Deadline Constrained Cyclic Scheduling on Pipelined Dedicated Processors Considering Multiprocessor Tasks and Changeover Times

Přemysl Šůcha [a,*], Zdeněk Hanzálek [a]

[a] *Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University, Karlovo náměstí 13, 121 35 Prague 2, Czech Republic*

Abstract

This paper presents a scheduling technique used to optimize computation speed of loops running on architectures that may include pipelined dedicated processors. The problem under consideration is to find an optimal periodic schedule satisfying the timing constraints. Motivated by FPGA (Field-Programmable Gate Array) architecture we formulate a problem of cyclic scheduling on one dedicated processor where tasks are constrained by the precedence delays. Further we generalize this result to the set of dedicated processors. We also show how the set of constraints in both problems can be extended by start time related deadlines, multiprocessor tasks, changeover times and minimization of data transfers. We prove that this problem is NP–hard by reduction of Bratley's scheduling problem $1|r_j, \widetilde{d}_j|C_{max}$ and we suggest a solution based on ILP (Integer Linear Programming) that allows one to minimize the completion time. Besides this, we suggest elimination of redundant constraints and binary variables in integer linear programming model which leads to a speedup of the scheduling algorithm. Finally, experimental results are shown on an application of recursive least square filter and benchmarks.

Key words: Cyclic scheduling, integer linear programming, high–level synthesis, changeover times, FPGA

* Corresponding author. Fax: + 420 2 2435 5703
Email addresses: suchap@fel.cvut.cz (Přemysl Šůcha), hanzalek@fel.cvut.cz (Zdeněk Hanzálek).

25 February 2008

1 Introduction

This paper deals with automatic parallelization of computation loops performing an identical set of operations repeatedly. One repetition of the loop is called an *iteration*. A parallel implementation of the loop implies that each operation of the loop is mapped on a hardware unit at a given time, therefore the scheduling theory is used to find start times of these operations.

1.1 Problem Statement and Motivation

Cyclic scheduling deals with a set of operations (generic tasks) that have to be performed an infinite number of times [1]. This approach is also applicable if the number of loop repetitions is large enough. A schedule is called *nonoverlapped* if all operations belonging to one iteration have to finish before the next operations of the next iteration can start. If execution of operations belonging to different iterations can interleave, the schedule is called *overlapped* [2]. An overlapped schedule can be more effective especially if hardware units are pipelined. The *periodic schedule* is a schedule of one iteration that is repeated with a fixed time interval called a *period* (also called *initiation interval*). The aim is then to find a periodic schedule with a minimum period [1,3,4].

The studied problem is motivated by an application of RLS (Recursive Least Squares) filter for active noise cancellation [5]. A design of such filter gets complicated, due to the representation of real numbers, calculated by arithmetic units in FPGAs. We 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 [6] 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 can be formalized as *cyclic*

scheduling of tasks with precedence delays 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 [7]. 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 *cyclic scheduling of tasks with precedence delays on set of dedicated processors*. This problem will be called m'–DEDICATED in the rest of this article.

In both architectures, rather complex arithmetic units are required. Therefore, scheduling of such dedicated HW resources has to carefully consider the algorithm structure, in order to achieve the desired performance of applications. Scheduling also helps to choose the appropriate arithmetic library prior to the algorithm implementation.

Beside of the period minimization we consider three secondary objectives: the *overlap minimization* (simplest objectives, which does not increase computation requirements), the *makespan minimization* (suitable for small number of iterations) and *minimization of data transfers* (objective considering hardware implementation). Further, we propose three extensions of our problem: *start time related deadlines*, *multiprocessor tasks* and *changeover times*.

In many practical applications, the task must be completed within a given time limit. Such constraint is usually represented as a classical deadline, related to the start of the schedule. In this article we use more general constraint, called *start time related deadline*, i.e. deadline related to the start time of another task [8]. The start time related deadlines can be conveniently used to model not only the classical deadline, but also the task synchronization, the watchdog timer and other practical time constraints.

Not only the arithmetic 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*.

Some of the recent FPGA devices have the capability of partial runtime reconfiguration [9]. The most important problem of dynamic reconfiguration, not seen in classical (i.e. static) design, is temporal interdependence of units to be reconfigured. The scheduling algorithm have to distinguish, whether and when the reconfiguration is performed. If the task is processed by a unit, which is not currently available in FPGA, the processing is charged by *reconfiguration time*. This problem is formalized using *changeover times*.

The ILP based scheduling method, shown in this article, solves the problems mentioned above. The optimal solution found by this approach was used to schedule problems like RLS filter [10] and it was further extended to handle imperfectly nested loops like FICMA algorithm [11]. Moreover, such an optimal solution can be used to obtain performance metric for heuristic algorithms aiming to solve larger instances.

1.2 Related Work

If the number of processors is not limited, cyclic scheduling can be used to built a periodic schedule in polynomial time [1,12]. Unfortunately, for a fixed number of processors the problem becomes NP–hard [3]. If all tasks have unit processing times, several special cases with polynomial time complexity exist [3]. Related terms to *cyclic scheduling*, used in the scheduling community, are *modulo scheduling* and *software pipelining* [13,14], usually used in the compiler community.

Periodic schedule is a particular case of the *K-periodic schedule*, where two consecutive iterations may not be the same, but the schedule of an iteration is repeated every K iterations (integer parameter K is called *periodicity*). Then the objective of the K-periodic scheduling is to maximize the *computation rate*, which is the ratio of K to period length [15,16] (in the rest of this article we assume only the periodic schedule, since the gain of the K-periodic schedule is very small and the implementation of the K-periodic schedule is more complex when leading,

for example, to larger FPGA design).

Existing methods for the scheduling of loops can be divided into heuristic approaches and methods using integer linear programming (ILP). The heuristics–based techniques do not guarantee optimal solutions but have much lower computing requirements making them applicable in code compilers. On the other hand, ILP is not a polynomial algorithm but for problems with reasonable size it finds an optimal solution in a reasonable amount of time. Moreover, the ILP formulation is convenient since it allows one to add additional constraints to the scheduling problem (e.g. number of available registers and processor communications) or to formulate rather complex objective criterion (e.g. combination of the overlap and makespan minimizations).

An overview of the heuristic approaches is presented e.g. in [17,1]. Some heuristics use a retiming technique [18] which is based on the transformation of a scheduled loop to another one with a shorter iteration period by rearranging delays. Other approaches [19–21] use generalization of non–cyclic approximation list scheduling algorithms to cyclic problems. A comparative study of relevant heuristic modulo scheduling techniques is shown in [22].

A common approach based on ILP uses a binary decision variable x_{it} determining whether a computation of node i starts at time step t. Unfortunately, the size of such an ILP model (corresponding to time complexity) depends on the period length. The ILP approach with x_{it} decision variable is used in [23], where the additional objective is to optimize word–length of arithmetic units. In [24], the ILP model is extended by automatic processor selection from a library and [2] applies the approach to architectures with interprocessor communication. A general solution for K-periodic schedules by ILP, where the number of variables does not depend on the period length is shown in [16] and extended in [25]. With respect to this work, our solution leads to a simpler ILP formulation with less integer variables.

We propose a method based on integer linear programming to find an optimal periodic schedule. With respect to the target applications, the method is designed for instances with long period. Motivated by HSLA architecture [6] we formulate 1–DEDICATED problem where one dedicated processor represents a bottle neck of such architecture (i.e. addition/subtraction unit). The *precedence delays* represent pipelining and processing time of tasks executed on an unlimited number of multiplication/division/square root units. Further, we generalize this result to m'–DEDICATED problem, where m' indicates the number of dedicated processors.

This paper is organized as follows: Motivated by [1], Section 2 describes the Basic Cyclic Scheduling (BCS) problem assuming an unlimited number of processors. The next Section presents formulation of the scheduling problem and we show that Bratley's problem [26] is polynomially reducible to it. Assuming the given schedule period, we formulate the ILP model in Section 4. Unlike the most frequent ILP models we suggest the model where the number of variables does not depend on the period length. Further, we show extensions considering start time related deadlines, multiprocessor tasks, changeover times and minimization of register requirements. Moreover, we use linear programming (LP) to reduce the number of integer variables. A solution of the original problem (minimization of the schedule period), using iterative calls of ILP, is presented in Section 5. Section 6 presents the results demonstrated on benchmarks and shows a comparison with ILP model using one binary decision variable for each task and each clock cycle within the period. Section 7 concludes the paper and shows comparison with results presented in [16]. A design of RLS filter on HSLA is shown in Appendix A.

2 Overview on Basic Cyclic Scheduling

Operations in a computation loop can be considered as a set of n *generic tasks* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed N times where N is usually very large. One execution of $\mathcal{T}$ labeled with the integer index $k \geq 1$ is called an *iteration*. Let us denote by T_i^k the k^{th} occur-

rence of the generic task T_i, which corresponds to the execution of statement i in the iteration k. Variable $s_i(k)$, the start time of task occurrence T_i^k, is in the case periodic schedule given as follows

$$s_i(k) = s_i + w \cdot (k-1), \tag{1}$$

where w is the period and s_i denotes the start time of task T_i in the first iteration, i.e. occurrence T_i^1. Therefore, the scheduling problem is to find start time s_i for each task T_i, since $s_i(k)$ can be simply deduced from (1).

Fig. 1.

Figure 1(a) shows an example of a simple computation loop where the processing time of addition/subtraction lasts 9 clock cycles and the processing time of multiplication/square lasts 2 clock cycles. Data dependencies of this problem can be modeled by a directed graph G. Edge e_{ij} from the node i to j is labeled by a couple of integer constants l_{ij} and h_{ij}. Length l_{ij} is equal to the processing time of task T_i. In fact, l_{ij} represents the minimal distance in clock cycles from the start time of the task T_i to the start time of T_j and it is always greater than zero. On the other hand, the height h_{ij} specifies the shift of the iteration index related to the data produced by T_i and read (consumed) by T_j. Therefore, each edge e_{ij} represents a precedence relation constraint of the type

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}. \tag{2}$$

The example in Figure 1(b) shows the data dependence graph of the computation loop shown in Figure 1(a).

The aim of the *Basic Cyclic Scheduling* (BCS) problem [1] is to find a periodic schedule while minimizing the period w. This scheduling problem is simply solved when the number of processors is not limited, i.e. it is sufficiently large. Thereafter the period w is given by the *critical*

circuit c in graph G. This is a circuit $c \in C(G)$ maximizing the ratio

$$w = \max_{c \in C(G)} \frac{\sum_{e_{ij} \in c} l_{ij}}{\sum_{e_{ij} \in c} h_{ij}}. \tag{3}$$

where $C(G)$ denotes the set of cycles in G. Any schedule with a shorter period cannot be feasible assuming that the schedule is periodic.

Since the tasks are repeated every w clock cycles, the periodic schedule is entirely given by the scalar w and the vector of the start times in the first iteration $s = (s_1, s_2, ..., s_n)$. An optimal periodic schedule can be provided in polynomial time, since the time complexity to find critical circuits is $O(n^3 \cdot \log(n))$ [27].

3 Cyclic Scheduling of tasks with Precedence Delays on Limited Number of Dedicated Processors

The Basic Cyclic Scheduling problem, solved in polynomial time, assumes that the number of processors is not limited. When the number of processors is restricted, the problem becomes NP–hard.

3.1 Representation of Pipelining by Precedence Delays

The scheduling problem related to HSLA architecture is even different, since the processors are pipelined and not identical. Some tasks run on one pipelined dedicated processor and the remaining tasks run on an unlimited number of processors. Further, while using FP32 architecture [7], we deal with a set of dedicated processors. Both architectures require a different model than graph G in the previous section where the length of edge is equal to the processing time, i.e. $l_{ij} = p_i$. Therefore, we introduce a model based on, so called, *precedence delays* defined as follows: the length of edge e_{ij} is greater than or equal to the processing time p_i assigned to node T_i, i.e. $l_{ij} \geq p_i$. Therefore, the processor is occupied by task T_i during the processing time p_i,

but task T_j may start at least l_{ij} clock cycles after the start time of T_i. Therefore, related length l_{ij} specifies the *precedence delay* from task T_i to task T_j.

Precedence delays are useful when we consider pipelined processors. The processing time p_i represents the time to feed the processor (i.e. new data can be fed to the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of computation (i.e. the input–output latency). Therefore, the result of a computation is available after l_{ij} clock cycles.

3.2 Representation of Reduced Tasks by Precedence Delays

In the case of an unlimited number of processors, the problem with precedence delays is still solvable using a polynomial algorithm for BCS [1]. But the assumption of an unlimited number of processors is not satisfied for HSLA architecture where some tasks are assigned to one dedicated processor (the addition/subtraction unit). Therefore we formulate it as 1–DEDICATED problem, i.e. *cyclic scheduling of tasks with precedence delays on one dedicated processor* (at the end of this section we show that this problem is NP–hard).

The suggested formulation of the scheduling problem uses a reduction of graph G to a *reduced graph* G'. All nodes (tasks) except the ones assigned to the dedicated processor are reduced. These tasks, running on an unlimited number of processors, will be further called *reduced tasks* and corresponding processors will be called *reduced processors*. Therefore $\mathcal{T}'$, the set of tasks assigned to the dedicated processor, relates to the set of nodes of G'.

Since we consider period w to be constant in each iteration of the scheduling algorithm (explained in Section 4), we can weight edges e_{ij} in graph G by *amplitude* $a_{ij}(w) = l_{ij} - w \cdot h_{ij}$. Then new precedence constraints in graph G' represented by edges e'_{ij} are given by the longest paths in original graph G weighted by $a_{ij}(w)$ (solved e.g. by Floyd's algorithm). There is e'_{ij} (the edge from T_i to T_j in G') of amplitude a'_{ij} if and only if there is a path from T_i to T_j in G such that this path goes only through the reduced nodes. The value of the amplitude a'_{ij} is the

length of the longest path from T_i to T_j in G weighted by a_{ij}.

Fig. 2.

Such reduction allows one to find the schedule for HSLA architecture by solving the 1–DEDICATED problem, since the computation time of the reduced tasks is included in the amplitude of the edges in G'. The reduction performed on graph G in Figure 1(b), is shown on Figure 2 as G'. The addition and subtraction operations (tasks T_1, T_3, T_4, T_5 and T_8) from the example, in Figure 1(b), are processed on the dedicated processor. The reduction technique for 1–DEDICATED problem, depicted above, remains the same for m'–DEDICATED problem.

3.3 Problem Complexity

The problem 1–DEDICATED is NP–hard, since Bratley's scheduling problem $1|r_j, \widetilde{d}_j|C_{max}$ (monoprocessor scheduling with release dates, with deadlines and without precedence constraints) [26] can be polynomially reduced (P–reduced) to it. Therefore, each instance of Bratley's problem can be P–reduced to an instance of our scheduling problem.

Fig. 3.

The P–reduction is shown in Figure 3. The independent task set of Bratley's problem is represented by nodes $T_1, ..., T_n$ and their release dates and deadlines are represented using precedence delays related to a dummy task T_0. The release date r_j of task T_j is the length of the edge e_{0j} from T_0 to T_j and $h_{0j} = 0$. Assuming $s_0 = s_i = 0$, Inequality (2) determines the restriction $s_j \geq r_j$, which is effectively the only restriction given by the release date.

Edges from T_i to T_0 represent deadlines. Let e_{i0} have the height $h_{i0} = 1$ and the length $l_{i0} = w - \widetilde{d}_i + p_i$, where w is equal to value of the maximum deadline (the moment when the next iteration will potentially start). In the same way the deadline restriction, i.e. $s_i + p_i \leq \widetilde{d}_i$, is obtained from (2) for each of these edges assuming $s_0 = s_j = 0$.

We remember that Bratley's problem $1|r_j, \widetilde{d}_j|C_{max}$ was proven to be NP–hard by P–reduction from a 3–PARTITION problem [28]. 1–DEDICATED problem is NP–hard, since each instance of Bratley's problem can be P–reduced to an instance of 1–DEDICATED problem as shown above. Further, m'–DEDICATED problem is NP–hard, since 1–DEDICATED is its subproblem.

4 Solution of Cyclic Scheduling on Dedicated Processors with Precedence Delays by ILP

Due to the NP–hardness it is meaningful to formulate our scheduling problem as a problem of Integer Linear Programming (ILP), since various ILP algorithms solve instances of reasonable size in reasonable time. The period w is assumed to be a constant in this section, since multiplication of two decision variables cannot be formulated as a linear inequality. First we demonstrate this method on 1–DEDICATED problem and then we generalize it to m'–DEDICATED problem.

Let $\hat{s}_i$ be the remainder after division of s_i (the start time of T_i in the first iteration) by w and let $\hat{q}_i$ be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \;\; \hat{s}_i \in \langle 0, w-1 \rangle , \;\; \hat{q}_i \geq 0. \tag{4}$$

This notation divides s_i into $\hat{q}_i$, the index of *execution period*, and $\hat{s}_i$, the number of clock cycles within the execution period. The schedule has to obey the two constraints explained in Subsections 4.1 and 4.2.

4.1 Precedence Constraint

The first constraint is the *precedence constraint* restriction corresponding to Inequality (2). It can be formulated using $\hat{s}$ and $\hat{q}$ as

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) \geq a'_{ij}. \tag{5}$$

Hence, we have ${n'}_e$ inequalities (${n'}_e$ is the number of edges in the reduced graph G'), since each edge represents one precedence constraint.

4.2 Processor Constraint for One Dedicated Processor

Processor constraints are the second type of restrictions. They are related to the dedicated processor restriction, i.e. at maximum one task is executed on one dedicated processor at a given time. The execution period, which is neither in the head nor in the tail of the schedule, contains all tasks even if they are from different iterations. Based on this observation, the processor constraints can be simply formulated using $\hat{s}_i$ (notice that the processor constraints do not depend on $\hat{q}_i$). Two disjoint cases can occur:

(i) In the first case, we consider task T_j to be followed by task T_i (both are from arbitrary iterations) within the execution period.

The corresponding constraint is therefore

$$\hat{s}_i - \hat{s}_j \geq p_j. \tag{6}$$

At the same time, T_i^{k-1} is followed by $T_j^{k'}$, therefore

$$\hat{s}_j - (\hat{s}_i - w) \geq p_i. \tag{7}$$

The conjunction of (6) and (7) into one double–inequality is

$$p_j \leq \hat{s}_i - \hat{s}_j \leq w - p_i. \tag{8}$$

(ii) In the second case, we consider task T_i to be followed by task T_j. To derive constraints for the second case, it is enough to exchange index i with index j in Double–Inequality (8)

$$p_i \leq \hat{s}_j - \hat{s}_i \leq w - p_j. \tag{9}$$

Using simple algebraic operations we derive a form similar to (8)

$$p_j \leq \hat{s}_i - \hat{s}_j + w \leq w - p_i. \tag{10}$$

Exclusive OR relation between the first case and the second case, i.e. either (8) holds or (10) holds, disables one to formulate the problem directly as the ILP model. Inequalities (8) and (10) can be reduced into one double–inequality, while using the binary decision variable $\hat{x}_{ij}$ ($\hat{x}_{ij} = 1$ when T_i is followed by T_j and $\hat{x}_{ij} = 0$ when T_j is followed by T_i)

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i. \tag{11}$$

To derive a feasible schedule on one dedicated processor, Double–Inequality (11) must hold for each unordered couple of two distinct tasks. Therefore, there are $(n'^2 - n')/2$ double–inequalities (where n' is the number of tasks in the reduced graph G'), i.e. there are $n'^2 - n'$ inequalities specifying the processor constraints.

4.3 Objective Function

Using ILP formulation we are able to test the schedule feasibility for a given value of w. In addition we minimize one of the following objective criterions.

4.3.1 Overlap Minimization

One of the simplest objectives is to minimize the iteration overlap by the objective function $\min \sum_{i=1}^{n} \hat{q}_i$. The summarized ILP model, using variables $\hat{s}_i, \hat{q}_i, \hat{x}_{ij}$, is shown in Figure 4. It contains $2n' + (n'^2 - n')/2$ variables and $n'_e + n'^2 - n'$ constraints.

Fig. 4.

4.3.2 Minimization of Iteration Makespan

The advantage of ILP formulation is the possibility to formulate various objective functions. For example, when N, the number of iterations, is small, it may be convenient to minimize the makespan of iteration, by adding one variable c_{max} and n' constraints of type

$$\hat{s}_j + \hat{q}_j \cdot w + p_j \leq c_{max}, \quad \forall T_j \in \mathcal{T}'. \tag{12}$$

Such a reformulated problem not only decides the feasibility of the schedule for the given period w, but if such a schedule exists, it also finds the one with the shortest tail.

4.3.3 Minimization of Data Transfers

Minimization of the data transfers among the tasks (i.e. the number of *intermediate results storages*) leads to simplification of multiplexers used in FPGA design. Such simplification leads to simpler HW design and shortening of the clock cycle. For example, we reached a remarkable FPGA design improvement on DIFFEQ benchmark [29] (a differential equation solver), implemented in Xilinx Virtex II (XC2V6000-6) device. The clock cycle length was shortened by 5.1% and number of FPGA slices was decreased by 5% considering architecture with HSLA library.

In order to minimize the data transfers, we add one slack variable Δ_{ij} to each precedence constraint (5) resulting at

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) + \Delta_{ij} = a'_{ij}. \tag{13}$$

When $\Delta_{ij} = 0$, the intermediate result is passed to the next task without storing in registers or memory. On the other hand, when $\Delta_{ij} > 0$, the memory or register is required. The aim is to minimize the number of $\Delta_{ij} > 0$. Therefore we introduce new binary variable Δ^b_{ij} which is

equal to 1 when $\Delta_{ij} > 0$ and Δ_{ij}^b is equal to 0 otherwise. This relation is formulated as

$$(w \cdot (\hat{q}_{max} + 1)) \cdot \Delta_{ij}^b \geq \Delta_{ij}, \quad \forall e_{ij} \in G, \tag{14}$$

where $(w \cdot (\hat{q}_{max} + 1))$ represents an upper bound on Δ_{ij} and the objective is to minimize $\sum \Delta_{ij}^b$. Meaning of $\hat{q}_{max}$ is explained further in Section 4.8. Such a reformulated problem not only decides the feasibility of the schedule for the given period w, but if such a schedule exists, it also finds the one with minimal data transfers among the tasks.

4.4 Problem Extension by Start Time Related Deadlines

In many practical applications, task T_i must be completed within given time limits. These constraints are usually represented as *deadline* $\widetilde{d}_i$, related to the beginning of the iteration or represented as *start time related deadlines* $\widetilde{d}_{ij}$, related to the start time of another task T_j. For reasons of simplicity, let us suppose $T_i, T_j \in \mathcal{T}'$, i.e. the deadlines are related only to the tasks running on dedicated processors.

A simple example is shown in Figure 5. In this case we want to define maximal time from start time of task T_2 to start time of task T_4 to be 6 clock cycles and from start time of task T_3 to start time of task T_5 to be 4 clock cycles, which is modeled by start time related deadlines depicted by dashed edges in Figure 5. An example of application is a two–channel filter (see technical report [30]). The time delay between the task reading input data (T_1^A resp. T_1^B) and the task producing output data (T_8^A resp. T_8^B) is restricted in each of the filters. Moreover, delay between reading the input data by the first channel and reading the input data by the second channel is restricted as well as production of data by both filters.

Fig. 5.

The ILP model presented in Figure 4 can be simply extended as follows: $\widetilde{d}_i$, task T_i deadline

related to the beginning of the iteration, can be included into the ILP model as

$$\hat{s}_i + \hat{q}_i \cdot w \leq \widetilde{l}_i, \quad \text{where } \widetilde{l}_i = \widetilde{d}_i - p_i \tag{15}$$

and $\widetilde{d}_{ij}$, task T_i deadline related to the start time of task T_j, as

$$(\hat{s}_i + \hat{q}_i \cdot w) - (\hat{s}_j + \hat{q}_j \cdot w) \leq \widetilde{l}_{ij} - w \cdot \widetilde{h}_{ij}, \quad \text{where } \widetilde{l}_{ij} = \widetilde{d}_{ij} - p_i. \tag{16}$$

The shift of iteration index between T_i and T_j is given by $\widetilde{h}_{ij}$. Both $\widetilde{l}_{ij}$ and $\widetilde{h}_{ij}$ are non–negative numbers. Please notice the similarity between the Inequality (16) denoting a start time related deadline and the Inequality (5) denoting a precedence relation. Hence we have $\widetilde{n}_e$ inequalities ($\widetilde{n}_e$ is the number of deadlines) added to ILP model in Figure 4.

A special case of constraint is when task T_j must be scheduled exactly $\widetilde{l}_{ij}$ clock cycles after task T_i. This constraint corresponds, for example, to the access to a memory where task T_i represents the memory reading operation and T_j operation that processes the data. To avoid the use of temporary register for specific data transfer, time delay between these tasks is given by the minimal latency l_{ij} of T_i to T_j. Such constraint is given by (16) where $\widetilde{l}_{ij} = l_{ij}$ and the sign of inequality is substituted by equality.

4.5 Generalization to a Set of Dedicated Processors

The above mentioned formulation of processor constraints for 1–DEDICATED problem allows one to generalize the approach to m'–DEDICATED problem, where m' indicates the number of dedicated processors $\mathcal{P}'$. The role of Double–Inequality (11) is to avoid conflicts of tasks T_i and T_j when the same processor is dedicated to both tasks (e.g. both T_i and T_j perform addition on an exclusive addition unit). Assuming m'–DEDICATED problem, the set of tasks $\mathcal{T}'$ is the conjunction of disjoint subsets $\mathcal{T}'_1, \ldots \mathcal{T}'_d, \ldots \mathcal{T}'_{m'}$. Both tasks T_i and T_j are assigned to the d-th dedicated processor if and only if $T_i \in \mathcal{T}'_d$ and $T_j \in \mathcal{T}'_d$.

The tasks assigned to different processors are not conflicting, i.e. they do not impose any restriction with respect to the feasibility of the schedule. Therefore Double–Inequality (11) is only considered for each couple of tasks assigned to the same dedicated processor.

Formulation of ILP model for m'–DEDICATED problem is obtained from Figure 4 when considering processor constraints $p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i$ only for couples of tasks that may conflict (i.e. $\forall i < j$ and $\exists\ \mathcal{T}'_d$ such that $T_i, T_j \in \mathcal{T}'_d$). Therefore there are $\sum_{d=1}^{m'}(n'^2_d - n'_d)$ inequalities specifying processor constraints of m'–DEDICATED problem, where $n'_d = |\mathcal{T}'_d|$.

4.6 Solution of Problems with Multiprocessor Tasks

The multiprocessor tasks scheduling problem is an extension of m'–DEDICATED problem, where task T_i is associated with a set of m_i processors $\mathcal{P}_i = \{P_i^1, \ldots, P_i^{m_i}\} \subset \mathcal{P}'$. Such problem ocurs, when the arithmetic unit (i.e. processor P_i^1) and the memory unit (i.e. processor P_i^2), are simultaneously required to perform task T_i. Problem with multiprocessor tasks can be solved by ILP model in Figure 4 by a slight modification of the processor constraints (11). The processor constraint between T_i and T_j in 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 problem with multiprocessor tasks, 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.

4.7 Solution of Problems with Changeover Times

This scheduling problem extends m'–DEDICATED problem by partitioning the set of tasks $\mathcal{T}$ into groups $G_1, \ldots, G_r$. If a task T_i from group G_k is scheduled immediately after a task T_j from different group G_l, there is a *changeover time* r_{lk} i.e. task T_i starts at the earliest r_{lk} time units after the finishing time of T_j, otherwise $r_{lk} = 0$.

The monoprocessor scheduling problem with changeover times can be also directly solved by

a modification of the processor constraints (11). Since the processor constraint restricts the overlap only between a couple T_i and T_j then processing times p_i and p_j can be considered different for different couples of tasks, i.e. different processor constraints. Let us consider task T_i belonging to group G_k and task T_j belonging to group G_l. Then processing time p_i used in processor constraint (11) is substituted by $p_i + r_{kl}$ and processing time p_j is substituted by $p_j + r_{lk}$. For tasks from the same group the processor constraint (11) stays the same.

4.8 Elimination of Redundant Processor Constraints

The time requirements to solve the generated ILP model roughly correspond to the number of integer variables, therefore it is meaningful to reduce this number as much as possible. Not all processor constraints are necessary, considering precedence constraints (5). Dealing with cyclic scheduling, one can find tasks from distinct iterations in one period. Therefore we specify *a priory* a parameter $\hat{q}_{max} = max\{\hat{q}_i\}$ given as an additional constraint to the ILP model in Figure 4. The smaller $\hat{q}_{max}$ the shorter the interval is (given as $(\hat{q}_{max} + 1) \cdot w$) in which all tasks from one iteration have to fit.

The necessity of Double–Inequality (11) for the couple of tasks T_i and T_j assigned to the same processor could be decided e.g. while using linear programming (LP) composed of constraints (17), (18) and (19), as explained below. If any feasible solution of this LP problem exists, then T_i and T_j can be in conflict (i.e. can be overlaped) and therefore corresponding double–inequality cannot be eliminated.

Tasks T_i and T_j overlap if and only if task T_i starts before task T_j completes AND task T_j starts before task T_i completes. Therefore when Inequality (18) and Inequality (19) hold, tasks T_i and T_j may overlap by at least one clock cycle assuming integer parameters of tasks. Inequality (17) represents all precedence constraints given by graph G' in terms of start times.

$$s_k - s_l \geq l_{kl} - w \cdot h_{kl}, \qquad \forall e'_{kl} \in G', \tag{17}$$

$$s_i - s_j + \delta \cdot w \leq p_j - 1, \tag{18}$$

$$s_j - s_i - \delta \cdot w \leq p_i - 1, \tag{19}$$

where

$$s_i \in \langle 0, (\hat{q}_{max} + 1) \cdot w \rangle .$$

The LP composed of constraints (17), (18) and (19) is called iteratively for all possible δ, the difference of the iteration index between tasks T_i and T_j within one period. If the polynomial time LP finds a feasible solution for any integer $\delta \in \langle -\hat{q}_{max}, \hat{q}_{max} \rangle$, tasks T_i and T_j can eventually cause conflict on the dedicated processor and then the corresponding Double–Inequality (11) is necessary.

The mentioned LP model operates on integer valued solutions since the system of Inequalities (17), (18), (19) forms a totally unimodular matrix and all input parameters are integers [31].

This elimination is performed in polynomial time and it leads to a decrease in the number of constraints and to a decrease in the number of binary decision variables $\hat{x}_{ij}$.

5 Minimization of the Period

We recall that the goal of cyclic scheduling is to find a feasible schedule with the minimal period w. Therefore, w is not constant as we assumed in the previous section, but due to the periodicity of the schedule it is a positive integer value. Period w^*, the shortest period resulting in a feasible schedule, is constrained by its lower bound w_{lower}, for which the feasibility needs to be tested, and its upper bound w_{upper}, which is feasible if at least one feasible solution exists. Optimal period w^* can be found iteratively by formulating one ILP model, mentioned in the previous section, for each iteration. These iterative calls of ILP do not need to be performed for all w between the lower and upper bounds, but the interval bisection method can be used, since w^* is not preceded by any feasible solution (i.e. no $w \leq w^* - 1$ results in a feasible solution). Therefore, there are at a maximum $\log_2 (w_{upper} - w_{lower})$ iterative calls of ILP.

5.1 Lower Bound

Lower bound w_{lower} is constrained by $w_{processsor\_load} = \max_{d \in \langle 1,m' \rangle} \left\{ \sum_{T_i \in \mathcal{T}'_d} p_i \right\}$, the processor load is given as the maximum sum of the processing times of tasks assigned to one processor. When the problem instance is free of deadlines, we can use Equation (3) to calculate $w_{lower\_unlimited}$, related to the critical circuit of G' (identical to the critical circuit of G). When deadlines are permitted, the problem instance does not need to be feasible even for an unlimited number of processors. The feasibility is constrained by the set of Inequalities (20), specifying precedence relations, and by the set of Inequalities (21), specifying start time related deadlines

$$s_i - s_j \geq l'_{ij} - w \cdot h'_{ij}, \qquad \forall e'_{ij} \in G', \tag{20}$$

$$s_i - s_j \leq \widetilde{l}_{ij} - w \cdot \widetilde{h}_{ij}, \qquad \forall \widetilde{d}_{ij}, \tag{21}$$

where

$s_i \geq 0$ and $w \geq 0$.

Therefore, $w_{lower\_unlimited}$, the minimal feasible period for unlimited number of processors, is calculated using the LP formulation given by (20), (21) while minimizing w. Solution of this LP formulation directly gives the optimal period of BCS problem extended by deadlines, since the period w is variable (due to the absence of the processor constraints).

Finally, w_{lower} is equal to $max(w_{processor\_load}, w_{lower\_unlimited})$, since the processor load in one iteration cannot exceed the period and the schedule of 1–DEDICATED problem or m'–DEDICATED problem cannot be executed faster than the schedule of BCS problem.

5.2 Upper Bound

The space of feasible periods w given by (20), (21) may have its upper bound due to the deadlines (e.g. $w = \infty$ is not feasible for a graph with a circuit of start time related deadlines). Therefore $w_{upper\_unlimited}$, maximal feasible period for unlimited number of processors, is calculated using the LP formulation given by (20), (21) while maximizing w.

The schedule found when calculating $w_{lower\_unlimited}$ allows two tasks of G' to be processed at the same time, which may result in a conflict when both tasks are assigned to the same processor. But we can derive a new schedule by a simple serialization of the conflicting tasks, while obtaining w_{serial_tasks}.

Finally, $w_{upper} = min(w_{serial\_tasks}, w_{upper\_unlimited})$.

5.3 Optimal Schedule Search

Period w^*, the shortest period of 1–DEDICATED or m'–DEDICATED problem is calculated with the interval bisection method. First, we run ILP model for $w = w_{lower}$, if it is feasible then $w^* = w_{lower}$. If not, then we proceed with $w = w_{upper}$. If it is not feasible then this problem has no solution due to deadlines, otherwise we continue with $w = \lfloor (w_{lower} + w_{upper})/2 \rfloor$ and so on.

The above mentioned method gives the feasible schedule of G', if one exists. The corresponding schedule of G is also feasible, since the tasks executed on an unlimited number of reduced processors comply only to the precedence relation constraints that are already included in the precedence delays of G'. Figure 6 shows the schedule of the example depicted in Figure 1(a), where the dedicated processor is shown on the bottom line. Expansion of tasks on reduced processors needs to satisfy precedence relations only; they can be simply executed in an optimal manner (e.g. as soon as possible) since there is an unlimited number of reduced processors available.

Fig. 6.

6 Results

The presented scheduling technique was implemented and run on an Intel Pentium 4 at 2.4 GHz using non-commercial ILP solver tool GLPK [32]. From the resulting schedule, we automat-

ically generated code in Handel-C language [33]. The complexity of the integer linear programs can be estimated using the number of integer variables in the ILP model but each change of the problem instance may lead to a significant change of algorithm computation time. The ILP model for 1–DEDICATED problem contains $2n' + (n'^2 - n')/2$ variables and $n'_e + \widetilde{n}_e + n'^2 - n'$ constraints before the elimination of redundant processor constraints. The efficiency of the elimination depends on the input graph G' and on the given $\hat{q}_{max}$. The ILP model for m'–DEDICATED problem before elimination is even smaller. It contains $n'_e + \widetilde{n}_e + \sum_{d=1}^{m'}(n'^2_d - n'_d)$ inequalities and $2n' + \sum_{d=1}^{m'}(n'^2_d - n'_d)/2$ variables.

6.1 Benchmarks

In this section we show the results on one application implemented in FPGA (see RLS filter in Appendix A) and several benchmarks (for more details see technical report [30]).

One benchmark is the second order wave digital filter (WDF) [34] consisting of eight tasks, where each circuit c has $\sum_{e_{ij} \in c} h_{ij} = 1$. The WDF_2chan is an extension of WDF consisting of two synchronized channels (two identical WDF filters) with 6 start time related deadlines. Another benchmark based on WDF is WDF_reconf considering one reconfigurable unit with symmetric changeover time $r_{ADD,MUL}$=$r_{MUL,ADD}$=1. The second benchmark (van Dongen) by van Dongen [15] consists of 10 tasks where $\sum_{e_{ij} \in c} h_{ij}$ varies from 1 to 8. The third benchmark (Elliptic) is a fifth–order wave digital elliptic filter [35] consisting of 34 tasks with $\sum_{e_{ij} \in c} h_{ij} = 1$. Further, we show a recursive least squares filter (RLS) [10] implementation on HSLA as a real signal processing application. Both, Elliptic and RLS, are scheduled on three different configurations of pipelined arithmetic units.

Tab. 1.

Experiments are summarized in Table 1 where n' denotes the number of tasks after reduction, $\#var$ denotes number of ILP variables, m' denotes the number of dedicated processors. Pa-

rameters of dedicated processors include processing time p_i (i.e. the time to feed the processor) and input–output latency l_{ij} (corresponding to the length of arc in graph G) of arithmetic operations. Parameters of reduced processors include just the latency l_{ij} of operations executed on an unlimited number of reduced processors. The column $\hat{q}_{max}$ is an upper bound of $\hat{q}_i$ given *a priory* and it only has influence on the elimination of redundant inequalities. The columns w_{lower} and w_{upper} are bounds of w, found by polynomial algorithms. The algorithm results are given by the shortest period resulting in a feasible schedule w^*, and by vector s (see Gantt charts in Appendix A and technical report [30]). $\sum \hat{q}_i$ denotes the value of the objective function and illustrates the amount of overlap. The column $\#iter$ denotes the number of iterative calls of ILP and $\#var_{elim}$ denotes the number of eliminated variables of ILP for w^* (the number of eliminated redundant inequalities is twice as big). The time required to compute the optimal solution without elimination, given as a sum of iterative calls of the ILP solver, is shown in the column $CPU\ time$. This time does not include the construction of the ILP model, since it is negligible from the complexity point of view. $CPU\ time_{elim}$ is the time to compute the optimal solution with eliminated redundant inequalities. The shortest period of a feasible schedule while setting $\hat{q}_{max} = 0$, denoted as $w^*_{nonover}$, has been calculated separately in order to illustrate the importance of overlapping. The schedule with $w^*_{nonover}$ disables overlapping of tasks from different iterations on the dedicated processor, but it still permits to complete the work executed in pipeline or on reduced processors (both corresponding to $l_{ij} - p_i$ clock cycles) during the next period.

Experiments WDF (the second order wave digital filter) and WDF_2chan (two identical WDF filters with deadlines) are both executed on one non–pipelined addition unit and one non–pipelined multiplication unit. WDF requires 4 calls of the ILP. Schedule of WDF_2chan is found for $w^* = w_{lower} = w_{processor\_load}$. For WDF_reconf experiment the scheduling algorithm found an nonoverlapped schedule with two reconfigurations within the period.

Experiment van Dongen, containing only addition operations, was executed on one pipelined addition unit. w_{lower}, the lower bound of period, is given by the sum of the tasks processing

times.

Experiments Elliptic$_1$ and Elliptic$_2$ are both executed on one pipelined addition/subtraction unit and an unlimited number of multiplication units. In both of them, there are a lot of redundant inequalities eliminated due to strong precedence relations. In case of the Elliptic$_2$, the optimization required longer CPU time since the unit utilization is higher. Experiment Elliptic$_3$ is an adaptation of Elliptic$_1$ with only one multiplication unit.

Experiment RLS$_1$, executed on one pipelined addition/subtraction unit and unlimited number of multiplication units, corresponds to the application of RLS filter for active noise cancellation using HSLA library (outlined in Appendix A). Experiment RLS$_2$ was executed on the architecture corresponding to FP32 library [7], i.e. one pipelined addition/subtraction unit, one pipelined multiplication unit and one pipelined division unit. Experiment RLS$_3$ was executed on architecture with one pipelined multiplication and one pipelined division unit and an unlimited number of addition/subtraction units.

6.2 Comparison With Binary Method

To compare our method with ILP method using one binary decision variable for each task and each clock cycle within the period (further called binary method) (e.g. [2]), we measure average CPU times for randomly generated instances of 1–DEDICATED problem. In this benchmark elimination of redundant processor constraints is not used.

Randomly generated graph G consists of $\lceil 2/3 \cdot n \rceil$ edges of height $h_{ij} = 0$ and $\lceil 1/2 \cdot n \rceil$ edges of height $h_{ij} > 0$. The height $h_{ij} > 0$ was chosen from a uniform distribution on the interval $\langle 1, 2 \rangle$ (and rounded towards nearest integer). The outdegree of nodes in generated graph G was restricted to 3. All tasks (corresponding to nodes) were associated with one dedicated processor with processing time $p_i = 2$. To demonstrate influence of precedence delay on $CPU\ time$, we performed two benchmarks, one for $l_{ij} = 4$ (see Figure 7) and one for $l_{ij} = 6$ (see Figure 8).

Fig. 7.

Fig. 8.

For each total number of tasks n, 500 test instances have been randomly generated. Using a logarithmic scale on horizontal axis, Figures 7(a) and 8(a) show for a given amount of time, how many instances (in %) have been solved. Average $CPU\ time$ in dependence on number of nodes (tasks) n is shown in Figures 7(b) and 8(b). The results show considerably lower time requirements of our method. From the comparison between Figures 7 and 8 it follows that for instances with greater precedence delays the two methods differ even more.

7 Conclusions

This paper presents an ILP–based cyclic scheduling method used to optimize computation speed of loops running on architectures including dedicated processors (each dedicated processor executes a disjoint set of tasks) and unlimited number of processors (these processors execute a set of operations disjoint with the ones executed on dedicated processors).

Our approach uses the graph reduction when operations executed on an unlimited number of processors are transformed to precedence delays. In order to exploit the graph structure we use polynomial time algorithms to simplify the optimization process. Namely, either graph algorithms calculating critical circuit or LP is used to find the lower bound of the optimal period. Further, we use LP to eliminate redundant inequalities and binary decision variables. Then an optimal periodic solution is searched iteratively using interval bisection method.

As shown in the experiments, w^* is very close to w_{lower}, since the critical circuit represents the main constraint for cyclic scheduling. This effect is quite important (see columns w^* and w_{lower} in Table 1), therefore it is a question whether simple incrementing of period, when starting from w_{lower}, is not better than interval bisection method.

The advantage of the ILP model presented in Figure 4 in comparison with common ILP models used for similar problems is that the number of variables is independent of the period length. The solution to one dedicated processor (called single functional unit) problem, shown in [16], also does not depend on the period length, but our solution is more efficient. Namely, the solution in [16] requires in addition binary variables induced by linearization of the multiplication by the period w. In our solution we search the value of w iteratively within a finite number of steps, and therefore w appears as a constant in the ILP model and it does not introduce any additional decision variable. Consequently, before the elimination of the redundant processor constraints our solution has less decision variables.

Our method of algorithm modeling, transformation and scheduling for FPGAs is fully automated. Therefore, it is easily incorporated in design tools (e.g. Handel C code generation from the resulting schedule) while representing considerable simplification for rapid prototyping. Moreover, we have shown how multiprocessor tasks, changeover times and minimization of data transfers can be considered in the method.

The results of our scheduling method applied to the RLS filter design are better than the ones achieved by experienced FPGA programmer (see results in [10]). For a given sampling period of $1/44100$ second, the filter was able to process 129 iterations by our method, in contrast to manual design achieving 75 iterations. This acceleration by 70% is mainly due to the schedule overlap (operations belonging to different iterations are interleaving) and pipelining, which is rather difficult to achieve in the manual design.

8 Acknowledgement

This work was supported by the Ministry of Education of the Czech Republic under research programme MSM6840770038 and by the ARTIST2 Network of Excellence on Embedded Systems Design IST-004527. The authors would like to thank the anonymous referees for providing many invaluable comments and suggestions that lead to significant improvement of this paper.

Appendix

A Application

RLS filter's algorithm is a set of equations (see the inner loop in Figure 9(a) solved in an inner and outer loop. The outer loop is repeated for each input data sampled every $1/44100$ second. The inner loop iteratively processes the sample up to the N-th iteration. The quality of filtering increases with increasing N. All iterations of the inner loop need to be finished before the end of the sampling period when the output data sample is generated and the new input data sample starts to be processed.

The specific inner loop of the RLS filter is shown in Figure 9(a) where the task labels are next to the arithmetic operations. Figure 9(b) shows the corresponding G', the graph after reduction. The schedule presented in Figure 10 was found by the first call of the ILP model for $w^* = 26$ (the same period as w_{lower} given by the critical circuit).

Fig. 9.

The real–time demo application implemented on a Celoxica rc200e development board (Chip xc2v1000–4, design clock 50 MHz) using 19-bit logarithmic number system arithmetic HSLA, reached 129 iterations of the inner loop on a sampling frequency 44100 Hz. More details on the application can be found in [10].

Fig. 10.

B List of Variables

e_{ij}	edge e_{ij} from the node i to j
h_{ij}	height of e_{ij}
i, j	indices
k	iteration index
l_{ij}	length of e_{ij}
$\widetilde{d}_{ij}$	start time related deadline
m'	number of processors (arithmetic units) after reduction
n	number of tasks
n'	number of tasks after reduction
p_i	processing time
$\hat{q}_i$	the index of the execution period
r_{lk}	changeover time
s_i	start time of task T_i in the first iteration
$\hat{s}_i$	the start time within the execution period
t	time
w	period
w^*	optimal period
$\hat{x}_{ij}$	binary decision variable of ILP model
G	graph
G'	reduced graph
G_k	group of tasks
N	number of iterations
$\mathcal{P}_i$	set of dedicated processors
$\mathcal{T}$	set of tasks
$\mathcal{T}_d$	set of tasks assigned to the d-th dedicated processor
T_i	task i
δ	difference between iteration indexes
Δ_{ij}	slack variable of ILP model
Δ_{ij}^b	binary decision variable of ILP model

References

[1] C. Hanen, A. Munier, A study of the cyclic scheduling problem on parallel processors, Discrete Applied Mathematics 57 (1995) 167–192.

[2] S. L. Sindorf, S. H. Gerez, An integer linear programming approach to the overlapped scheduling of iterative data–flow graphs for target architectures with communication delays, in: PROGRESS 2000 Workshop on Embedded Systems, 2000, pp. 95–104.

[3] A. Munier, The complexity of a cyclic scheduling problem with identical machines and precedence constraints, European Journal of Operational Research 91 (1996) 471–480.

[4] R. Govindarajan, E. R. Altman, G. R. Gao, A framework for resource–constrained rate–optimal software pipelining, IEEE Transactions on Parallel and Distributed Systems 7 (11) (1996) 1133–1149.

[5] A. Heřmánek, Z. Pohl, J. Kadlec, FPGA implementation of the adaptive lattice filter, in: Field–Programmable Logic and Applications, Vol. 2778 of Lecture Notes in Computer Science, Springer, Berlin, 2003, pp. 1095–1099.

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

[7] Celoxica Ltd., Platform Developers Kit: Pipelined Floating-point Library Manual (2004).
URL `http://www.celoxica.com`

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

[9] R. Bartosinski, M. Daněk, P. Honzík, 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, Monterey, California, USA, 2005, pp. 274–274.

[10] P. Šůcha, Z. Pohl, Z. Hanzálek, Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit, in: 10th IEEE Real–Time and Embedded Technology and Applications Symposium, 2004, pp. 404–412.

[11] A. Heřmánek, J. Schier, P. Šůcha, Z. Pohl, Z. Hanzálek, Optimization of finite interval CMA implementation for FPGA, in: IEEE Workshop on Signal Processing Systems, 2005, pp. 75–80.

[12] B. D. de Dinechin, Simplex scheduling: More than lifetime–sensitive instruction scheduling, Proceedings of the International Conference on Parallel Architecture and Compiler Techniques (1994).

[13] M. Lam, Software pipelining: an effective scheduling technique for VLIW machines, in: PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation, 1988, pp. 318–328.

[14] B. R. Rau, C. D. Glaeser, Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing, Proceedings of the 20th Annual Workshop on Microprogramming and Microarchitecture (1981).

[15] V. H. Dongen, G. R. Gao, A polynomial time method for optimal software pipelining, Lecture Notes in Computer Science, Springer–Verlag, ISBN 3-540-55895-0 (1992) 613–624.

[16] D. Fimmel, J. Müller, Optimal software pipelining under resource constraints, Journal of Foundations of Computer Science 12 (6) (2001) 697–718.

[17] V. H. Allan, R. B. Jones, R. M. Lee, S. J. Allan, Software pipelining, in: ACM Computing Surveys, Vol. 27(3), 1995, pp. 367–432.

[18] L.-F. Chao, A. LaPaugh, E.-M. Sha, Rotation scheduling: a loop pipelining algorithm, IEEE Transaction on Computer–Aided Design of Integrated Circuits and Systems 16 (3) (1997) 229–239.

[19] P. Chrétienne, List schedules for cyclic scheduling, in: Discrete Applied Mathematics, Vol. 94(1–3), Elsevier Science Publishers B. V., 1999, pp. 141–159.

[20] P. Chrétienne, On Graham's bound for cyclic scheduling, Parallel Computing 26 (9) (2000) 1163–1174.

[21] B. R. Rau, Iterative modulo scheduling: an algorithm for software pipelining loops, in: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, New York, NY, USA, 1994, pp. 63–74.

[22] J. M. Codina, J. Llosa, A. Gonzlez, A comparative study of modulo scheduling techniques, in: International Conference on Supercomputing (ICS), 2002, pp. 97–106.

[23] W. S. K.-I. Kum, Combined word-length optimization and high-level synthesis of digital signal processing systems, IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems 20 (8) (2001) 921–930.

[24] I. Kazuhito, E. Lucke, K. Parhi, ILP based cost–optimal DSP synthesis with module selection and data format conversion, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 6 (1999) 582–594.

[25] D. Fimmel, J. Müller, A flow graph formulation of optimal software pipelining, in: In Proceedings of the Third International Conference on Parallel and Distributed Computing, Applications and Technologies PDCAT, 2002, pp. 422–429.

[26] J. Błazewicz, K. Ecker, G. Schmidt, J. Węglarz, Scheduling Computer and Manufacturing Processes, 2nd Edition, Springer-Verlag, Heidelberg, Germany, 2001.

[27] K. Ito, K. K. Parhi, Determining the minimum iteration period of an algorithm, Journal of VLSI Signal Processing Systems 11 (3) (1995) 229–244.

[28] J. K. Lenstra, A. R. Kan, P. Brucker, Complexity of machine scheduling problems, Annals of Operations Research (1977) 343–362.

[29] E. G. P. Paulin, J. Knight, Hal: A multi-paradigm approach to automatic data path synthesis, in: 23rd IEEE Design Automation Conf, Las Vegas, 1986, pp. 263–270.

[30] P. Šůcha, Z. Hanzálek, Benchmarks for cyclic scheduling from class of digital signal processing algorithms, Tech. rep., CTU FEL DCE, Prague, http://dce.felk.cvut.cz/sucha/articles/cyclicschedulingbenchmarks.pdf (2007).

[31] H. P. Williams, Model building in mathematical programming, 4th Edition, John Wiley & Sons, New York, 1999.

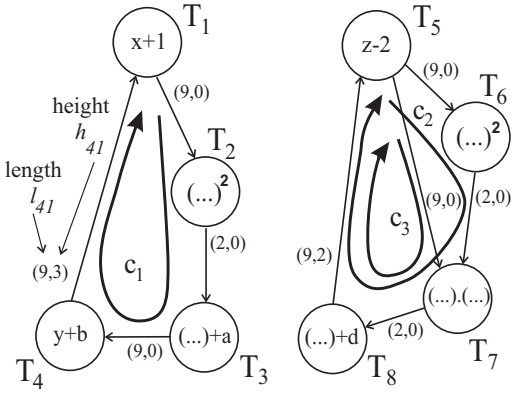
[32] A. Makhorin, GLPK (GNU Linear Programming Kit) Version 4.4 (2004).
URL `http://www.gnu.org/software/glpk/`

[33] Celoxica Ltd., Handel-C Language Reference Manual (2004).
URL `http://www.celoxica.com`

[34] A. Fettweis, Wave digital filters: theory and practice, Proceedings of the IEEE 74 (1986) 270–327.

[35] E. Bonsma, S. Gerez, A genetic approach to the overlapped scheduling of iterative data–flow graphs for target architectures with communication delays, in: ProRISC Workshop on Circuits, Systems and Signal Processing, 1997, pp. 75–84.

for k=1 **to** N **do**
y(k)=(x(k-3) + 1)^2 + a
x(k)=y(k) + b
z(k)=(z(k-2) - 2)^3 + d
end



(a) (b)

Fig. 1. (a) An example of a computation loop. (b) Corresponding data dependency graph G. The operation to the power three is the product of the power two (tasks T_6) and the power one (tasks T_7). G contains three cycles c_1, c_2 and c_3 with average cycle times $\{29/3, 22/2, 20/2\}$. Period $w = 11$ is obtained with respect to the critical circuit c_2.

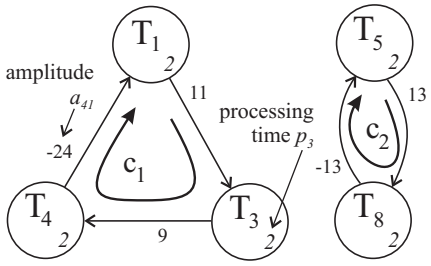


Fig. 2. Reduced graph G' of the graph from Figure 1(b).

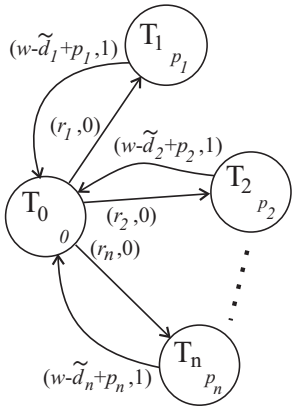


Fig. 3. Polynomial reduction of Bratley's problem $1|r_j, \tilde{d}_j|C_{max}$ to 1–DEDICATED. Each independent task of the set $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ is linked with a dummy task T_0 using precedence delays to specify the task's release date and deadline.

$$\min \sum_{i=1}^{n} \hat{q}_i$$

subject to

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq a'_{ij}, \qquad \forall e'_{ij} \in G'$$

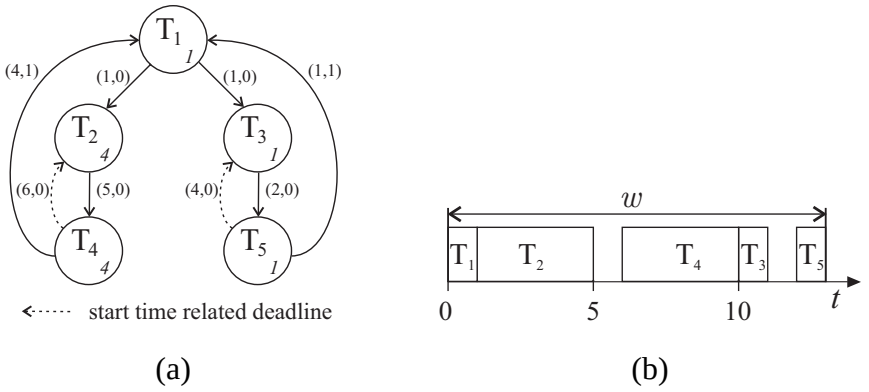
$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i\,, \qquad \forall i, j \text{ where } i < j \text{ and } T_i, T_j \in \mathcal{T}'$$

where

$\hat{s}_i \in \langle 0, w-1 \rangle$, $\hat{q}_i \geq 0$, $\hat{x}_i \in \langle 0, 1 \rangle$,

$\hat{s}_i, \hat{q}_i, \hat{x}_{ij}$ are integers.

Fig. 4. ILP model for 1–DEDICATED problem.



(a) (b)

Fig. 5. (a) An example with start time related deadlines. (b) Optimal schedule with $w = 13$.

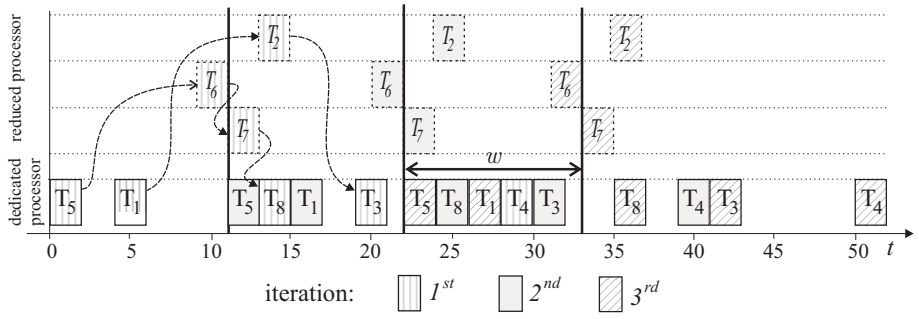
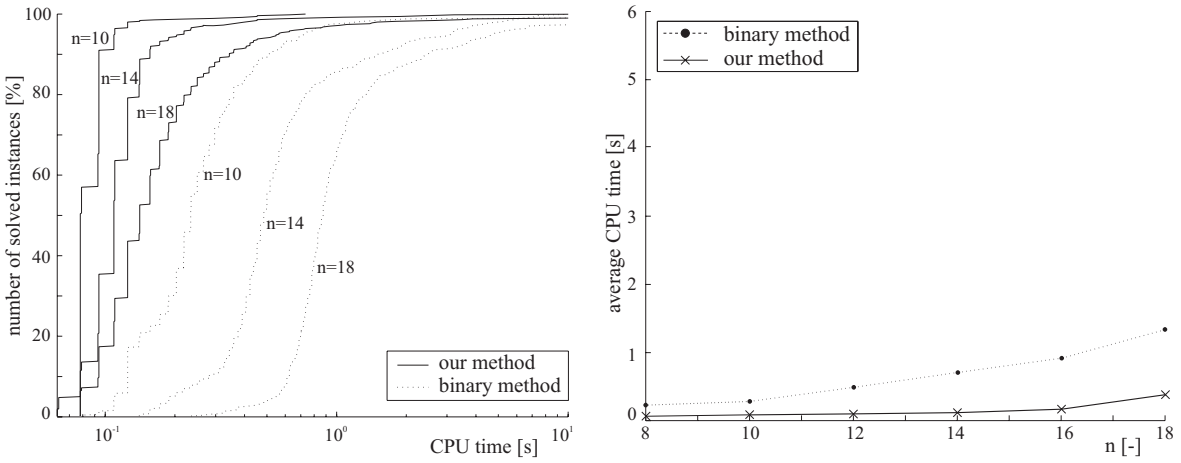
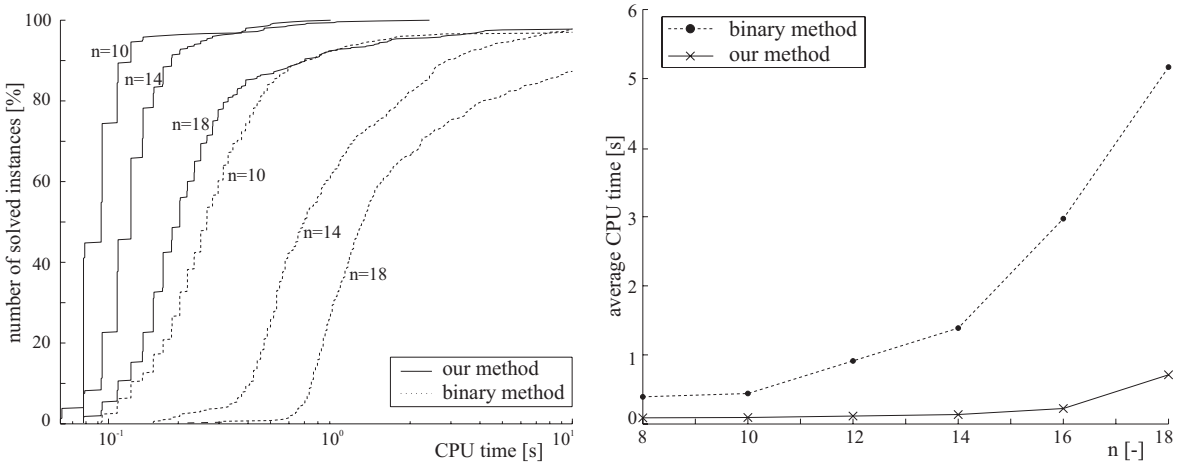


Fig. 6. The schedule ($w^* = w_{lower} = 11$) including expansion to reduced processors.



(a) (b)

Fig. 7. Comparison with binary method ($p_i = 2$ and $l_{ij} = 4$). (a) Number of solved instances depending on CPU time. (b) Average CPU time.



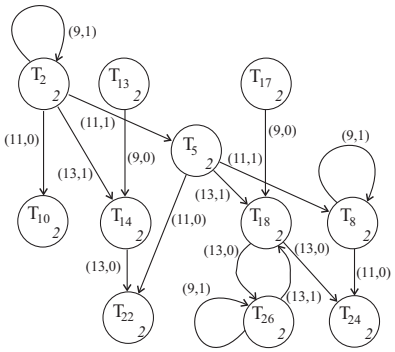
(a) (b)

Fig. 8. Comparison with binary method ($p_i = 2$ and $l_{ij} = 6$) (a) Number of solved instances depending on CPU time. (b) Average CPU time.

for k=1 **to** N **do**

T_2,T_1:	$\eta(k)$	$= \eta(k-1) - (\gamma^f_{old}(k) * \psi_{old}(k-1))$
T_3:	$f(k)$	$= \gamma_{old}(k-1) * \eta(k-1)$
T_5,T_4:	$\psi(k)$	$= \psi_{old}(k-1) - (\gamma^b_{old}(k) * \eta(k-1))$
T_6:	$b(k)$	$= \gamma(k-1) * \psi(k-1)$
T_8,T_7:	$\alpha(k)$	$= \alpha(k-1) - (\kappa_{old}(k) * \psi(k-1))$
T_{10},T_9:	$\gamma^f(k)$	$= \gamma^f_{old}(k) + (bn_{old}(k) * \eta(k))$
T_{13},T_{11},T_{14},T_{12}:	$F(k)$	$= (\nu + (\lambda * F_{old}(k))) + (f(k) * \eta(k-1))$
T_{17},T_{15},T_{18},T_{16}:	$B(k)$	$= (\nu + (\lambda * B_{old}(k))) + (b(k) * \psi(k-1))$
T_{19}:	$f^n(k)$	$= f(k)/F(k)$
T_{20}:	$b^n(k)$	$= b(k)/B(k)$
T_{22},T_{21}:	$\gamma^b(k)$	$= \gamma^b_{old}(k) + (f^n(k) * \psi(k))$
T_{24},T_{23}:	$\kappa(k)$	$= \kappa_{old}(k) + (b^n(k) * \alpha(k))$
T_{26},T_{25}:	$\gamma(k)$	$= \gamma(k-1) - (b^n(k) * b(k))$

end



(a) (b)

Fig. 9. (a) The inner loop of RLS filter. (b) Corresponding reduced graph G.

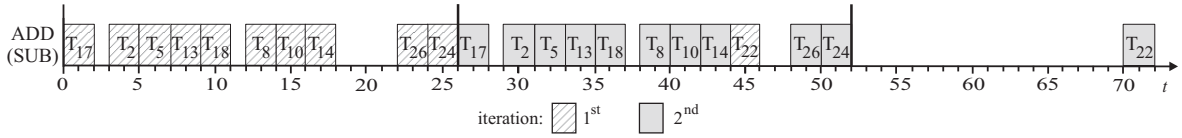


Fig. 10. Two iterations of the schedule of the RLS_1 filter on one dedicated processor ($w^* = 26$).

Table 1
Overview of the Experimental Results

Benchmark	n'	$\#var$	m'	parameters of processors		$\hat{q}_{max}$	w_{lower}	w_{upper}	w^*	$\sum_{i=1}^{n} \hat{q}_i$	$\#iter$	$\#var_{elim}$	$CPU\ time$	$CPU\ time_{elim}$	$w^*_{nonover}$
	[-]	[-]	[-]	dedicated proc.	reduced proc.	[-]	[clk]	[clk]	[clk]	[-]	[-]	[-]	[s]	[s]	[-]
WDF	8	32	2	p_{ADD}=l_{ADD}=1, p_{MUL}=l_{MUL}=3	-	1	6	12	8	2	4	4	0.035	0.02	10
WDF_2chan	16	104	2	p_{ADD}=l_{ADD}=1, p_{MUL}=l_{MUL}=3	-	1	12	21	12	3	1	26	1.06	0.53	16
WDF_reconf	8	39	1	p_{ADD}=l_{ADD}=1, p_{MUL}=l_{MUL}=3 $r_{ADD,MUL}$=$r_{MUL,ADD}$=1	-	1	6	15	14	0	5	5	4.29	3.23	14
van Dongen	10	65	1	p_{ADD}=1, l_{ADD}=2	-	2	10	13	10	1	1	1	0.73	0.73	11
$Elliptic_1$	26	377	1	p_{ADD}=1, l_{ADD}=9	l_{MUL}=2	1	96	110	97	1	4	246	31.70	0.83	98
$Elliptic_2$	26	377	1	p_{ADD}=2, l_{ADD}=9	l_{MUL}=2	1	96	124	98	1	6	236	115.98	27.94	100
$Elliptic_3$	34	421	2	p_i=1, l_{ADD}=9, l_{MUL}=2	-	1	96	114	97	1	5	268	301.80	30.96	98
RLS_1	11	77	1	p_{ADD}=2, l_{ADD}=9	l_{MUL}=l_{DIV}=9	1	26	34	26	1	1	1	0.48	0.53	28
RLS_2	26	186	3	p_i=1, l_{ADD}=10, l_{MUL}=7, l_{DIV}=28	-	1	69	79	69	0	1	4	0.12	0.33	69
RLS_3	15	109	2	p_i=1, l_{MUL}=3, l_{DIV}=6	l_{ADD}=1	1	17	27	17	0	1	3	1.27	0.23	17