cost:**
   In order to find the shortest schedule for which the objective function is minimal, a time cost factor can be introduced. The last time that all operations have finished is given by \( \max_{v \in V} (\sigma(v) + \delta_v(v)) \).

The problem is now to minimize:

\[
C(\sigma) = \sum_{m \in L} (c(m) \cdot \max_{0 \leq t \leq T_{\text{max}}} DF_{\sigma}(m, t)) + c_{\text{time}} \cdot \max_{v \in V} (\sigma(v) + \delta_v(v)).
\]

When the time cost factor is small enough, the number of modules that are needed is not affected. If \( c_{\text{time}} T_{\text{max}} \) is smaller than the smallest module cost factor, then it is not possible that modules are 'exchanged' for timeslots.
3. Pipelined designs:

If a system has to be able to accept input and to produce output every $d_{in}$ cycles, then the system is called a 'pipeline' with data introduction time $d_{in}$. Probably, more modules will be needed to implement the system. This follows from the distribution function for pipelines:

$$DFP_\sigma(m, t) = \sum_{k=0}^{[(T_{max} - t)/d_{in}]} DFP_\sigma(m, t + k \cdot d_{in}) \text{ for all } m \in L \text{ and } 0 \leq t \leq d_{in} - 1$$

The cost function becomes:

$$CP(\sigma) = \sum_{m \in L} c(m) \cdot \max_{0 \leq t \leq d_{in} - 1} DFP_\sigma(m, t)$$
3. ILP formulations

The goal of an ILP formulation is to describe the scheduling problem as a set of linear inequalities and an objective function. Theory of integer linear programming problems can be found in [Garfinkel172], [Kovács80] and [Taha75].

The task of a scheduler is to find values for $\sigma(v)$ for all $v \in V$. Thus variables are needed to represent $\sigma(v)$ for all $v \in V$. Further, because the cost of a schedule depends on the number of modules that are needed, variables have to be introduced, which describe these module counts.

The data flow graph of figure 4 is used to illustrate the consequences of the different formulations that will be given. The module set that is used is given in figure 5. The assignment function is defined straightforward in figure 6. $T_{\text{max}} = 5$ in the example.

$$X = (a.b + 1) + (b.c + 10)$$

Figure 4. Example data flow graph
\[ L = \{ \text{Inpad, Outpad, Const, Mult, Adder} \}; \]

\[ \rho(\text{Inpad}) = \{ \text{in} \}; \quad \delta_{\text{Inpad}}(\text{in}) = 0; \]
\[ c(\text{Inpad}) = 0; \]

\[ \rho(\text{Outpad}) = \{ \text{out} \}; \quad \delta_{\text{Outpad}}(\text{out}) = 0; \]
\[ c(\text{Outpad}) = 0; \]

\[ \rho(\text{Const}) = \{ 1, 10 \}; \quad \delta_{\text{Const}}(1) = \delta_{\text{Const}}(10) = 0; \]
\[ c(\text{Const}) = 0; \]

\[ \rho(\text{Mult}) = \{ * \}; \quad \delta_{\text{Mult}}(\ast) = 1; \]
\[ c(\text{Mult}) = 1; \]

\[ \rho(\text{Adder}) = \{ + \}; \quad \delta_{\text{Adder}}(+) = 1; \]
\[ c(\text{Adder}) = 1; \]

**Figure 5.** Example module type set

**Figure 6.** Example assignment function

3.1 Simple ILP model

The most straightforward ILP model is to represent \( \sigma(v) \) as a variable \( s_v \) for all \( v \in V \) and to represent the module counts: \( \max_{0 \leq \delta_{\sigma} \leq T_{\text{max}}} DF_\sigma(m, t) \) with \( M_m \) for all \( m \in L \).

From the schedule definitions (Definition 3) it follows:

(S1) \( 0 \leq s_v \leq T_{\text{max}} - \delta_v(v) \)

for all \( v \in V \)

(S2) \( s_{v_1} - s_{v_2} \leq -\delta_v(v_1) \)

for all \( e \in E \) with \( e = (v_1, v_2) \)

(S3) \( s_v \) integer for all \( v \in V \)

The objective function becomes:
In order to calculate $M_m$, the definition of $DF_{\sigma}$ (Definition 5) has to be used. To calculate $DF_{\sigma}$, the number of $s_v$-variables have to be counted, that have a value in a certain range. This is not a linear function of the variables $s_v$. This problem can only be solved by introducing binary variables that describe whether $s_v$ has its value in a specific range.

### 3.2 Binary model

To specify whether $\sigma(v)$ (and thus $s_v$) has value $t$, the following binary variables are introduced:

$$x_{v,t} \in \{0, 1\} \text{ for all } v \in V \text{ and } 0 \leq t \leq T_{\text{max}}.$$ 

$x_{v,t} = 1$, if $\sigma(v) = t$ and $x_{v,t} = 0$ otherwise.

Then also the constraint that $\sigma(v)$ has exactly one value has to be added:

$$\sum_{t=0}^{T_{\text{max}}} x_{v,t} = 1$$

Now $\sigma(v)$ can be expressed in terms of these variables as follows:

$$\sigma(v) = \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v,t}$$

The schedule definition is now transformed into:

(X1) \hspace{1em} 0 \leq \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v,t} \leq T_{\text{max}} - \delta_v(v)

for all $v \in V$  

(X2) \hspace{1em} \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v_1,t} - \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v_2,t} \leq -\delta_v(v_1)

for all $e \in E$ \hspace{1em} e = (v_1, v_2)$

(X3) \hspace{1em} \sum_{t=0}^{T_{\text{max}}} x_{v,t} = 1 \text{ for all } v \in V$

(X4) \hspace{1em} x_{v,t} \in \{0, 1\}, \text{ for all } v \in V \text{ and } 0 \leq t \leq T_{\text{max}}$

The number of modules that is needed to implement the system is given by the following set of inequalities. They are of the form 

$DF_{\sigma}(m, T) \leq M_m$ \text{ for all } 0 \leq T \leq T_{\text{max}}$

(X5) \hspace{1em} \sum_{v \in V \mid \alpha(v) = m \land \delta_v(v) = 0} x_{v, T} + \sum_{v \in V \mid \alpha(v) = m \land \delta_v(v) > 0} \left( \sum_{i=0}^{T-\delta_v(v)+1} x_{v,i} \right) - M_m \leq 0$

for all $m \in L$ \text{ and } 0 \leq T \leq T_{\text{max}}.$
(X6) \( M_m \) integer, for all \( m \in L \).

Finally, the objective function becomes:

\[(XO) \ \text{Minimize} \ \sum_{m \in L} c(m) \cdot M_m \]

The construction for module counts ‘.. \( - M_m \leq 0 \) ’ together with the minimization criterion, takes care that the values of \( M_m \) correspond to the actual module counts.

This ILP formulation is used in [Hafer83], [Hwang90/91], [Lee89] and [Papachristou90]. It contains \( O(|V| \cdot T_{\text{max}}) \) variables and \( O(|V| \cdot T_{\text{max}} + |E| + |L| \cdot T_{\text{max}}) \) constraints.

The number of variables can be reduced by calculating the first and last timeslot that operations can be scheduled (this can be done using asap/alap scheduling). Assume that for operation \( v \) it holds that \( F_v \leq \sigma(v) \leq L_v \), then \( x_{v,t} = 0 \) for all \( t < F_v \) and \( t > L_v \). These values can be substituted.

In that case also the equations (X1) can be omitted because now it holds that:

\[ \sum_{t=F_v}^{L_v} x_{v,t} = 1 \] for all \( v \in V \) and \( x_{v,t} \in \{0, 1\} \) for all \( v \in V \) and \( F_v \leq t \leq L_v \)

And thus:

\[ 0 \leq \text{first}(v) = F_v \leq \sum_{t=F_v}^{L_v} t \cdot x_{v,t} \leq L_v = \text{last}(v) \leq T_{\text{max}} - \delta_v(v) \]

because \( \text{first} \) and \( \text{last} \) are schedules.

The ILP model that arises this way is called the X-model. For the graph given in figure 4, the equations describing the schedule problem are given in here.

Example equations:

Occurrence equations (X3):

\[
\begin{align*}
X_{10} + X_{11} + X_{12} &= 1; \\
X_{20} + X_{21} + X_{22} &= 1; \\
X_{30} + X_{31} + X_{32} &= 1; \\
X_{40} + X_{41} + X_{42} &= 1; \\
X_{50} + X_{51} + X_{52} &= 1; \\
X_{61} + X_{62} + X_{63} &= 1; \\
X_{71} + X_{72} + X_{73} &= 1; \\
X_{82} + X_{83} + X_{84} &= 1; \\
X_{92} + X_{93} + X_{94} &= 1; \\
X_{101} + X_{102} + X_{103} &= 1; \\
X_{111} + X_{112} + X_{113} &= 1;
\end{align*}
\]
Precedence equations (X2):

\begin{align*}
0X10 + 1X11 + 2X12 &- 0X40 - 1X41 - 2X42 <= 0; \\
0X20 + 1X21 + 2X22 &- 0X40 - 1X41 - 2X42 <= 0; \\
0X20 + 1X21 + 2X22 &- 0X50 - 1X51 - 2X52 <= 0; \\
0X30 + 1X31 + 2X32 &- 0X50 - 1X51 - 2X52 <= 0; \\
0X40 + 1X41 + 2X42 - 1X61 - 2X62 - 3X63 &<= -1; \\
0X40 + 1X41 + 2X42 - 1X101 - 2X102 - 3X103 &<= -1; \\
0X50 + 1X51 + 2X52 - 1X71 - 2X72 - 3X73 &<= -1; \\
0X50 + 1X51 + 2X52 - 1X111 - 2X112 - 3X113 &<= -1; \\
1X61 + 2X62 + 3X63 - 2X82 - 3X83 - 4X84 &<= -1; \\
1X71 + 2X72 + 3X73 - 2X82 - 3X83 - 4X84 &<= -1; \\
2X82 + 3X83 + 4X84 - 3X93 - 4X94 - 5X95 &<= -1; \\
1X101 + 2X102 + 3X103 - 1X61 - 2X62 - 3X63 &<= 0; \\
1X111 + 2X112 + 3X131 - 1X71 - 2X72 - 3X73 &<= 0;
\end{align*}

Module count equations (X5):

\begin{align*}
X40 + X50 - Mmult &<= 0; \\
X41 + X51 - Mmult &<= 0; \\
X42 + X52 - Mmult &<= 0; \\
X61 + X71 - Madd &<= 0; \\
X62 + X72 + X82 - Madd &<= 0; \\
X63 + X73 + X83 - Madd &<= 0; \\
X84 - Madd &<= 0;
\end{align*}

Integer constraints (X4) and (X6):

All \( Xij \in \{0,1\}, \text{mult and Madd are integer}; \)

Objective function (X0):

\begin{align*}
\text{Minimize Cost} = Cmult \cdot Mmult + Cadd \cdot Madd.
\end{align*}

Before solving these equations, first the concept of 'tight equations' will be described.

### 3.3 Tighter constraints

Generally, there are many different ways to model a problem with an ILP formulation. There are however differences with respect to the effort necessary to find the optimal solution.

Assume that the solution method is based on first solving the corresponding LP problem. This is the linear programming problem that arises from the ILP problem by releasing the integrality conditions. The constraints now describe a solution space in \( \mathbb{R}^n \) that contains more than only integer points. Depending on the formulation of the ILP problem, the size of the LP solution space is variable, although it still contains the same integer points. Now one formulation is said to be 'tighter' than another, if the corresponding LP solution space is smaller.
Further, when the corresponding LP problem is considered, the constraints describe a multi-dimensional convex polyhedron. Optimal solutions to the LP problem are found on corners of this polyhedron. It is not hard to prove that, for the tightest formulation of an ILP problem, all corners of the polyhedron have to be integral. The optimal solution of the LP problem then corresponds to the optimal solution of the ILP problem.

When this theory is applied to the ILP formulation of the scheduling problem given in the previous section, it can be found that the equations describing the precedence relations are not very tight. In [Gebotys90/91] it is suggested to replace these equations by the following constraints:

\[ (XG) \sum_{i=1}^{\max} x_{v_1,i} + \sum_{i=0}^{T-1+\delta_{e}(v_1)} x_{v_2,i} \leq 1 \]
for all \( e \in E | e = (v_1, v_2) \) and \( 0 \leq T \leq T_{\max} \)

In [Gebotys91] it is proved that these constraints are tighter than the original (X2), although more equations are needed: \( O( (|V| + |E| + |L|) \cdot T_{\max} ) \).

Of course, the reduction of variables and equations mentioned in § 3.2 can be performed:

1. For the equations with \( T > L_{v_1} \), the first part vanishes after the substitution. The remaining equation \( \sum_{i=0}^{T-1+\delta_{e}(v_1)} x_{v_2,i} \leq 1 \) can be left out because equation (X3) is tighter.
2. For the equations with \( T + \delta_{e}(v_1) - 1 < F_{v_2} \), the second part vanishes after substitution. The remaining equation \( \sum_{i=T}^{\max} x_{v_1,i} \leq 1 \) can be omitted as well, for the same reason.

The ILP-formulation that arises from the X-model by replacing equations (X2) by (XG) is called the XG-model. For the example data flow graph, the equations (XG) are given here.

Example equations used in [Gebotys90/91]:

\[
\begin{align*}
X_{11} + X_{12} + X_{40} &\leq 1; \\
X_{12} + X_{40} + X_{41} &\leq 1; \\
X_{21} + X_{22} + X_{40} &\leq 1; \\
X_{22} + X_{40} + X_{41} &\leq 1; \\
X_{21} + X_{22} + X_{50} &\leq 1; \\
X_{22} + X_{50} + X_{51} &\leq 1; \\
X_{31} + X_{32} + X_{50} &\leq 1; \\
X_{32} + X_{50} + X_{51} &\leq 1; \\
X_{41} + X_{42} + X_{61} &\leq 1; \\
X_{42} + X_{61} + X_{62} &\leq 1; \\
X_{41} + X_{42} + X_{101} &\leq 1;
\end{align*}
\]
The idea behind these equations is:

a. If $\sigma(v_1) \geq T$, then for operation $v_2$, which succeeds operation $v_1$, it holds that $\sigma(v_2) \geq T + \delta_{v_1}(v_1)$. In short:

$$\sigma(v_1) \geq T \Rightarrow \sigma(v_2) \geq T + \delta_{v_1}(v_1) \Rightarrow \neg(\sigma(v_2) \leq T + \delta_{v_1}(v_1) - 1)$$

In terms of the variables this gives:

$$\sum_{t=T}^{T_{\text{max}}} x_{v_1,t} = 1 \Rightarrow \sum_{t=0}^{T+\delta_{v_1}(v_1)-1} x_{v_1,t} = 0.$$ 

b. Further, if $\sigma(v_1) < T$, then it is easy to see that:

$$\sum_{t=T}^{T_{\text{max}}} x_{v_1,t} = 0 \Rightarrow \sum_{t=0}^{T+\delta_{v_1}(v_1)-1} x_{v_1,t} \leq 1.$$ 

Together, this leads to the equations given in [Gebotys90/91].

The fact that equations (XG) are tighter than the original equations (X2) describing the precedence relation is illustrated in the following example.

Example of the tightness of the equations (XG):

In the given example it holds:

\[(X3) : x_{61} + x_{62} + x_{63} = 1;
\quad x_{82} + x_{83} + x_{84} = 1;\]

Together with, in the case of the X-model:

\[(X2) : x_{61} + 2x_{62} + 3x_{63} - 2x_{82} - 3x_{83} - 4x_{84} \leq -1;\]

or, in the case of the XG-model:

\[(XG) : x_{62} + x_{63} + x_{82} \leq 1;
\quad x_{63} + x_{82} + x_{83} \leq 1;\]

Suppose that in the algorithm, $x_{62}$ has got the value 1. This means that operation $v_6$ is scheduled to timeslot 2. Now operation $v_8$, which succeeds $v_6$, can not be scheduled to timeslot 2 so it should
follow that $x_82 = 0$;

When $x_62 = 1$ is substituted, it follows from (X3) that $x_61 = 0$ and $x_63 = 0$. When these values are substituted in the precedence equations, we get:

$$(X2') : -2x_82 - 3x_83 - 4x_84 <= -3;$$
or

$$(XG') : x_82 <= 0;$$

$$x_82 + x_83 <= 1;$$

From (X2') nothing can be concluded about the values of $x_82$, $x_83$ or $x_84$. From (XG') it is obvious that $x_82$ is bound to 0. The equations of [Gebotys90/91] thus are tighter.

### 3.4 Alternative binary model

Another way of tightening the precedence constraints is to transform the $x_{v,t}$ variables into a new set of variables. A transformation for which the precedence relations are as tight as possible, is based on the following idea:

All solutions of the equation \( \sum_{t=0}^{T_{\text{max}}} x_{v,t} = 1 \) can be described with $T_{\text{max}} + 1$ vectors of dimension $T_{\text{max}} + 1$. Exactly one ordinate of these vectors is '1'; the other ordinates are '0'. These vectors describe a $T_{\text{max}}$-dimensional space. This space can also be described in the following way:

\[
\begin{pmatrix}
  x_{v,0} \\
x_{v,1} \\
x_{v,2} \\
  \vdots \\
x_{v,T_{\text{max}}-1} \\
x_{v,T_{\text{max}}} \\
\end{pmatrix} = \begin{pmatrix}
  1 \\
  0 \\
  0 \\
  \vdots \\
  0 \\
  0 \\
\end{pmatrix} + \begin{pmatrix}
  -1 \\
  1 \\
  0 \\
  \vdots \\
  0 \\
  0 \\
\end{pmatrix} \cdot \begin{pmatrix}
  \lambda_{v,0} \\
  \lambda_{v,1} \\
  \lambda_{v,2} \\
  \vdots \\
  \lambda_{v,T_{\text{max}}-2} \\
  \lambda_{v,T_{\text{max}}-1} \\
\end{pmatrix}
\]

Now transform the set \( \{x_{v,t}\} \) to the set \( \{\lambda_{v,t}\} \) by taking: \( x_{v,t} = \lambda_{v,t+1} - \lambda_{v,t} \)

From the constraint $x_{v,t} \in \{0, 1\}$ it follows that:

1. $1 = \lambda_{v,t-1} \geq \lambda_{v,0} \geq \cdots \geq \lambda_{v,T_{\text{max}}-1} \geq \lambda_{v,T_{\text{max}}} = 0$ for all $v \in V$ and
2. $\lambda_{v,t}$ integer for all $v \in V$ and $0 \leq t \leq T_{\text{max}}$.

Now $\sigma(v)$ can be expressed in terms of $\{\lambda_{v,t}\}$ as follows:

\[
\sigma(v) = \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v,t} = \sum_{t=0}^{T_{\text{max}}-1} \lambda_{v,t}, \text{ for all } v \in V.
\]
The schedule definition is now:

\[(L1) \quad 0 \leq \sum_{i=0}^{T_{\text{max}}-1} \lambda_{v,i} \leq T_{\text{max}} - \delta_v(v)\]

for all \(v \in V\)

\[(L2) \quad \sum_{i=0}^{T_{\text{max}}-1} \lambda_{v_1,i} - \sum_{i=0}^{T_{\text{max}}-1} \lambda_{v_2,i} \leq -\delta(v_1)\]

for all \(e \in E \setminus e = (v_1, v_2)\)

\[(L3) \quad 1 = \lambda_{v,-1} \geq \lambda_{v,0} \geq \cdots \geq \lambda_{v,T_{\text{max}}-1} \geq \lambda_{v,T_{\text{max}}} = 0\]

for all \(v \in V\)

\[(L4) \quad \lambda_{v,i} \text{ integer}\]

for all \(v \in V\) and \(0 \leq t \leq T_{\text{max}} - 1\).

The equations describing the number of modules become:

\[(L5) \quad \sum_{v \in V \land \alpha(v)=m \land \delta_v(v)=0} (\lambda_{v,T-1} - \lambda_{v,T}) + \sum_{v \in V \land \alpha(v)=m \land \delta_v(v)>0} (\lambda_{v,T-\delta_v(v)} - \lambda_{v,T}) - M_m \leq 0\]

for all \(m \in L\) and \(0 \leq T \leq T_{\text{max}}\).

\[(L6) \quad M_m \text{ integer for all } m \in L\]

Finally the objective function:

\[(L0) \quad \text{Minimize } \sum_{m \in L} c(m) \cdot M_m\]

For the example data flow graph the resulting equations are as follows:

Occurrence equations (L3):

1 >= L10 >= L11 >= 0;
1 >= L20 >= L21 >= 0;
1 >= L30 >= L31 >= 0;
1 >= L40 >= L41 >= 0;
1 >= L50 >= L51 >= 0;
1 >= L60 >= L61 >= 0;
1 >= L70 >= L71 >= 0;
1 >= L80 >= L81 >= 0;
1 >= L90 >= L91 >= 0;
1 >= L100 >= L101 >= 0;
1 >= L110 >= L111 >= 0;
Precedence equations (L2):

\[ \begin{align*}
L_{10} + L_{11} - L_{40} - L_{41} & \leq 0; \\
L_{20} + L_{21} - L_{40} - L_{41} & \leq 0; \\
L_{20} + L_{21} - L_{50} - L_{51} & \leq 0; \\
L_{30} + L_{31} - L_{50} - L_{51} & \leq 0; \\
L_{40} + L_{41} - 1 - L_{61} - L_{62} & \leq -1; \\
L_{40} + L_{41} - 1 - L_{101} - L_{102} & \leq -1; \\
L_{50} + L_{51} - 1 - L_{71} - L_{72} & \leq -1; \\
L_{50} + L_{51} - 1 - L_{111} - L_{112} & \leq -1; \\
1 + L_{61} + L_{62} - 2 - L_{82} - L_{83} & \leq -1; \\
1 + L_{71} + L_{72} - 2 - L_{82} - L_{83} & \leq -1; \\
2 + L_{82} + L_{83} - 3 - L_{93} - L_{94} & \leq -1; \\
L_{101} + L_{102} - 1 - L_{61} - L_{62} & \leq 0; \\
L_{111} + L_{112} - 1 - L_{71} - L_{72} & \leq 0;
\end{align*} \]

Module count equations (L5):

\[ \begin{align*}
1 - L_{40} + 1 - L_{50} - M_{\text{mult}} & \leq 0; \\
L_{40} - L_{41} + L_{50} - L_{51} - M_{\text{mult}} & \leq 0; \\
L_{41} + L_{51} - M_{\text{mult}} & \leq 0; \\
1 - L_{61} + 1 - L_{71} & \text{ M}_{\text{add}} \leq 0; \\
L_{61} + L_{62} + L_{71} - L_{72} + 1 - L_{82} & \text{ M}_{\text{add}} \leq 0; \\
L_{62} + L_{72} + L_{82} & \text{ M}_{\text{add}} \leq 0; \\
L_{83} & \text{ M}_{\text{add}} \leq 0;
\end{align*} \]

Integer constraints (L4) and (L6):

\( \text{L}_{ij} \) integer for all \( ij \)

FCMmult, Madd integer.

Objective function (LO):

Minimize \( \text{Cost} = \text{C}_{\text{mult}} \cdot \text{M}_{\text{mult}} + \text{C}_{\text{add}} \cdot \text{M}_{\text{add}} \)

In this formulation the equations describing the precedence relations are not yet as tight as possible. This can be achieved by first calculating \( F_v \) and \( L_v \) as defined in § 2.2. Then it follows that: \( \lambda_{v,t} = 1 \) for \( t \leq F_v - 1 \) and \( \lambda_{v,t} = 0 \) for \( t \geq L_v \).

Also (L1) can be omitted then, because \( L_v \leq T_{\text{max}} - \delta_{v}(v) \) for each \( v \in V \).

Second, corresponding to the equations of [Gebotys90/91], the following equations are used to replace equations (L2):

\[ \begin{align*}
\lambda_{v_2,T+\delta_{v}(v_1)} - \lambda_{v_1,T} & \geq 0 \\
& \text{for all} \ e \in E \ e = (v_1,v_2) \ \text{and} \ -1 \leq T \leq T_{\text{max}} - \delta_{v}(v_1)
\end{align*} \]

For the example in figure 4, the resulting equations is given here. The ILP formulation in this section is called the L-model.
Example (LG) equations:

Precedence relations (LG):

\[
\begin{align*}
L_{40} - L_{10} & \geq 0; \\
L_{41} - L_{11} & \geq 0; \\
L_{40} - L_{20} & \geq 0; \\
L_{41} - L_{21} & \geq 0; \\
L_{50} - L_{20} & \geq 0; \\
L_{51} - L_{21} & \geq 0; \\
L_{50} - L_{30} & \geq 0; \\
L_{51} - L_{31} & \geq 0; \\
L_{61} - L_{40} & \geq 0; \\
L_{62} - L_{41} & \geq 0; \\
L_{101} - L_{40} & \geq 0; \\
L_{102} - L_{41} & \geq 0; \\
L_{71} - L_{50} & \geq 0; \\
L_{72} - L_{51} & \geq 0; \\
L_{111} - L_{50} & \geq 0; \\
L_{112} - L_{51} & \geq 0; \\
L_{82} - L_{61} & \geq 0; \\
L_{83} - L_{62} & \geq 0; \\
L_{82} - L_{71} & \geq 0; \\
L_{83} - L_{72} & \geq 0; \\
L_{94} - L_{83} & \geq 0; \\
L_{95} - L_{84} & \geq 0; \\
L_{61} - L_{101} & \geq 0; \\
L_{62} - L_{102} & \geq 0; \\
L_{71} - L_{111} & \geq 0; \\
L_{72} - L_{112} & \geq 0;
\end{align*}
\]

For this formulation the equations describing the precedence relations are as tight as possible. This is illustrated examples that follows here. The number of variables that is needed is \(O(|V| \cdot T_{\text{max}})\), the number of equations is \(O( (|V| + |L| + |E|) \cdot T_{\text{max}})\).

Example of the tightness of the L-model:

In the given example it holds:

\[
(L3) \quad 1 \geq L_{61} \geq L_{62} \geq 0; \\
\quad 1 \geq L_{82} \geq L_{83} \geq 0;
\]

And

\[
(L2) \quad 1 + L_{61} + L_{62} - 2 - L_{82} - L_{83} \leq -1;
\]

or

\[
(LG) \quad L_{82} - L_{61} \geq 0; \\
\quad L_{83} - L_{62} \geq 0;
\]

Suppose in the algorithm operation \(L_{61}\) has got value 1. This means that operation \(v_6\) is scheduled after timeslot 1. The consequence is that operation \(v_8\) is scheduled after timeslot 2. Thus \(L_{82}\) should get value 1.

When \(L_{61} = 1\) is substituted in the equations (L2) and (LG), we get:

\[
(L2') \quad L_{62} - L_{82} - L_{83} \leq -1
\]
\[(LG') : \begin{align*}
L_{82} & \geq 1; \\
L_{83} - L_{62} & \geq 0;
\end{align*}\]

From (L2') nothing can be concluded about the values of \(L_{62}, L_{82}\) or \(L_{83}\). From (LG') it follows that \(L_{82}\) is bound to value 1. The equations (LG) are thus tighter than (L2).

These equations are even tighter than those of [Gebotys90/91]. The equations that are used there, are:

\[(XG) : \begin{align*}
x_{62} + x_{63} + x_{82} & \leq 1; \\
x_{63} + x_{82} + x_{83} & \leq 1;
\end{align*}\]

If \(x_{61}\) has got value 0, then operation \(v_6\) is scheduled after timeslot 1 and operation \(v_8\) should be scheduled after timeslot 2: \(x_{82} = 0\). This can not be concluded from equations (XG). However, it can be concluded from the formulation (LG) as was mentioned above. Thus the alternative model is tighter for the precedence relations.

### 3.5 Extensions to the ILP model

The extensions to the scheduling model as described in § 2.5 can also be modeled in ILP equations.

#### 3.5.1 Conditional operations and loops

In order to schedule conditional branches and loops, extra variables are needed to describe the values of the distribution function for each block. The rest of the formulation then follows straightforward from the definitions of these distribution functions. See [Hwang91] for details.

#### 3.5.2 Timing edges

Assume that \(e \in E\) with \(e = (v_1, v_2)\) and \(\tau_e(v) = \text{\textquoteleft timing}\), then the three types of timing edges can be described as follows:

1. **Maximum timing:**
   a. In the X-model:
      \[
      \sum_{t=0}^{T_{max}} t \cdot x_{v_1,t} - \sum_{t=0}^{T_{max}} t \cdot x_{v_2,t} \geq -t_{max}
      \]
   b. In the XG-model:
      \[
      \sum_{t=T}^{T_{max}} x_{v_2,t} + \sum_{t=0}^{T_{max}-1} x_{v_1,t} \leq 1 \text{ for all } (v_1, v_2) \in E \text{ and } 0 \leq T \leq T_{max}
      \]
   c. In the L-model:
      \[
      \lambda_{v_1,t-t_{max}} - \lambda_{v_2,t} \geq 0 \text{ for all } t_{max} \leq t \leq T_{max} - 1
      \]
2. **Exact timing:**
a. In the X-model:
\[ x_{v_1,t} = x_{v_2,t+t_{ex}} \] for all \( 0 \leq t \leq T_{\text{max}} - t_{ex} \)
b. In the XG-model:
\[ x_{v_1,t} = x_{v_2,t+t_{ex}} \] for all \( 0 \leq t \leq T_{\text{max}} - t_{ex} \)
c. In the L-model:
\[ \lambda_{v_1,t} = \lambda_{v_2,t+t_{ex}} \] for all \( 0 \leq t \leq T_{\text{max}} - 1 - t_{ex} \)

3. Minimum timing:
   a. In the X-model:
   \[
   \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v_1,t} - \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v_2,t} \geq -t_{\text{min}}
   \]
   b. In the XG-model:
   \[
   \sum_{t=0}^{T_{\text{max}}} x_{v_1,t} + \sum_{t=0}^{T_{\text{max}}-t_\text{min}} x_{v_2,t} \leq 1 \] for all \( e = (v_1, v_2) \in E \) and \( 0 \leq T \leq T_{\text{max}} \)
   c. In the L-model:
   \[
   \lambda_{v_2,t+t_{\text{min}}} - \lambda_{v_1,t} \geq 0 \] for all \( 0 \leq t \leq T_{\text{max}} - 1 - t_{\text{min}} \)

3.5.3 Other constraints

The following constraints can be added:

1. Hardware constraints:
   When the maximum number for a certain module type \( m \in L \) is given by \( M_{m,\text{max}} \),
   the additional constraint is: \( M_m \leq M_{m,\text{max}} \)

2. Time cost:
   When the time cost factor is given by \( c_{\text{time}} \), the variable \( T_I \) is introduced. \( T_I \)
   describes the time that the last operation finishes. The following constraints are added:
   a. In the X-model:
   \[
   \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v,t} - T_I \leq -\delta_v(v) \] for all \( v \in V \)
   b. In the XG-model:
   \[
   \sum_{t=0}^{T_{\text{max}}} t \cdot x_{v,t} - T_I \leq -\delta_v(v) \] for all \( v \in V \)
   c. In the L-model:
   \[
   \sum_{t=0}^{T_{\text{max}}-1} t \cdot \lambda_{v,t} - T_I \leq -\delta_v(v) \] for all \( v \in V \)

   The objective function is changed into:
   \[
   \text{Minimize } \sum_{m \in L} c(m) \cdot M_m + c_{\text{time}} \cdot T_I
   \]
3. Pipelined designs:
   When the data introduction time is $d_{in}$, the equations referring to the module counts are changed into:
   a. In the X-model and XG-model:
      \[
      (XP) \quad \sum_{k=0}^{[(T_{max}-k)/d_{in}]} \left( \sum_{v \in V} \right) x_{v,T+k,d_m} + \\
      \sum_{v \in V} \left( \sum_{i=T+k,d_m}^{T+k-l} x_{v,i} \right) - M_m \leq 0
      \]
      for all $m \in L$ and $0 \leq T \leq l - 1$.
      This equation is used to replace equation (X5).
   b. In the L-model:
      \[
      (LP) \quad \sum_{k=0}^{[(T_{max}-k)/d_{in}]} \left( \sum_{v \in V} \right) (\lambda_{v,T-1+k,d_m} - \lambda_{v,T+k,d_m}) + \\
      \sum_{v \in V} (\lambda_{v,T-\delta(v)+k,d_m} - \lambda_{v,T+k,d_m}) - M_m \leq 0
      \]
      for all $m \in L$ and $0 \leq T \leq d_{in} - 1$.
      This equation is used to replace equation (L5).
   The objective function does not change.
4. ILP solving methods

There are two fundamental different ways to solve Integer Linear Programming (ILP) problems ([Garfinkel79], [Kovács80], [Taha75]). One is to solve the corresponding LP problem and to try to find integer solutions close to the LP optimum. The other is to enumerate possible solutions and to select the best according to the objective function.

For the first method an existing algorithm was used. This algorithm was designed to solve arbitrary (Mixed) Integer Linear Programming problems. By using this algorithm there is no way to account for the special structure of the ILP problem used for scheduling. The only part that can be changed is the input: the formulation of the problem. There is no other way to influence the speed of the algorithm.

Therefore, also an enumeration algorithm was developed. This algorithm will be described in more detail in this chapter.

4.1 The enumeration algorithm

The principle of enumeration is to test all possible solutions and to select the best according to the objective function. This can be done in a recursive way.

The enumeration method that is used for the scheduling problem is given in figure 7. As ILP formulation the X-model is used.

First a value of the objective function (the cost) is selected. Then all possible combinations of values of $M_m$ for which $\sum_{m \in L} c(m) \cdot M_m = \text{cost}$ are enumerated. For each combination it is checked whether there exists a solution for the $x_{v,t}$ variables. This order of enumeration is used, because only the values of $M_m$ affect the cost directly.

The check for the existence of a solution for the $x_{v,t}$ variables can be regarded as a test for the combination of values of the $M_m$ variables. This test is performed by the function 'Solution_exists()'.

In the function 'Solution_exists()', the function 'Consequences()' is called. This function substitutes the variables that have got a value. Further it checks whether some variables are bound to values.

For example, if in the equation $\sum_{t=0}^{T_{\text{max}}} x_{v,t} = 1$ the variable $x_{v,t_1}$ has value 1, then all the other
main()
|
Input: Dfb, L, α, Tcycle, T_max;
Generate_equations();
double Objective_value = ∞;

// The following program part will be specified later.
double Cost;
while ( Objective_value not optimal )
{
    Cost = Select_Cost_value(); // such that Cost < Objective_value
    for ( all M_m | \sum_m c(m) M_m == Cost )
    {
        Substitute_M_m(); // in the equations.
        if ( Solution_exists() ) // search a solution for the x_v variables
        {
            Objective_value = Cost;
            Write_intermediate_solution();
        }
    }

    // Last found schedule was optimal
    Output="#Last Schedule#
    }

int Solution_exists()
{
    // OL and O2 are ordered lists of x_v variables.
    // All variables that have a value are put in Ol, the others in O2.
    // Initially Ol is empty and O2 contains all variables.
    Variable *v;
    int possible = Consequences();
    while (O2 ≠ ∅) // possible,
    {
        v = Select_to_assign();
        Remove(v, O2);
        Append(v, Tail(Ol));
        while (Consequences() == FALSE && possible)
        {
            for : v = Tail(Ol) v && v-Bound: v = Tail(Ol))
            {
                Remove(v, Tail(Ol));
                Append(v, O2);
            }
            if (O1 ≠ ∅)
            {
                v = Select_to_complement();
                Remove(v, O1);
                Append(v, Tail(Ol));
            }
            else
            {
                possible = FALSE;
            }
        }
    }

    return possible;
}

Figure 7. Enumeration algorithm
variables are bound to 0. These values are substituted in the equations as well.

If this does not lead to contradictions in the sense that variables are bound to 0 by one equation and bound to 1 by another equation at the same time, then the function returns TRUE, else the function returns FALSE.

The speed of the algorithm now mainly depends on the number of times the test 'Solution_exists()' is performed, and on the way that variables are bound to values.

The issue is now to find a way of assigning values to $M_m$, in such a way that as little tests as possible have to be performed. Further an efficient method of bounding variables has to be developed.

4.2 Module counts

The enumeration algorithm starts with choosing a value for the cost. In order to bound the search space for the cost, first an upper and lower bound for the cost are determined. Because the cost only depends on the values of $M_m$, first upper and lower bound are determined for these values.

The upper bound for $M_m$ is taken to be the number of operations that are assigned to the module: $MaxM_m = \{v \in V \mid \alpha(v) = m\}$

The lower bound for $M_m$ is the minimum value for which there exists a solution for the other module variables and the $x_{vl}$ variables. This minimum $(MinM_m)$ can be determined as follows:

Assign to $M_m$ for all $m \in L$ with $m \neq m_1$ the maximum value $MaxM_m$. $M_{m_1}$ is chosen as variable. The test 'Solution_exists()' now only depends on the value of $M_{m_1}$.

It holds that:

a. If the test fails for $M_{m_1} = a$, then the test will fail for all $M_m \leq a$, because: if a schedule is impossible when at most $a$ modules of a certain type can be used, the schedule will never be possible if less than $a$ modules can be used.

b. If the test succeeds for $M_{m_1} = a$, then the test will succeed for all $M_m \geq a$, because: if scheduling is possible when $a$ modules can be used, then scheduling will be certainly possible if more than $a$ modules can be used.
Further, the test will certainly fail for $M_{m_1} = 0$ and the test will certainly succeed when $M_{m_1} = MaxM_{m_1}$.

It can thus be concluded that the lower bound of $M_{m_1}$ can be found by using binary search in the $M_{m_1}$-space, with starting bounds 0 and $MaxM_{m_1}$.

When all maxima and minima are known, an upper and a lower bound can be given on the optimum value of the objective function:

$$\sum_{m \in L} c(m) \cdot MinM_m \leq \text{Optimal cost} \leq \sum_{m \in L} c(m) \cdot MaxM_m$$

The next step is to select values for the cost.

The selection for the values for the cost is performed with linear search, starting at the lower bound that was found. The first time there is a combination for the $M_m$ variables that gives this cost and for which the test was successful, the minimum value for the cost is found. The corresponding schedule is the optimal schedule. (see figure 8).

```c
double cost = \sum_{m \in L} c(m) \cdot MinM_m - 1;
int found = FALSE;
while (! found )
{
    ++cost;
    for( all $M_m : \sum_{m \in L} c(m) \cdot M_m == cost$ )
    {
        Substitute_M_m();
        if (Solution_exists())
        {
            Write_solution();
            found = TRUE;
            break;
        }
    }
}
```

**Figure 8.** Linear search algorithm in the cost space

Of course, linear search is expensive in terms of calculation time. In general it can, however, not be avoided. In the range of possible values for the objective function, there is no regularity in the way the tests depend on the value of the cost.

If, however, the costs of the module types are all integer multiples of the smallest module type cost, then the search for the minimum can be simplified by using binary search in the cost space. This can be seen as follows:
Introduce the notation $c'(m) = \frac{c(m)}{\min_{m_1 \in L} c(m_1)}$ for all $m \in L$. Now for all $m \in L$, $c'(m)$ is integer. Further, there is at least one $m \in L$, say $m_1$, for which $c'(m) = 1$.

The object function is changed into:

Minimize $\sum_{m \in L} c'(m) \cdot M_m$.

Of course the minimum for this object function is reached for the same values of $M_m$ as the minimum for (XO). The value of this object function is always integer.

Assume that for a certain combination of values for $M_m$ the test 'Solution_exists()' succeeds and $\sum_{m \in L} c(m) \cdot M_m = c$. Then for all $c_1 \geq c$ there exists a combination of values $M_m$ that gives cost $c_1$ and for which the test 'Solution_exists()' is successful. This combination can be found by increasing $M_m$, with $c_1 - c$.

This implies that for every cost that is bigger than the asked minimum, there is a combination of values for which the test is successful. Thus binary search in the cost space can be used (see figure 9).

So, by first performing a check whether the module type costs satisfy the mentioned condition, the algorithm will be faster for special cases of the scheduling problem.
\[ c'(m) = \frac{c(m)}{\text{min}_{m \in L} c(m)} \]

// Precondition to use this algorithm is, that \( c'(m) \) is integer for all \( m \in L \)

```c
int LB = \sum_{m \in L} c'(m) \cdot \text{Min}_m; - 1;
int UB = \sum_{m \in L} c'(m) \cdot \text{Max}_m;
int Cost;
int found;

while ( LB + 1 < UB )
{
    Cost = (LB + UB) / 2;
    for ( all \( M_m \) \| \sum_{m \in L} c(m) \cdot M_m = Cost )
    {
        if (Solution_exists())
        {
            found = TRUE;
        }
    }
    if (found == TRUE)
    {
        UB = Cost;
    }
    else
    {
        LB = Cost;
    }
}
```

Figure 9. Binary search in the cost space

### 4.3 Combining equations

The use of a general ILP solving method to solve the equations of the scheduling problem, is not very wise. In the enumeration algorithm it can be accounted for the special structure of the problem by combining equations at the appropriate moment. The idea is: if a collection of linear inequalities holds for certain values of \( x_{v,t} \), then a linear combination of them should hold as well.

#### 4.3.1 Precedence relations

The first step is to combine occurrence relations with precedence relations. This is illustrated the following example:

Example of combining the precedence equations:

In the example of chapter 3 it holds:

\[
(X3) : X61 + X62 + X63 = 1; \quad X82 + X83 + X84 = 1;
\]

And

\[
(X2) : X61 + 2X62 + 3X63 - 2X82 - 3X83 - 4X84 <= -1;
\]
Suppose that in the algorithm, \( X_{62} \) has got the value 1. This means that operation \( v_6 \) is scheduled to timeslot 2. Now operation \( v_8 \), which succeeds \( v_6 \), can not be scheduled to timeslot 2 so it should follow that \( X_{82} = 0 \);

When \( X_{62} = 1 \) is substituted, it follows from (X3) that \( X_{61} = 0 \) and \( X_{63} = 0 \). When these values is substituted in the precedence equation we get:

\[
(X2') : -2X_{82} - 3X_{83} - 4X_{84} \leq -3;
\]

From (X2') nothing can be concluded about the values of \( X_{82}, X_{83} \) or \( X_{84} \). However when equation (X3) is added to this equation 4 times, then the equation becomes:

\[
(X2'') : 2X_{82} + X_{83} + 0X_{84} \leq 1;
\]

From this equation it follows that \( X_{82} \) is bound to 0, because if \( X_{82} \) has value 1, then constraint (X2'') becomes: \( 'X_{83} \leq -1' \) which can never be satisfied.

In general, if, at a certain point in the algorithm it holds that:

1. \( \sum_{t=F_{v_1}}^{L'_{v_1}} x_{v_1,t} = 1 \)
2. \( \sum_{t=F_{v_2}}^{L'_{v_2}} x_{v_2,t} = 1 \)
3. \( \sum_{t=F_{v_1}}^{L'_{v_1}} t \cdot x_{v_1,t} - \sum_{t=F_{v_2}}^{L'_{v_2}} t \cdot x_{v_2,t} \leq -\delta_v(v_1) \)

with \( F_{v_1} \leq F'_{v_1} \leq L'_{v_1} \leq L_{v_1} \), and \( F_{v_2} \leq F'_{v_2} \leq L'_{v_2} \leq L_{v_2} \).

And assume that \( L'_{v_1} + \delta_v(v_1) > L'_{v_2} \) and \( F'_{v_1} + \delta_v(v_1) > F'_{v_2} \) then it should follow that:

a. \( x_{v_2,t} = 0 \) for \( F'_{v_2} \leq t < F'_{v_1} + \delta_v(v_1) \), because, if operation \( v_2 \) is scheduled before \( F'_{v_1} + \delta_v(v_1) \), then operation \( v_1 \) cannot be scheduled \( \delta_v(v_1) \) steps earlier.

b. \( x_{v_1,t} = 0 \) for \( L'_{v_2} - \delta_v(v_1) < t \leq L'_{v_1} \), because, if operation \( v_1 \) is scheduled after \( L'_{v_2} - \delta_v(v_1) \), then operation \( v_2 \) cannot be scheduled \( \delta_v(v_1) \) steps later.

These facts can only be found when equations 1, 2, and 3 are combined. The idea is to add equations 1, 2 and 3 linearly. We then get the equation \( (4) = a \cdot (1) + b \cdot (2) + c \cdot (3) \) with \( c = 1 \).

4. \( \sum_{t=F_{v_1}}^{L'_{v_1}} (t-a) \cdot x_{v_1,t} - \sum_{t=F_{v_2}}^{L'_{v_2}} (t-b) \cdot x_{v_2,t} \leq -a + b - \delta_v(v_1) \)

for arbitrary \( a \) and \( b \).

If \( a \) is chosen to be \( F'_{v_1} \) and \( b \) to be \( L'_{v_2} \), then all coefficients of the variables in equation 4 are positive. The right-hand value equals: \( -F'_{v_1} + L'_{v_2} - \delta_v(v_1) \)
Now for \( L'_y - \delta_s(v_1) < t \leq L'_y \) the coefficient of \( x_{v_1,t} \) equals \( t - F'_{v_1} \). Assume that \( x_{v_1,t} = 1 \), then, since all coefficients are positive, the minimum value of the left-hand side is reached, when all other variables have value 0. The constraint cannot be satisfied because the left-hand value, \( t - F'_{v_1} \), is bigger than the right-hand value, \(-F'_{v_1} + L'_y - \delta_s(v_1)\). By consequence \( x_{v_1,t} \) is bound to value 0.

In the same way the variables \( x_{v_2,t} \) are bound to 0 for \( F'_{v_2} \leq t < F'_{v_2} + \delta_s(v_1) \).

It is important to note, that it was concluded that the mentioned variables were bound to 0, after examining only equation 4.

In fact, the actual constraints that were combined are:

\[
a \cdot \sum_{t=F_{v_1}}^{L_{v_1}} x_{o_1,t} + b \cdot \sum_{t=F_{v_2}}^{L_{v_2}} x_{o_2,t} + (\sum_{t=F_{v_1}}^{L_{v_1}} t \cdot x_{v_1,t} - \sum_{t=F_{v_2}}^{L_{v_2}} t \cdot x_{v_2,t}) \leq -a + b - \delta_s(v_1)
\]

This can be used within the function 'Consequences' where it is determined whether some variables are bound to values. The equations are updated dynamically. The coefficient \( a \) is determined by the smallest \( t \) for which \( x_{v_1,t} \) has no value yet. Coefficient \( b \) is determined by the biggest \( t \) for which \( x_{v_2,t} \) has no value yet.

In this way the precedence relations are as tight as possible: Every recursion step maximal information is retrieved from these equations by combining them. It is thus not necessary to use another formulation for the scheduling problem.

It can be argued that these combined constraints could be added all to the first formulation of the problem. The reason that this is not useful, is that there are \( O(T_{\text{max}}^2) \) of these constraints. Many of them are superfluous but they will increase the calculation time.

### 4.3.2 Module equations

The second step is to combine the equations describing the module counts with the occurrence equations. The idea will be described by the following example.

Example of combining the module equations:

In the example of chapter 3 it holds that:

\[
\begin{align*}
(X3) : & \quad X61 + X62 + X63 = 1; \\
& \quad X71 + X72 + X73 = 1; \\
& \quad X82 + X83 + X84 = 1; \\
(X5) : & \quad X61 + X71 - \text{Madd} <= 0; \\
& \quad X62 + X72 + X82 - \text{Madd} <= 0; \\
& \quad X63 + X73 + X83 - \text{Madd} <= 0; \\
& \quad X84 - \text{Madd} <= 0;
\end{align*}
\]
And assume that in the algorithm the following choices were made: Madd = 1 and X61 = X71 = X84 = 0.

When these values are substituted, the equations become:

\[(X3') : X62 + X63 = 1;
X72 + X73 = 1;
X82 + X83 = 1;
(X5') : X62 + X72 + X82 <= 1;
X63 + X73 + X83 <= 1;
\]

The two other equations become superfluous.

These constraints cannot be satisfied: three operations should share two modules. This can be concluded by adding the module equations \((X5')\) and by substituting the occurrence equations \((X3')\):

adding : \(X62 + X63 + X72 + X73 + X82 + X83 <= 2;\)
substituting : \(3 <= 2; IMPOSSIBLE!\)

It is important to note that not all equations of \((X5)\) that refer to Madd are added. This is an essential condition for the use of this test.

In general, combining module equations with occurrence equations is performed as follows:

First, all equations, that are associated with the same module and that are not all substituted, are added. Then all occurrence relations are subtracted in such a way that the left-hand side vanishes. If the number on the right-hand side is smaller than 0, the constraint is not satisfiable, and further trying is useless.

Clearly, this test is only used to determine whether further search is useful or not. In the algorithm this test is performed within the function ‘Consequences()’ just after all variables have been substituted and just before any bounding takes place.

It can be questioned why no such equations were added to the first formulation. The reason is that the equations are combined after the variables, that have a value, are substituted. In some equations all variables are substituted. These equations are not considered in this test. This means that only some of the equations associated with a certain module are combined. To add all possible combinations means to add \(O(\mid L\mid 2^{max})\) equations of which only a small part will be considered.

The idea about combining module equations is: the complexity of the test ‘Solution_exists()’ increases with \(O(\mid L\mid T_{max})\), but the number of times this test is performed is reduced with \(O(2^{\mid V\mid T_{max}})\).
5. Results

In order to test the algorithms, they were implemented in C++, and run on a HP-750.

5.1 Test conditions

The input of the scheduler is processed with the neat++-package. From the data flow graph the equations were derived in a straightforward way.

To solve the equations with the general MILP-solver, they are written to a file that is input to this solver. The output of this program was interpreted by the scheduler and the optimal schedule is output. The three different formulations X-model, XG-model and L-model were tested.

To solve the equations with the enumeration algorithm, an internal database containing all variables and equations is produced. Within this database the equations can be manipulated. For this algorithm only the X-model was used as explained in the previous chapter.

The algorithms were tested for a number of data flow graphs, with different module sets and values of $T_{\text{max}}$.

The most important criterion for testing is the speed of the algorithm, since every algorithm will give an optimal solution in terms of the objective function.

The relevant characteristics of the module sets are as follows:

a. Module set 1 contains an adder and a multiplier with the same cost. The delay of adder is 1 cycle and the delay of the multiplier is 2 cycles.

b. Module set 2 contains an adder and a multiplier with the same cost. Both the adder and the multiplier have delay 1 cycle.

c. Module set 3 contains an ALU that can be used as an adder and as a multiplier. The delay for all operations is 1.

More information on the graphs can be found in appendix A and in [Heijligers91]. Equivalent test results can be found in [Heijligers91].
5.2 Test results

In tables 1, 2 and 3 the CPU time is given for the various algorithms, formulations, data flow graphs, values of $T_{\text{max}}$, and module sets.

Of the results of equivalent tests in [Heijligers91], it can be concluded that list scheduling needs a few seconds for smaller examples up to at most about 40 seconds for bigger examples. This algorithm was implemented in Lucid Common LISP and run on Apollo DN 3000.

The number of modules that is needed for every data flow graph is given in appendix A.

<table>
<thead>
<tr>
<th>$D_{\text{fg}}$</th>
<th>$T_{\text{max}}$</th>
<th>Algorithm</th>
<th>MILP-solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Enumeration</td>
<td>X-model</td>
</tr>
<tr>
<td>$f2$</td>
<td>8</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{fdct}$</td>
<td>8</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{mwd}$</td>
<td>65</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{ptseng}$</td>
<td>5</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>$s2$</td>
<td>10</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\text{xd3}$</td>
<td>17</td>
<td>1.2</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

5.3 Conclusions

From the tables it can be concluded that the ILP scheduling method is only running fast for the smallest possible values of $T_{\text{max}}$. For larger values, the computation time rises to hours and days.

Because not many exact computation times are known, only the following prudent observations can be made:
TABLE 2. CPU times in seconds for scheduling, using module set 2

<table>
<thead>
<tr>
<th>Dfg</th>
<th>$T_{\text{max}}$</th>
<th>Algorithm</th>
<th>MILP-solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Enumeration</td>
<td>X-model</td>
</tr>
<tr>
<td>f2</td>
<td>6</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1.1</td>
<td>749.4</td>
</tr>
<tr>
<td>fdct</td>
<td>6</td>
<td>1.3</td>
<td>222.4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1630.0</td>
<td>*</td>
</tr>
<tr>
<td>mwd</td>
<td>53</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>ptseng</td>
<td>4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>s2</td>
<td>9</td>
<td>1.4</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1.0</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

A table entry containing a '*' means: time > 1 day and stopped.

TABLE 3. CPU times in seconds for scheduling, using module set 3

<table>
<thead>
<tr>
<th>Dfg</th>
<th>$T_{\text{max}}$</th>
<th>Algorithm</th>
<th>MILP-solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Enumeration</td>
<td>X-model</td>
</tr>
<tr>
<td>f2</td>
<td>6</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td>fdct</td>
<td>6</td>
<td>1.3</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>635.7</td>
<td>*</td>
</tr>
<tr>
<td>mwd</td>
<td>53</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>ptseng</td>
<td>4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0.3</td>
<td>1.3</td>
</tr>
<tr>
<td>s2</td>
<td>9</td>
<td>1.8</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3.8</td>
<td>6.0</td>
</tr>
<tr>
<td>xd3</td>
<td>14</td>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

A table entry containing a '*' means: time > 1 day and stopped.
1. For the smallest values of $T_{\text{max}}$ the MILP-solver is faster than the enumeration algorithm. For larger values the enumeration algorithm is faster. In general, the combining of equations is very effective.

2. The MILP-solver is faster for the XG-model and the L-model than for the X-model. Both of them are just about as fast. The tightening of the constraints influences the speed of the algorithm positively.

3. The list scheduler is faster than all the ILP schedulers. List scheduling gives results for bigger examples in acceptable time.

4. The optimum schedule is found either within, say, half hour or takes a couple of days.

Comparing to computation times as given in [Hwang88-91] and [Gebotys91], the times found here are very high. The difference lies probably in the ILP solving method that is used.

It can be concluded prudently that of the ILP schedulers, the one using the enumeration algorithm, is the best in terms of calculation times.
6. Conclusions

The only difference between the ILP schedulers is the speed. All algorithms gave the same results for the objective value.

Comparing the speeds of the algorithms the enumeration algorithm is the fastest for bigger problems. The calculation times, however are very long as compared to the times needed by a heuristic algorithm such as list scheduling. Of course, optimal solutions are not guaranteed in that case.

Because of the long calculation times, the ILP scheduler is not suitable for an interactive design environment. However, it can be useful in the following two cases:

1. When a scheduling problem is considered, the following steps are made: First a lower bound of the objective function can be found by a preselection algorithm that calculates an assignment function. Then an upper bound can be found by a fast heuristic scheduler, for example a list scheduler. If there is not much difference between these bounds, an ILP scheduler could not improve the solution considerably. If, however, the upper and lower bounds, that were found, differ considerably, exhaustive search of the solution space could give better solutions. In that case the ILP scheduler can be used.

2. Further the results of an ILP scheduler are valuable as reference to determine the quality of heuristic schedulers. In that case the execution times are not very important.

Future work can be dedicated first to speeding up the enumeration algorithm. When the algorithm is fast enough, extensions to the model such as branches and loops can be included in the ILP formulation.
References


[Hafer83] Hafer L.J., A.C. Parker, A formal method for the specification, analysis and design of register transfer level digital logic,
IEEE Trans. on CAD, vol CAD-2, no. 1, january 1983.

[Heijligers91] Heijligers M.J.M.,
*Time constrained scheduling for high level synthesis*,

[Hemani90] Hemani A., A. Postula,
*A neural net based self organizing scheduling algorithm*,
Proceedings of the EDAC, Glasgow, March 1990, pp. 136-139.

[Hwang88] Hwang K.S., A.E. Casavant, M. Dragomirecky, M.A. d’Abreu,
*Constrained conditional resource sharing in pipelining synthesis*,
6th Int. Conf. on Computer Aided Design, November 7-10, 1988, Santa Clara, California, Digest of technical papers, pp 52-55,

[Hwang89] Hwang K.S., A.E. Casavant, C.T. Chang, M.A. d’Abreu,
*Scheduling and hardware sharing in pipelined data paths*,
7th Int. Conf. on Computer Aided Design, November 5-9, 1989, Santa Clara, California, Digest of technical papers, pp 24-27,

[Hwang90] Hwang C.T., Y.C. Hsu, Y.L. Lin,
*Optimum and heuristic data path scheduling under resource constraints*,

[Hwang91] Hwang C.T., J.H. Lee, Y.C. Hsu,
*A formal approach to the scheduling problem in high level synthesis*,

[Kovács80] Kovács L.B.,
*Combinatorial methods of discrete programming*, (Vol. 2 of *Mathematical methods of operations research*),
Akadémiai Kiadó, Budapest, 1980.

[Ku90] Ku D., G. DeMicheli,
*Relative scheduling under timing constraints*,
27th ACM/IEEE Design Automation Conference, June 24-28,
Lee J.H., Y.C. Hsu, Y.L. Lin,
*A new integer linear programming formulation for the scheduling problem in data path synthesis*,

McFarland M.C., A.C. Parker, R. Camposano,
*Tutorial on high level synthesis*,

McFarland M.C., A.C. Parker, R. Camposano,
The high level synthesis of digital systems,

Nestor J.A., G. Krishnamoorthy,
*SALSA: A new approach to scheduling with timing constraints*,

Papachristou C.A., H. Konuk,
*A linear program driven scheduling and allocation method followed by an interconnect optimization algorithm*,

Park N., A.C. Parker,
*Sehwa: A program for the synthesis of pipelines*,
[Parker86] Parker A.C.,  
*MAHA: A program for data path synthesis*,  

[Paulin89] P.G. Paulin, J.P. Knight,  
"Force directed scheduling for the behavioral synthesis of ASIC's",  

[Potasman90] Potasman R., J. Lis, A. Nicolau, D. Gajski,  
*Percolated based synthesis*,  

[Potkonjak89] Potkonjak M., J. Rabaey,  
*A scheduling and resource allocation algorithm for hierarchical signal flow graphs*,  

[Stok91] Stok L.,  
*Architectural synthesis and optimization of digital systems*, PhD Thesis, Eindhoven University of Technology,  

[Taha75] Taha H.A.,  
*Integer programming, theory, applications and computations*,  

[Tseng86] Tseng C.J., D.P. Siewiorek,  
*Automated synthesis of data paths in digital systems*,  

[Wakabayashi89] Wakabayashi K., T. Yoshimura,  
*A resource sharing and control synthesis method for conditional branches*,  
7th Int. Conf. on Computer Aided Design, November 5-9, 1989,
Appendix A : Module counts

In this appendix the test results are summarized.

The sizes of the data flow graphs for which ILP scheduling was performed, are given in table A1.

<table>
<thead>
<tr>
<th>Table A1. Size of the data flow graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Dfg )</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td>(</td>
</tr>
</tbody>
</table>

The module sets are characterized as follows:

Module set 1

\[
L = \{ \text{adder, multiplier} \};
\]

\[
\rho(\text{adder}) = \{ '+', '-' \};
\]

\[
\delta_{\text{adder}}('+) = \delta_{\text{adder}}('-') = 1;
\]

\[
\rho(\text{multiplier}) = \{ '*', '/' \};
\]

\[
\delta_{\text{multiplier}}('*') = \delta_{\text{multiplier}}('/) = 1;
\]

\[
c(\text{adder}) = c(\text{multiplier}) = 1;
\]

Module set 2

\[
L = \{ \text{adder, multiplier} \};
\]

\[
\rho(\text{adder}) = \{ '+', '-' \};
\]

\[
\delta_{\text{adder}}('+) = \delta_{\text{adder}}('-') = 1;
\]

\[
\rho(\text{multiplier}) = \{ '*', '/' \};
\]

\[
\delta_{\text{multiplier}}('*') = \delta_{\text{multiplier}}('/) = 2;
\]

\[
c(\text{adder}) = c(\text{multiplier}) = 1;
\]

Module set 3

\[
L = \{ \text{ALU} \};
\]

\[
\rho(\text{ALU}) = \{ '+', '-', '*', '/' \};
\]

\[
\delta_{\text{ALU}}('+') = \delta_{\text{ALU}}('-') = 1;
\]

\[
\delta_{\text{ALU}}('*') = \delta_{\text{ALU}}('/) = 1;
\]

\[
c(\text{ALU}) = 1;
\]

The numbers of modules that were found by the enumeration algorithm and the CPU time this algorithm needed, are given in table A2 as function of the data flow graph, module set and number of timeslots.
'-' means that $T_{\text{max}}$ is too small.

'+' means that $T_{\text{max}}$ is bigger than necessary; the number of modules that is needed can not be reduced further.

'*' means that the calculation time is bigger than 1 day.

**TABLE A2.** Overview test results enumeration algorithm

<table>
<thead>
<tr>
<th>$Dfg$</th>
<th>$T_{\text{max}}$</th>
<th>Module set 1</th>
<th>Module set 2</th>
<th>Module set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>adder</td>
<td>mult</td>
<td>CPU</td>
<td>adder</td>
</tr>
<tr>
<td>f2</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fdct</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mwd</td>
<td>53</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ptseng</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>s2</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
<td>3</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>-</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>3</td>
<td>3</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>