Scheduling of Tasks with Precedence Delays and Relative Deadlines - Framework for Time-optimal Dynamic Reconfiguration of FPGAs

Přemysl Šůcha, Zdeněk Hanzálek

Centre for Applied Cybernetics, Department of Control Engineering
Czech Technical University in Prague, {suchap,hanzalek}@fel.cvut.cz

Abstract

This paper is motivated by existing architectures of field programmable gate arrays (FPGAs). To facilitate the design process we present an optimal scheduling algorithm using a very universal framework, where tasks are constrained by precedence delays and relative deadlines. The precedence relations are given by an oriented graph, where tasks are represented by nodes. Edges in the graph are related either to the minimum time or to the maximum time elapsed between the start times of the tasks. This framework is used to model the runtime dynamic reconfiguration, synchronization with an on-chip processor and simultaneous availability of arithmetic units and SRAM memory. The NP-hard problem of finding an optimal schedule satisfying the timing and resource constraints while minimizing the makespan C_{max}, is solved using two approaches. The first one is based on Integer Linear Programming and the second one is implemented as a Branch and Bound algorithm. Experimental results show the efficiency comparison of the ILP and Branch and Bound solutions.

Index Terms – Off-line scheduling, high–level synthesis, branch and bound, ILP, FPGA.

1 Introduction

The studied problem is motivated by existing architectures of field programmable gate arrays (FPGAs). Some of these architectures support runtime dynamic reconfiguration allowing one to change parts of the device, e.g. floating point units, while the rest operates at full speed. An example of such an architecture is Xilinx Virtex2 (i.e. starter kit from Memec with XC2V1000-FG365-4) which contains a static part with a MicroBlaze 32-bit RISC soft processor and a dynamically reconfigurable part [1]. Another one, based on Atmel AT94K, contains a static part with the hard core of the AVR 8-bit microcontroller and a dynamic part with user defined designs [1], in our case different floating point units.

The *arithmetic units* (i.e. coarse grain modules computing some function) are usually pipelined. Therefore, time properties of operations (tasks T_i) on the units are characterized by the time needed to feed the unit (*processing time* p_i) and the input–output latency (*precedence delay*). The input–output latency is formalized as a positive weight w_{ij} (associated with the data produced by T_i). Synchronization with the external environment (including an on-chip processor) can be modeled by *relative deadlines* formalized as a negative weight w_{ij}.

For illustration of the arithmetic units, we may consider two types of arithmetic libraries operating with real numbers. The first type is the logarithmic number system arithmetics, namely the High-Speed Logarithmic Arithmetic (HSLA) library [11] implementing multiplication, division and square-root operations simply as fixed-point addition, subtraction and right shift. Addition and subtraction operations require more complicated evaluation, hence only one pipelined addition/subtraction unit is usually available for a given application. On the other hand, the number of multiplication, division and square root units can be almost unlimited. For that reason, the scheduling of algorithms on HSLA (without reconfiguration) can be formalized as *scheduling of tasks with precedence delays and relative deadlines on one dedicated processor*. This problem will be called 1–DEDICATED in the rest of this article.

The second type is the floating-point library, namely FP32 by Celoxica [5]. It uses the widely known IEEE format to store the data. In this case, each unit requires an important number of hardware elements on the gate array, hence only one unit of each kind is usually available for a given application. Scheduling of algorithms on an FP32 can be formalized as *scheduling of tasks with precedence delays and relative deadlines on the set of dedicated processors*. This problem will be called m–DEDICATED and can be reduced to the 1–DEDICATED problem.

The most important problem of dynamic reconfiguration, not seen in the classical (i.e. static) design, is the temporal interdependence of the individual parts to be reconfigured. Scheduling algorithms have to distinguish, whether and when the reconfiguration is performed. When task T_j is processed immediately after task T_i and task T_j is processed by a different reconfigurable unit (i.e. currently not available in FPGA), the processing of T_i is charged by the *reconfiguration time*. This makes it almost impossible to manually produce efficient designs without a proper methodology. For simplicity, in this article, we assume that one unit is present in the reconfigurable part at a given time. The monoprocessor version of this problem can be formalized as a *monoprocessor problem with changeover times*. We will show its reduction to the m–DEDICATED problem and direct solution by ILP.

Not only the reconfigurable units, but also the memory units, become limited resources that have to be taken into consideration in many practical applications of FPGAs. In such cases, some arithmetic units and some memory units, may be required at one moment. This problem is formalized as *multiprocessor task scheduling*. We will also show its reduction to the m–DEDICATED problem and direct solution by ILP.

1.1 Related Work

Traditional off–line scheduling algorithms (e.g. [2]), typically assume that deadlines are absolute, i.e. the deadlines are related to the beginning of the schedule. On the other hand, when assuming, for example, interactions among an on-chip processor, pipelined units and the external environment, we may conveniently represent a given timing requirement by a deadline of the task T_j related to the start time of the task T_i. In such a case, when deadlines cannot be calculated *a priory*, we use *relative deadlines*.

In connection with project planning, the concept of precedence delays and relative deadlines (called positive and negative time-lags or start-to-start constraints or generalized precedence relations) have been introduced by Roy [12] and further developed by Brucker et al. [4]. A heuristic solution by Hurink [7] is based on a local search algorithm. Another heuristic approach to the scheduling of signal processing algorithms with relative deadlines was studied in [8]. A more general problem known in the Operation Research literature as the resource constrained project scheduling problem with generalized precedence relations (or RCPSP/max) [6] considers resources with capacity greater than 1.

From the time complexity point of view, the 1–DEDICATED problem is NP-hard, since the scheduling problem $1|r_j, \widetilde{d}_j|C_{max}$ [3] is reducible to it. Moreover, other NP-hard problems (e.g. the pipeline scheduling problem presented in [10]) are also reducible to it. On the other hand, the 1–DEDICATED problem is decidable, since it can be solved by ILP.

1.2 Outline

This paper shows, how the time-optimal FPGA design with several relevant architectural features (pipelined units, FPGA fabric interaction with an on-chip processor, arbitrary/restricted number of units, simultaneous memory access, dynamic reconfiguration) may be formalized as off–line scheduling of non–preemptive tasks with precedence delays (minimum delay between two tasks) and relative deadlines (maximum delay between two tasks). We adopt this simple but very universal model, we propose two original algorithms solving the scheduling problem and we evaluate their performance. The first algorithm is based on the Branch and Bound (B&B) method and the second one on Integer Linear Programming (ILP).

The main contributions of this paper are: (a) formalization of the time-optimal FPGA design with pipelined units, an on-chip processor, arbitrary/restricted number of dedicated units, simultaneous memory access and dynamic reconfiguration by the scheduling problem with precedence delays and relative deadlines. (b) an original solution of the m–DEDICATED problem by B&B algorithm. (c) an original ILP based solution of the m–DEDICATED problem. (d) direct ILP based solutions of multiprocessor task scheduling and the multiprocessor problem with changeover times (i.e. without reduction to m–DEDICATED). (e) a performance evaluation of the B&B solution, ILP solution and heuristic solution [8].

This paper is organized as follows: Section 2 presents the formulation of the m–DEDICATED problem and polynomial reductions of the mentioned scheduling problems. Section 3 presents the solution of the m–DEDICATED problem by ILP and direct solutions of multiprocessor task scheduling and the multiprocessor problem with changeover times. The next section describes the B&B algorithm, which solves the m–DEDICATED problem. Finally, the experimental results and comparisons are summarized in Section 5.

2 Problem Statement

2.1 Formulation of the m–DEDICATED Problem

The set of n tasks $\mathcal{T} = \{T_1, \ldots T_i, \ldots T_n\}$ is constrained by the precedence relations, given by task-on-node graph G. Each operation is represented by task T_i corresponding to node T_i on graph G and has a non-negative processing time p_i. Timing constraints between two nodes are represented by a set of directed edges. Each edge e_{ij} from node T_i to node T_j is labeled by an integer weight w_{ij}. There are two kinds of edges: the *forward edges* with positive weights and the *backward edges* with negative weights. The forward edge, from node T_i to node T_j with the positive weight w_{ij}, indicates that s_j, the start time of T_j, must be at least w_{ij} time units after s_i, the start time of T_i. The weight w_{ij}, associated with the forward edge, specifies so called *precedence delays*. We use the precedence delay to represent, for example, the input–output latency of the pipelined unit.

The backward edge, from node T_j to node T_i with the negative weight w_{ji}, indicates that s_j must be no more than $|w_{ji}|$ time units after s_i. Therefore, each negative weight w_{ji} represents $\widetilde{d}_j(s_i)$, the deadline of T_j, such that $\widetilde{d}_j(s_i) = s_i + |w_{ji}| + p_j$. A so-called *limited graph* can be obtained by removing all backward edges from graph G. The limited graph is acyclic.

In this paper, we are concerned with non-preemptive scheduling on the set of m dedicated processors $\mathcal{P} = \{P_1, \ldots P_d, \ldots P_m\}$. The set of tasks $\mathcal{T}$ is a conjunction of disjoint sets $\mathcal{T}_1, \ldots \mathcal{T}_d, \ldots \mathcal{T}_m$. Both tasks T_i and T_j are assigned to the dedicated processor P_d, if and only if, $T_i \in \mathcal{T}_d$ and $T_j \in \mathcal{T}_d$. Therefore, if s_i is the start time of task T_i scheduled on processor P_d, then no other task can be scheduled before $s_i + p_i$ time units on the same processor.

The scheduling problem is to find a *feasible schedule*, satisfying the timing and resource constraints, while minimizing the makespan C_{max}. Let S be a schedule given as a vector $S = (s_1, s_2, \ldots s_n)$, such that each couple of nodes T_i and T_j with a nonzero edge w_{ij} has the start times satisfying the equation

$$s_j - s_i \geq w_{ij}. \tag{1}$$

Equation (1) holds for both, the forward and backward edges.

Let $\mathcal{T} = \{\mathcal{T}_1, \ldots \mathcal{T}_d, \ldots \mathcal{T}_m\}$ be the set of $n = n_1 + \ldots + n_d + \ldots + n_m$ tasks to be scheduled, where n_d is the number of tasks assigned to the dedicated processor P_d. The processing time vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, $n \times n$ dimensional matrix of weights $\mathbf{W}$ and the sets of tasks running on the dedicated processors $\mathcal{T}_1, \ldots \mathcal{T}_d, \ldots \mathcal{T}_m$ are input parameters of the addressed problem. Matrix $\mathbf{W}$ is composed of timing constraints w_{ij} related to edges e_{ij} (between tasks T_i and T_j). There is no edge from node T_i to node T_j when $w_{ij} = -\infty$. There is a forward edge when $w_{ij} > 0$ and there is a backward edge when $w_{ji} < 0$, and $w_{ii} = -\infty$.

Example: One instance of the scheduling problem under consideration containing six tasks T_1,T_2,…,T_6 is given by the graph G and the corresponding matrix of weights $\mathbf{W}$ in Figure 1. The set of tasks $\mathcal{T}_1 = \{T_1, T_2, T_6\}$ is assigned to the first processor and $\mathcal{T}_2 = \{T_3, T_4, T_5\}$ to the second one. The processing times are $\mathbf{p} = (1, 3, 2, 3, 4, 5)$ and the precedence relations are given by the corresponding graph including backward edges (the start time of task T_6 has to be at a maximum of 11 time units after the start time of task T_1, and the start time of task T_3 has to be at a maximum of 3 time units after the start time of task T_1).

$$\mathbf{W} = \begin{pmatrix} -\infty & -\infty & 1 & 2 & 3 & -\infty \\ -\infty & -\infty & 4 & 4 & 4 & 13 \\ -3 & -\infty & -\infty & -\infty & -\infty & 4 \\ -\infty & -\infty & -\infty & -\infty & -\infty & 4 \\ -\infty & -\infty & -\infty & -\infty & -\infty & 4 \\ -11 & -\infty & -\infty & -\infty & -\infty & -\infty \end{pmatrix}$$

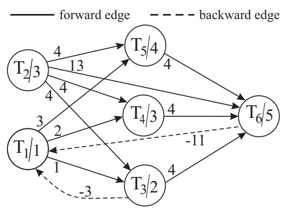

Figure 1. A scheduling problem example.

2.2 Reduction of m–DEDICATED to 1–DEDICATED

This subsection shows, how each task of the m–DEDICATED problem can be transformed to an instance of a 1–DEDICATED problem scheduled on one virtual processor. The most important property of the problem formulation by tasks with precedence delays and relative deadlines is the universality. Brucker [4] has shown that many scheduling problems can be polynomially reduced to this problem and effectively solved as outlined in Subsections 2.3 and 2.4.

The reduction from m–DEDICATED to 1–DEDICATED requires an upper bound $\overline{C}$ of the C_{max}, that can be found e.g. as $\overline{C} = \sum_{i=1}^{n} \max\left(p_i, \max_{j \in \langle 1,n \rangle} l_{ij}\right)$. The main idea lies in the assignment of all tasks on one virtual monoprocessor while using forward and backward edges to enclose $\mathcal{T}_d$ into the d-th interval on the virtual processor. Let $T_i \in \mathcal{T}_{d_i}$ and $T_j \in \mathcal{T}_{d_j}$, then task T_i is assigned to the time interval $\left\langle (d_i - 1) \cdot \overline{C}, d_i \cdot \overline{C} \right\rangle$ on the monoprocessor while recalculating its start time as $s'_i = s_i + (d_i - 1) \cdot \overline{C}$.

Then the timing constraint (1) is translated to $s'_j - s'_i \geq w'_{ij}$, where $w'_{ij} = w_{ij} + (d_i - d_j) \cdot \overline{C}$. Furthermore, the enclosure of tasks $T_i \in \mathcal{T}_{d_i}$ into the interval $\left\langle (d_i - 1) \cdot \overline{C}, d_i \cdot \overline{C} \right\rangle$ is realized by introduction of a dummy task T_0 with $p_0 = 0$ and by introduction of the couple of new edges for each task T_i such as $s'_i - s'_0 \leq (d_i - 1) \cdot \overline{C}$ and $s'_0 - s'_i \leq d_i \cdot \overline{C}$.

Finally, we add a second dummy task T_{n+1} with $p_{n+1} = 0$ to formulate the C_{max} minimization of the m–DEDICATED problem in terms of C_{max} minimization of the 1–DEDICATED problem. Therefore, for each task T_i, we add one relation $s_i + p_i \leq s_{n+1}$.

2.3 Reduction of the Problem with Multiprocessor Task to m–DEDICATED

The multiprocessor task scheduling problem is an extension of the m–DEDICATED problem, where task T_i is associated with a set of processors $\mathcal{P}_i = \left\{ P_i^1, \ldots, P_i^{m_i} \right\} \subset \mathcal{P}$. The reduction of this problem to the m–DEDICATED problem is based on the propagation of each task T_i to the set of virtual tasks $T_i^1, \ldots T_i^v, \ldots T_i^{m_i}$. These tasks have the same processing time as T_i and each tasks T_i^v is assigned to virtual processor $P_i^v \in \mathcal{P}_i$. These tasks are forced to start at the same time by a couple of *synchronization relations*

$$s_i^v - s_i^1 \geq 0 \quad \text{and} \quad s_i^1 - s_i^v \geq 0. \qquad (2)$$

Finally, we define the timing constraint (1) between tasks T_i^1 and T_j^1 if and only if, there is a corresponding timing constraint between the original tasks T_i and T_j.

2.4 Reduction of the Monoprocessor Problem with Changeover Times to m–DEDICATED

This scheduling problem extends the 1–DEDICATED problem by partitioning the set of tasks $\mathcal{T}$ into groups $G_1, \ldots, G_r$. If task T_i from group G_k is scheduled immediately after task T_j from different group G_l, there is a *reconfiguratin time* r_{lk} i.e. task T_i starts at the earliest r_{lk} time units after the finishing time of T_j, otherwise $r_{lk} = 0$.

This problem can be also polynomially reduced to an m–DEDICATED scheduling problem. All tasks from $\mathcal{T}$ are processed on the first processor P_1. Furthermore, for each pair T_i, T_j belonging to different groups G_k, G_l we introduce a virtual processor P_{ij}. Two tasks T_{ij}^i and T_{ij}^j, with processing time $p_{ij}^i = p_i + r_{kl}$ and $p_{ij}^j = p_j + r_{lk}$ respectively, are dedicated to this virtual processor. By a couple of synchronization relations, task T_{ij}^i and T_{ij}^j are forced to start at the same time as T_i and T_j on processor P_1, respectively.

Finally C_{max} minimization is given by C_{max} of the processor P_1 that is reflected by introduction of a dummy task T_{n+1} in a similar way as in Subsection 2.2.

Due to the above mentioned reductions, we are able to formulate many practical scheduling problems on FPGA architectures with dynamic reconfiguration and limited resources such as the 1–DEDICATED problem. In the rest of the paper we show the solution of this problem by ILP and B&B algorithm. In order to illustrate various features of our solutions, we shall develop the more complex one, i.e. m–DEDICATED, and in the experimental section we compare the performance of ILP and B&B solutions.

3 Solution of m–DEDICATED Problem by ILP

Due to NP–hardness of the m–DEDICATED problem, it is meaningful to formulate it as problem of ILP, since various ILP algorithms solve instances of reasonable size in reasonable time. The schedule has to obey two kinds of constraints. The first is the *precedence constraint* restriction corresponding to Inequality (1). Since each edge represents one precedence constraint, we have n_e inequalities (n_e is the number of edges in graph G).

The second kind of restrictions are *processor constraints*, i.e. for each couple of tasks T_i and T_j assigned to the same processor, at maximum, one is executed at a given time. Two disjoint cases can occur. In the first case, we consider task T_j to be followed by task T_i. In the second case, we consider task T_i to be followed by task T_j.

Exclusive OR relation between the first case and the second case hinders the solution to the problem directly by LP (Linear Programming), since there is AND relation among all inequalities in the LP program. Therefore, we use a binary decision variable x_{ij} ($x_{ij} = 1$ when T_i is followed by T_j and $x_{ij} = 0$ when T_j is followed by T_i) and a very large positive number $\overline{C}$ ($\overline{C}$ is the upper bound of C_{max}).

Then the restrictions can be formulated as one double–inequality

$$p_j \leq s_i - s_j + \overline{C} \cdot x_{ij} \leq \overline{C} - p_i. \qquad (3)$$

To derive a feasible schedule, the double–inequality (3) must hold for each unordered couple of two tasks assigned to the same processor. Therefore, there are at maximum $\sum_{d=1}^{m}(n_d^2 - n_d)/2$ double–inequalities specifying processor constraints, where $n_d = |\mathcal{T}_d|$. The (3) is redundant for tasks T_i and T_j when there is a path on the limited graph from T_i to T_j or from T_j to T_i since the order of tasks is determined by precedence constraints as $w_{ij} \geq p_j$ for the forward edges.

Makespan minimization is realized by adding one variable C_{max} satisfying

$$s_i + p_i \leq C_{max}, \quad \forall T_i \in \mathcal{T} \tag{4}$$

for all sink nodes of the limited graph. Then the objective function is to minimize C_{max}. The summarized ILP program, using variables s_i, x_{ij}, C_{max}, is shown in Figure 2.

$$\begin{array}{ll}
\multicolumn{2}{l}{\min C_{max}} \\
\multicolumn{2}{l}{\text{subject to}} \\
s_j - s_i \geq w_{ij}, & \forall e_{ij} \in G \\
p_j \leq s_i - s_j + \overline{C} \cdot x_{ij} \leq \overline{C} - p_i, & \forall i < j : T_i, T_j \in \mathcal{T}_d \\
s_i + p_i \leq C_{max}, & \forall T_i \in \mathcal{T} \text{ sink node} \\
\multicolumn{2}{l}{\text{where}} \\
\multicolumn{2}{l}{s_i \in \langle 0, \overline{C} - p_i \rangle,\ x_i \in \{0,1\},\ C_{max} \in \langle 0, \overline{C} \rangle} \\
\multicolumn{2}{l}{s_i, x_{ij} \text{ are integers.}}
\end{array}$$

Figure 2. ILP program solving the m–DEDICATED scheduling problem.

3.1 Direct Solution of Problems with Multiprocessor Task and Changeover Times

The generality of ILP allows one to formulate scheduling problems outlined in Subsections 2.3 and 2.4 directly without polynomial reduction that increases the number of nodes in both cases. The problem with multiprocessor task can be solved by the ILP program in Figure 2 by a slight modification of the processor constraints (3). The processor constraint between T_i and T_j in the m–DEDICATED problem is considered, when $T_i, T_j \in \mathcal{T}_d$, i.e. both tasks are dedicated to the same processor, otherwise it is omitted. Therefore, in the problem with multiprocessor task, the processor constraint between T_i and T_j is considered when $\mathcal{P}_i \cap \mathcal{P}_j \neq \emptyset$, i.e. tasks T_i and T_j can not overlap if they require the same processor.

The monoprocessor scheduling problem with changeover times can also be directly solved by modification of the processor constraints (3). Since the processor constraint restricts the overlap between a couple T_i and T_j only, then processing times p_i and p_j can be considered different for different couples of tasks, i.e. different processor constraints. Therefore, considering tasks T_i and T_j belonging to different groups G_k, G_l, the processing time used in the processor constraint (3) for these tasks is considered as $p_{ij}^i = p_i + r_{kl}$ and $p_{ij}^j = p_j + r_{lk}$, respectively. For tasks from the same group the processor constraint (3) stays the same.

4 Solution of the m–DEDICATED Problem by the Branch and Bound Algorithm

The Branch and Bound algorithm creating schedule S defines a partitioning of $\mathcal{T}$, the set of tasks, into three disjoint subsets $\mathcal{T}_S(S)$, $\mathcal{T}_C(S)$ and $\mathcal{T}_R(S)$, where $\mathcal{T}_S(S)$ is the set of already scheduled tasks, $\mathcal{T}_C(S)$ is the set of *candidate* tasks and $\mathcal{T}_R(S)$ is the set of remaining tasks.

A task $T_k \in \mathcal{T}_C$ is a candidate to be scheduled into the partial schedule S if T_k has not been scheduled yet and all its predecessors belong to $\mathcal{T}_S$, the set of scheduled tasks.

If node $T_k \in \mathcal{T}_C$, assigned to the processor P_d, is chosen to be scheduled at time h_d, it is added to the current partial schedule S with the start time equal to the maximum of all its precedence timing constraints and the current time h_d

$$\begin{aligned} s_k &= \max(h_d, \max_{\forall T_i \in \mathcal{T}_K}(s_i + w_{ik})) \\ \mathcal{T}_K &= \{T_i \in \mathcal{T}_S : w_{ik} > 0\}. \end{aligned} \tag{5}$$

After scheduling task T_k, sets $\mathcal{T}_S(S)$, $\mathcal{T}_C(S)$ and $\mathcal{T}_R(S)$ have to be actualized.

The scheduling problem can be solved by the enumeration of a finite set F of *feasible solutions* respecting timing constraints while the objective is to find an *optimal solution* $S^* \in F$ with the minimal C_{max}. For enumeration of F, the Branch and Bound (B&B) method is used.

The branching procedure can be conveniently represented as a search tree (see Figure 3). Each vertex in the n-th level represents one *final solution*. All n tasks are scheduled in the final solution. All vertices in levels 1 to $n-1$ correspond to a *partial solution* representing an uncompleted schedule.

In order to implement the scheme of the B&B algorithm for our scheduling problem, one must first describe the branching procedure and the search strategy. A very simple recursive procedure creating the search tree of feasible solutions (see Equation (1)) can be stated as follows:

```
B&B algorithm:
1. [Initialization] Set $s_i = \infty \ \forall \ T_i \in \mathcal{T}$. $\mathcal{T}_S = \emptyset$.
   Find $\mathcal{T}_C$, the set of schedulable tasks (tasks without predecessors). The best known final solution $S^B = S$.
2. [Recursion] Call vertex_exploration($\mathcal{T}_S$,$\mathcal{T}_C$, $S$).
3. $S^* = S^B$

Recursive procedure: vertex_exploration($\mathcal{T}_S$, $\mathcal{T}_C$, $S$)
1. [Bounding] Explained in Section 4.1. If the solution is feasible, then go to step 2, otherwise go to step 4.
2. [Test the solution]
   (a) If $\mathcal{T}_C = \emptyset$, the current solution $S$ is a final solution. Assign the current solution $S$ to
       $S^B$ if $C_{max}(S) < C_{max}(S^B)$. Go to step 4.
   (b) If $\mathcal{T}_C \neq \emptyset$, the current solution is a partial solution and then go to step 3.
3. [Scheduling of candidates] $\forall T_k \in \mathcal{T}_C$ do:
   • Schedule the task $T_k$ by creating $S^N$ such that $s_i^N = s_i$ for all $i \neq k$ and $s_k$ is calculated using Equation (5).
   • Create new sets $\mathcal{T}_C^N$ and $\mathcal{T}_S^N$.
   • Call vertex_exploration($\mathcal{T}_S^N$, $\mathcal{T}_C^N$, $\mathcal{S}^N$).
4. [Return]
```

Example: (continued from Section 2.1) The complete set of solutions and partial solutions arranged in the search tree can be found in Figure 3. The leaves in the 6-th level of the search tree are the final solutions. The non-feasible solutions are crossed out. In this example, there are two feasible final solutions $S_1 = (3, 0, 4, 6, 9, 13)$ and $S_2 = (3, 0, 4, 10, 6, 14)$ therefore, the solution S_1 is optimal, i.e. $S^* = S_1$.

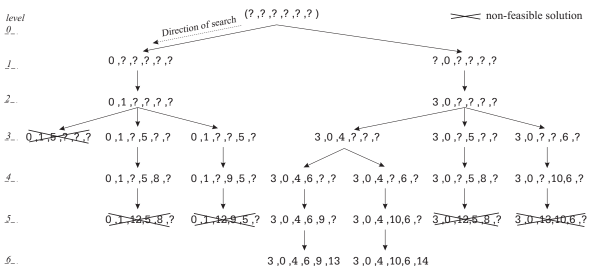

Figure 3. An example of the search tree.

4.1 Bounding in the Search Tree

A bounding mechanism reduces the size of the search tree (used in step 1 of the *vertex_exploration* procedure). At the same time the bounding mechanism cannot eliminate any vertex on the unique path from the root to S^*, the optimal solution.

Our B&B algorithm uses two bounding methods *Critical Path Bounding* and *Remaining Processing Time Bounding* reducing the number of search steps of the B&B algorithm. These methods are based on the lower bound estimation of s_j and they are implemented as an extension of step 1 in the *vertex_exploration* procedure. For more details see [13].

4.2 Scheduling Anomaly

The branching mechanism only determines the order of the tasks scheduled, as soon as possible, according to Equation (5). This approach satisfies feasibility only with respect to the forward edges and the minimum C_{max} of the schedule. Unfortunately, this mechanism may result in a scheduling *anomaly* in the constructed schedule with respect to feasibility given by backward edges. Let us consider a backward edge e_{ij} from task T_j to T_i. If *lateness* $L_j = s_i - s_j - w_{ji}$ of task T_j is greater than zero (task T_j missed its deadline by L_j ticks) then it is necessary to test, whether task T_i can be scheduled L_j ticks later (see example in Figure 4).

The test is performed via shifting T_i by L_j and by recalculation of the start times of other scheduled tasks while Equation (5) is used to satisfy feasibility with respect to the forward and backward edges. This "shifting" can cause an increase of s_j by the value in $\langle 0, L_j \rangle$. If this value is 0, "shifting" does not increase the start time of T_j, the solution is feasible and the branching procedure can continue with the recalculated schedule S^N. Otherwise, the partial solution is infeasible.

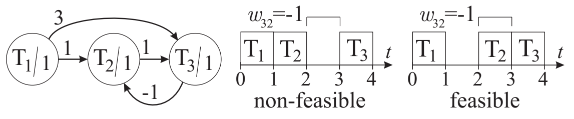

Figure 4. A scheduling anomaly.

5 Experimental Results

The presented scheduling algorithms were implemented and tested in C language. The integer linear program was solved by a non-commercial ILP solver tool LP_SOLVE [9].

#	CPU time [s]			# inspected vertices [-]		
tasks	B&B0	B&B	ILP	B&B0	B&B	ILP
12	0.0016	0.0005	0.0244	516	63	137
14	0.0237	0.0022	0.1282	5605	264	868
16	0.2229	0.0116	0.8583	38490	1183	5355
18	5.3345	0.0835	7.4887	755950	6120	38740
20	133.8341	0.7664	51.0145	15991000	51000	221000

Table 1. Experimental results of the scheduling algorithm complexity.

#	CPU time [s]			# insp. vertices [-]		
processors	B&B	B&B_R	ILP	B&B	B&B_R	ILP
1	0.084	0.084	7.489	6120	6120	38740
2	0.610	0.266	0.228	50426	6620	1387
3	1.251	0.205	0.086	105420	4340	540
4	2.000	0.132	0.029	181230	2270	160
5	3.838	0.090	0.018	276620	1550	70

Table 2. Influence of the number of processors (for 18 nodes with 27 forward edges and 9 backward edges).

# backward	CPU time [s]		# insp. vertices [-]	
edges	B&B	ILP	B&B	ILP
8	0.0686	5.6610	5193	28765
12	0.0440	2.5917	2896	13343
16	0.0298	1.3459	1888	6763
20	0.0129	0.3298	703	669
24	0.0043	0.1041	210	478

Table 3. Influence of the number of backward edges (for 18 nodes with 27 forward edges on one processor).

To compare the ILP method with the B&B algorithm we measured the average CPU times for randomly generated instances. The weights of the forward edges w_{ij} were chosen from a uniform distribution on the interval $\langle 1, 16 \rangle$ (and rounded to the nearest integer). The processing time p_i was generated from uniform distribution on the interval $\left\langle 1, \min_{T_j \in \mathcal{T}}(w_{ij}) \right\rangle$. And weights of the backward edges w_{ji} were chosen from a uniform distribution on the interval $\langle f_{ij}, 2.2 \cdot f_{ij} \rangle$ where f_{ij} is the longest path by the forward edges. A dedicated processor for a node was chosen from a uniform distribution on the interval $\langle 1, m \rangle$.

CPU time depends on the efficiency of the algorithm implementation, compilation and speed of the computer. ILP also uses a branch and bound algorithm while solving linear programs in each vertex of its search tree. Therefore, comparison of the average number of processed vertices allows one to examine the results of both methods. The total number of processed vertices of ILP in Table 1 is given by an average value of the variable *total_nodes* declared in lpkit.h of LP_SOLVE tool.

Table 1 shows the algorithm complexity for one dedicated processor ($m = 1$). Five hundred scheduling instances have been generated per each number of nodes (tasks) and the mean value of the CPU time and the number of inspected solutions in each scheduling problem is shown in Table 1. The scheduling problems were generated in a random manner as explained above. The number of forward edges in Table 1 is equal to $3 \cdot n/2$ and the number of backward edges is equal to $n/2$.

When all bounding methods are combined together (column B&B in Table 1), the number of inspected states is 0.3% of all feasible solutions (B&B0) for 20 nodes. The ILP solution is less efficient, since it inspects 1.4% of all feasible schedules in the same case.

The influence of the number of processors is illustrated in Table 2, where column B&B indicates the performance of the B&B algorithm solving the m–DEDICATED problem as described in Section 4. Column B&B_R indicates the case, when the instance of the m–DEDICATED problem is firstly transformed to the 1–DEDICATED problem and then solved by the B&B algorithm. With a growing number of processors, B&B_R behaves much better than B&B, since it does not suffer from enumeration of multiple partial solutions. The average CPU times of the ILP method decreases with the number of processors since for a fixed number of nodes the ILP model, for the problem with more dedicated processors, has less decision variables $(\sum_{d=1}^{m}(n_d^2 - n_d)/2)$. On the other hand the B&B algorithm has the opposite behavior since the number of combinations increases with the number of processors.

Table 3 shows the influence of the number of backward edges on the average CPU time and the average number of inspected vertices in the search tree. For both methods, it holds that with the increase of the number of backward edges the time complexity of the problem decreases, since each relative deadline limits the state space of feasible solutions and the optimal solution can be found faster.

The comparison with polynomial heuristics H0 and H1 [8] is shown in Table 4. The first column (failure) shows the percentage of cases when H0, H1 respectively have not found any feasible solution, while both B&B and ILP give the feasible solutions. The second column shows the average relative error of $C_{max}(S^{H0})$ and $C_{max}(S^{H1})$ found by heuristics H0 and H1 respectively, with respect to the optimal value of $C_{max}(S^*)$ in cases when H0, H1 respectively have found the feasible solution. The relative error was calculated as $(C_{max}(S^{H0}) - C_{max}(S^*))/C_{max}(S^*)$, $(C_{max}(S^{H1}) -$

# backward edges	failure [%]		avg. rel. err. [%]	
	H0	H1	H0	H1
4	42.2	15.6	29.01	24.98
6	55.6	23.5	22.43	17.12
8	65.1	30.3	10.71	10.70
10	73.2	32.4	7.60	8.30
12	79.8	33.7	5.05	6.42
14	81.1	35.3	3.41	4.52
16	83.9	38.7	1.77	3.06

Table 4. Comparison with heuristics H0 and H1. Influence of the number of backward edges (for 18 nodes with 27 forward edges on one processor).

$C_{max}(S^*))/C_{max}(S^*)$ respectively. The results in Table 4 show limited applicability of heuristics H0 and H1 for instances with a growing number of backward edges, where the B&B algorithm and ILP method have much lower computing time requirements (see Table 3).

Note: The experiments were performed on a PC Intel Pentium 4, 2.4GHz. Results in tables 2, 3 and 4 are mean values over two hundred randomly generated sets of input data.

6 Conclusions

This paper deals with problems related to the optimal scheduling of FPGA design with pipelined function units, an on–chip processor, an arbitrary/restricted number of units, simultaneous memory access and dynamic reconfiguration. We have shown that all these problems can be polynomially reduced to the scheduling problem with precedence delays and relative deadlines.

Furthermore, we have presented our original Branch and Bound algorithm and ILP based solution for the scheduling problem with precedence delays and relative deadlines on the set of dedicated processors. It was shown that the ILP based solution can solve problems with simultaneous memory access and dynamic reconfiguration without adding virtual tasks (required by reduction to the m–DEDICATED problem) and therefore is more effective. Experimental results show impressive power of the rather simple and elegant ILP solution, namely in the case of the increasing number of dedicated processors. In comparison with the available heuristics, it is obvious that the heuristics are not suitable for instances with more relative deadlines. On the other hand, the B&B and ILP solutions have lower computation requirements with increasing number of relative deadlines and therefore the optimal solution can be found faster.

7 Acknowledgments

This work was supported by the Ministry of Education of the Czech Republic under Project 1M0567.

References

[1] R. Bartosinski, M. Daněk, P. Honzík, and R. Matoušek. Dynamic reconfiguration in FPGA-based SoC designs. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, pages 274–274, Monterey, California, USA, 2005.

[2] J. Błazewicz, K. Ecker, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer, second edition, 2001.

[3] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints. Naval Res. Logist. Quart. 18, 1971.

[4] P. Brucker, T. Hilbig, and J. Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94(1-3):77–99, May 1999.

[5] Celoxica Ltd. *Platform Developers Kit: Pipelined Floating-point Library Manual*, 2004. `http://www.celoxica.com`.

[6] A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.

[7] J. Hurink and J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 112(1-3):179–197, 2001.

[8] M. Jacomino, D. Gutfreund, and J. Pulou. Scheduling real-time processes with timing constraints and its applications to cyclic systems. Report internal, 1999.

[9] J. C. Kantor. *Mixed Integer Linear Program Solver LP_SOLVE 4.0*. ftp://ftp.es.ele.tue.nl/pub/lp_solve/, 1995.

[10] A. M. Kordon. Minimizing makespan for a bipartite graph on a single processor with an integer precedence delay. *Operations Research Letters*, 32(6):557–564, November 2004.

[11] R. Matoušek, M. Tichý, Z. Pohl, J. Kadlec, and C. Softley. Logarithmic number system and floating–point arithmetics on FPGA. In *Field–Programmable Logic and Applications: Reconfigurable Computing is Going Mainstream*, volume 2438 of *Lecture Notes in Computer Science*, pages 627–636, Berlin, 2002. Springer.

[12] B. Roy. Contribution de la théorie des graphes à l'étude de certains problèmes linéaires. *C. R. Acad. Sci. Paris*, 248:2437–2439, 1959.

[13] P. Šůcha and Z. Hanzálek. Scheduling with start time related deadlines. In *CCA/ISIC/CACSD'04 IEEE Computer Aided Control Systems Design*, September 2004.