

# Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit

Přemysl Šůcha<sup>1</sup>, Zdeněk Pohl<sup>2</sup> and Zdeněk Hanzálek<sup>1</sup>

<sup>1</sup>Department of Control Engineering, Faculty of Electrical Engineering  
Czech Technical University in Prague  
{suchap,hanzalek}@fel.cvut.cz

<sup>2</sup> Department of Signal Processing, Institute of Information Theory and Automation  
xpohl@utia.cas.cz

## Abstract

This paper presents a scheduling technique for library of arithmetic logarithmic modules for FPGA illustrated on RLS filter for active noise cancellation. The problem under assumption is to find an optimal periodic cyclic schedule satisfying the timing constraints. The approach is based on transformation to monoprocessor cyclic scheduling with precedence delays. We prove that this problem is NP-hard and we suggest solution using Integer Linear Programming where moreover iteration overlapping or  $C_{max}$  can be minimized. Results of optimized application show the utility of this approach.

**Keywords:** Cyclic scheduling, monoprocessor, iterative algorithms, integer linear programming, FPGA.

## 1 Introduction

This paper deals with automatic parallelisation of algorithms typically found in control and signal processing applications. Dynamic properties of these applications are characterized by their time constants. Due to Shannon's theorem the length of the sampling period needs to be at maximum a half of the shortest time constant of the system under control. These applications have natural real-time requirements since the algorithm release date is at the beginning of the sampling period and the deadline is at the end of the sampling period. Advanced applications usually require quite complex algorithms typically given by the set of recurrent equations (e.g. Recursive Least Squares identification used for adaptive control and filtering).

Such algorithm can be implemented as a computation loop performing an identical set of operations repeatedly. Each repetition of the loop is called *iteration*. A parallel implementation of

the loop implies that each operation of the loop is mapped on a hardware unit at a given time instant, therefore the scheduling theory can be used to find start times of these operations.

*Cyclic scheduling* deals with a set of operations (generic tasks) that have to be performed infinitely often [12]. This approach is also applicable if number of repetition of loop is large enough. A schedule is called *nonoverlapped* if all operations belonging to the same iteration have to finish before the next operations of the next iteration can start. If operations belonging to different iterations can execute simultaneously, the schedule is called *overlapped* [23]. Overlapped schedule can be more effective especially if hardware units are pipelined. The *periodic schedule* is a schedule of one iteration is repeated with a fixed time interval called *period*. The aim is then to find a periodic schedule with minimum period [12, 20, 8, 11, 21].

If the number of processors is not limited, a periodic schedule can be build in polynomial time [12, 8]. It is proven that for fixed number of processors the problem becomes NP-hard. [20] Anyhow if all tasks have unit processing times, several special cases with polynomial time complexity are available [20]. Another approaches are based on heuristics [11, 12, 7] or approximation list scheduling algorithms [5, 6]. Some solutions are based on branch and bound techniques or integer linear programming [11, 15, 3, 23, 4, 22].

The hardware architecture under consideration is based on library of arithmetic logarithmic modules implemented in FPGA. This library contains pipelined addition/subtraction unit which is usually present only once on contrary to arbitrary number of multiplication/division/square root units. Therefore our scheduling problem is different from the ones presented above.

In this paper we propose an optimal cyclic scheduling method based on integer linear programming (ILP). Presented solution is based on property of logarithmic arithmetic library allow-

ing to formulate *monoprocessor* (pipelined addition/subtraction unit) *cyclic scheduling* problem for the set of tasks constrained by the *precedence delays* (representing pipelining and processing time of tasks executed on unlimited number of multiplication/division/square root units). Unlike the most frequent ILP models we suggest the model where the number of variables does not depend on the period length.

From the time complexity point of view presented scheduling problem is NP-hard that is shown in chapter 4.

This paper is organized as follows. Section 2 describes motivation application (RLS filter for active noise cancellation) implemented on FPGA using HSLA (High-Speed Logarithmic Arithmetic). In section 3 Basic Cyclic Scheduling (BCS) problem is explained assuming unlimited number of processors. It can be skipped by the reader familiar with this concept. Next section presents our original contribution - formulation of the scheduling problem suited for applications using HSLA. It is shown that the problem is NP-hard. Optimal solution of this problem using iterative calls of ILP is presented in section 5. Efficiency of this solution is based on calculation of BCS finding lower and upper bound of the schedule period. Section 6 presents the results - RLS filter automatic parallelisation is derived as one instance of formulated scheduling problem (monoprocessor cyclic scheduling with precedence delays). This chapter includes also resulting filter parameters, so that our solution is comparable to other technologies (e.g. DSPs).

## 2 RLS filter - motivation example

The studied problem is motivated by application of RLS (Recursive Least Squares) filter for active noise cancellation [13] (illustrated in Figure 1). The filter uses HSLA (High-Speed Logarithmic Arithmetics), library of arithmetic logarithmic modules for FPGA [18]. The logarithmic number system arithmetic is an alternative approach to floating-point arithmetic. A real number is represented as the fixed point value of logarithm to base 2 of its absolute value. An additional bit indicates the sign. Multiplication, division and square root are implemented as fixed-point addition, subtraction and right shift therefore they are executed very fast on a few gates. Contrariwise addition and subtraction require more complicated evaluation using look-up table with second order interpolation. Addition and subtraction require more hardware elements on FPGA, 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 roots units can be nearly by arbitrary.

RLS filter's algorithm is a set of equations (see inner loop in Figure 11) solved in an inner and outer loop. The outer loop is repeated for each input data sample each  $1/44100$  seconds. The inner loop iteratively processes the sample up to the  $N$ -th iteration ( $N$  is the filter order). Quality of filtering increases with increasing filter order.  $N$  iterations of inner loop need to be finished before the end of the sampling period when output data sample is generated and new input data sample starts to be processed.

The scheduling method shown below applies for cyclic scheduling on the architectures consisting of one dedicated processor (like one pipelined addition/subtraction unit in HSLA) performing a given set of tasks and arbitrary number of processors performing disjunctive set of tasks (like multiplication, division and square root simply implemented on separate gates in HSLA). The tasks are constrained by precedence relations corresponding to the algorithm data dependencies. The optimization criterion is related to the minimization of the cyclic scheduling period  $w$  (like in RLS filter application the execution of the maximum number of inner loop periods  $w$  within the given sampling period increases the filter quality).

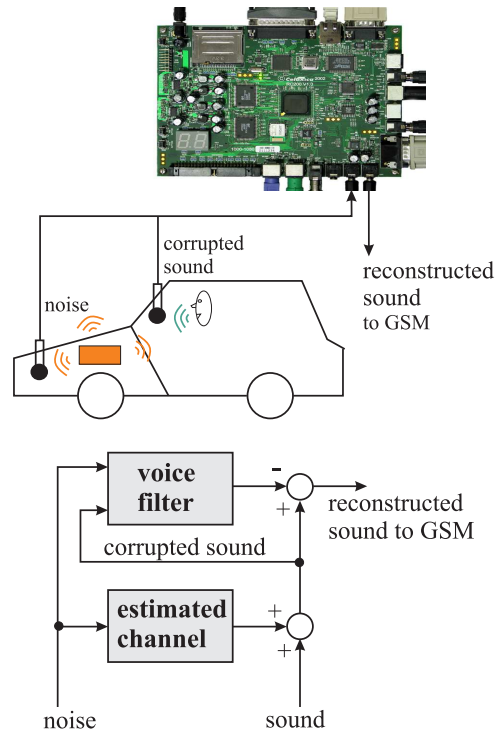


Figure 1: Illustration of active noise cancellation - adaptive RLS filter estimates parameters of changing channel in order to reconstruct original clear sound.

### 3 Basic Cyclic Scheduling

Operations in a computation loop can be considered as a set of  $n$  generic tasks  $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$  to be performed  $N$  times where  $N$  is usually very large. Each performance of  $\mathcal{T}$  marked with integer index  $k \geq 1$  is called *iteration*. Let us denote by  $\langle i, k \rangle$  the  $k^{\text{th}}$  occurrence of the generic task  $T_i$ , which corresponds to the execution of statement  $i$  in iteration  $k$ . The scheduling problem is to find a start time  $s_i(k)$  of every occurrence  $\langle i, k \rangle$  [12]. Figure 2 shows example of simple computation loop with corresponding times of operations.

```

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

```

operation (task)	proc. time $p$
+, -	9
*, /, $^2$ , $\sqrt{\quad}$	2

Figure 2: Example of a recurrent loop and corresponding processing times.

Data dependencies of this problem can be modeled by a directed graph  $G$ . Edges  $e_{ij}$  from the node  $i$  to  $j$  is weighted by couple of integer constants  $l_{ij}$  and  $h_{ij}$ . Length  $l_{ij}$  is equal to  $p_i$ , the processing time of task  $T_i$ . In fact  $l_{ij}$  represents minimal distance in clock cycles from start time of task  $T_i$  to start time of  $T_j$  and it is always greater than zero. On the other hand the height  $h_{ij}$  specifies a shift of the iteration index related to the data produced by  $T_i$  and consumed by  $T_j$ . Therefore each edge  $e_{ij}$  represents the set of  $N$  relation constraints of the type:

$$s_i(k) + l_{ij} \leq s_j(k + h_{ij}), \quad \forall k \in \langle 1, N \rangle \quad (1)$$

Figure 3 shows data dependences graph of a computation loop shown in Figure 2.

Purpose of *Basic Cyclic Scheduling* (BCS) [12] is to find periodic schedule (with a period  $w$ ) while minimizing the  $C_{max}$  of schedule. The problem can be formulated as minimization of average cycle time ( $C_{max}$  divided by  $k$ , the number of iterations). When assuming large number of iterations, the average cycle time minimization can be formulated as minimization of  $w$ , since:

$$w = \lim_{k \rightarrow \infty} \frac{\max_{T_i \in \mathcal{T}} (s_i(k) + p_i)}{k} \quad (2)$$

The 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

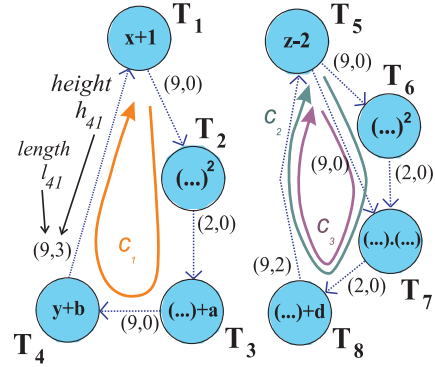


Figure 3: Graph  $G$ , representing data of the computation loop, contains three cycles  $c_1$ ,  $c_2$  and  $c_3$  with average cycles times  $\{29/3, 22/2, 20/2\}$ . With respect to the critical circuit is  $c_2$  with  $w = 11$ .

by *critical circuit*  $c$  in the graph  $G$ . This is the circuit  $c \in C(G)$  maximizing the ratio:

$$w(G) = \max_{c \in C(G)} \frac{\sum_{T_i \in c} l_{ij}}{\sum_{T_i \in c} h_{ij}} \quad (3)$$

Any schedule with shorter period can't be feasible assuming that the schedule of iterations is identical. The start time of the tasks  $T_i$  in iteration  $k$  is given as follows:

$$s_i(k) = s_i + w \cdot (k - 1) \quad (4)$$

where  $s_i$  denotes start time of the task  $T_i$  in the first iteration, i.e. occurrence  $\langle i, 1 \rangle$ . Using equation (4), the set of  $N$  precedence constraint (1) can be reformulated as one inequality:

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij} \quad (5)$$

Since the tasks are repeated every  $w$  time units the periodic schedule is entirely given by scalar  $w$  and vector of start times in the first iteration  $s = (s_1, s_2, \dots, s_n)$ . An optimal periodic schedule can be provided in polynomial time, since the time complexity to find a critical circuits is  $O(n^3 \cdot \log(n))$  and each task in the first iteration is allocated to processors with respect to constraint (5). Figure 4 shows feasible periodic schedule for the recurrent loop given in Figure 2 when  $N = 3$ . Three iterations (each distinguished by a different hatch) are executed in three periods and remaining time 33-51 corresponds to the schedul tail.

Please notice that the schedule shown in Figure 4 is optimal with respect to  $C_{max}$  but not optimal with respect to the number of processors (e.g. task  $T_5$  can be scheduled on processor  $B$  together with the task  $T_2$ ).

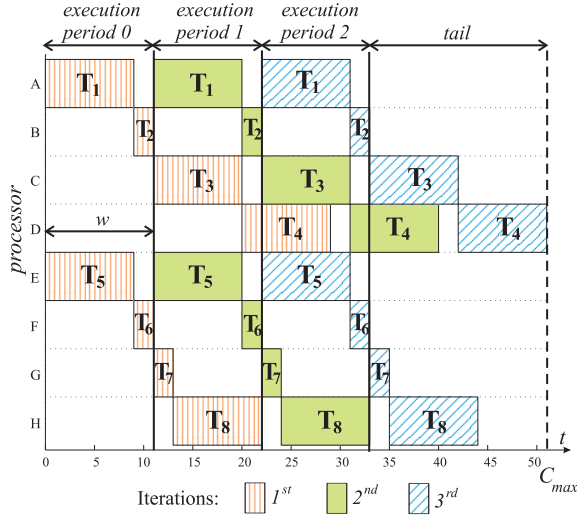


Figure 4: Feasible periodic schedule ( $w = 11$ ) optimal with respect to minimal  $w$ .

## 4 Dedicated processor

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 (polynomial algorithms are known [20, 8] only for some special sub cases). The scheduling problem related to our motivation example is even different since some tasks run on one pipelined dedicated processor and remaining tasks run on arbitrary number of processors. This problem requires a different model than the graph  $G$  in the previous chapter where  $l_{ij} = p_i$ , therefore we introduce the model based on so called *precedence delays*.

In this new model the length of the edge  $e_{ij}$  is greater or equal to the processing time  $p_i$  assigned to the node  $T_i$ . Therefore the processor is occupied by the task  $T_i$  during processing time  $p_i$ , but the task  $T_j$  may start at least  $l_{ij}$  time units after the start time of  $T_i$ . Therefore related length  $l_{ij}$  specifies the *precedence delay* from the task  $T_i$  to the task  $T_j$ .

The precedence delays are useful when we consider pipelined processors. The processing time  $p_i$  represents the time to feed the processor and length  $l_{ij}$  represents the time of computation. Therefore the result of computation is available after  $l_{ij}$  time units.

In the case of unlimited number of processors, the problem with precedence delays is still solvable using polynomial BCS. But assumption of unlimited number of processors is not satisfied for our application where some tasks are running on one dedicated processor (the addition/subtraction unit of HSLA). This problem can be formulated as *monoprocessor cyclic scheduling with precedence*

*delays* (in the end of this chapter we show that this problem is NP-hard).

Suggested formulation is based on the following reduction of the graph  $G$  to  $G'$  while using calculation of the longest paths (solved e.g. by Floyd's algorithm). All nodes (tasks) except the ones running on dedicated processor are eliminated. Therefore tasks running on dedicated processor constitute the nodes of  $G'$ . There is  $e'_{ij}$  (the edge from  $T_i$  to  $T_j$  in  $G'$ ) of height  $h'_{ij}$  if and only if there is the path from  $T_i$  to  $T_j$  in  $G$  of height  $h'_{ij}$  such that this path goes only through eliminated nodes (tasks scheduled on arbitrary number of processors). The value of length  $l'_{ij}$  is the longest path from  $T_i$  to  $T_j$  in  $G$  of height  $h'_{ij}$ .

Such reduction allows to find the schedule for our application by solving the problem of monoprocessor cyclic scheduling with precedence delays. The operations addition and subtraction from example on Figure 3 are all processed on the dedicated processor. Reduction performed on graph  $G$  from illustration example is in Figure 5.

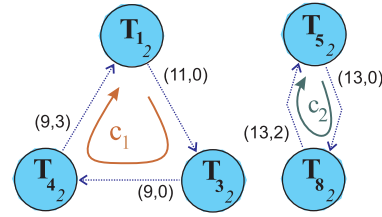


Figure 5: Reduced graph  $G'$ .

### 4.1 Problem Complexity

The problem of monoprocessor cyclic scheduling with precedence delays is NP-hard, since Bratley's scheduling problem  $1|r_j, \tilde{d}_j|C_{max}$  [2] can be transformed to it. The transformation is shown in Figure 6. The independent task set of Bratley's problem is represented by the nodes  $T_1, \dots, T_n$  and their release dates and deadlines are represented using precedence delays related to dummy task  $T_0$ .  $r_j$ , release date 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 (5) determines 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  $e_{i0}$  has the height  $h_{i0} = 1$  and the length  $l_{i0} = w - \tilde{d}_i + p_i$ , where  $w$  is equal to value of maximum deadline (the moment when the next iteration will potentially start). In the same way the deadline restriction, i.e.  $s_i + p_i \leq \tilde{d}_i$ , is obtained from (5) for each of these edges assuming  $s_0 = s_j = 0$ .

We remind that Bratley's problem  $1|r_j, \tilde{d}_j|C_{max}$  was proven to be NP-hard by transformation from 3-PARTITION problem [16]. Our problem of monoprocessor cyclic scheduling with precedence

delays is NP-hard, since each instance of Bratley's problem can be transformed to the instance of our scheduling problem as shown above.

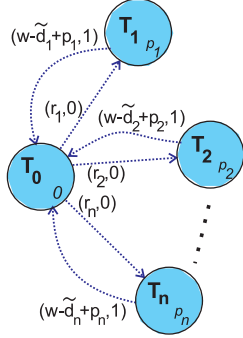


Figure 6: Transformation of Bratley's problem  $|r_j, \tilde{d}_j|C_{max}$  to monoprocessor cyclic scheduling with precedence delays.

## 5 ILP Formulation of monoprocessor cyclic scheduling with precedence delays

It is useless to look for polynomial algorithm finding optimal solution of our scheduling problem, since it is NP-hard. Therefore it is meaningful to formulate it as problem of ILP, since various ILP algorithms solve instances of reasonable size in reasonable time.

### 5.1 Precedence Constraint

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

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \quad \hat{s}_i \in \langle 0, w - 1 \rangle, \quad \hat{q}_i \geq 0 \quad (6)$$

This notation divides  $s_i$  into  $\hat{q}_i$ , index of *execution period*, and  $\hat{s}_i$ , number of clock cycles within the execution period. The schedule has to obey to two constraints. The first is *precedence constraint* restriction corresponding to inequality (5). It can be formulated using  $\hat{s}$  and  $\hat{q}$ :

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l'_{ij} - w \cdot h'_{ij} \quad (7)$$

Each edge represents one precedence constraint, hence we have  $n'_e$  inequalities ( $n'_e$  is the number of edges in reduced graph  $G'$ ).

### 5.2 Processor Constraint

The second kind of restrictions are *processor constraints*. They are related to the monoprocessor restriction, i.e. at maximum one task is executed at a given time. The execution period, which is

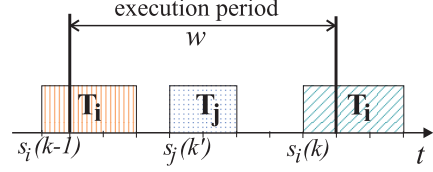


Figure 7: Processor constraint illustration example.  $T_i$  and  $T_j$  are tasks without precedence constraint ( $p_i = 2, p_j = 3$ ). Start times are  $\hat{s}_i = 7$  and  $\hat{s}_j = 3$ . Period  $w=8$ .

neither in the tail nor in the head of the schedule, contains all tasks even if they are from different iterations. Please see for example execution period 2 (time 22-33) in Figure 4. Based on this observation the processor constraints can be simply formulated using  $\hat{s}$  (please notice that processor constraints do not depend on  $\hat{q}$ ). Two disjunctive cases can occur:

In the first case we consider the task  $T_j$  to be followed by the task  $T_i$  (both are from arbitrary iterations) within execution period (see the  $k'$ -th occurrence of  $T_j$  and the  $k$ -th occurrence of  $T_i$  in Figure 7). Corresponding constraint is therefore:

$$\hat{s}_i - \hat{s}_j \geq p_j \quad (8)$$

At the same time the  $(k-1)$ -th occurrence of  $T_i$  is followed by the  $k'$ -th occurrence of  $T_j$ , therefore:

$$\hat{s}_j - (\hat{s}_i - w) \geq p_i \quad (9)$$

Both inequalities of the first case can be joined in one double-inequality:

$$p_j \leq \hat{s}_i - \hat{s}_j \leq w - p_i \quad (10)$$

In the opposite second case we consider the task  $T_i$  to be followed by the task  $T_j$ . To derive constraints for the second case it is enough to exchange index  $i$  with index  $j$  in double-inequality (10):

$$\begin{aligned} p_i &\leq \hat{s}_j - \hat{s}_i \leq w - p_j \\ p_i - w &\leq \hat{s}_j - \hat{s}_i - w \leq -p_j \\ p_j &\leq \hat{s}_i - \hat{s}_j + w \leq w - p_i \end{aligned} \quad (11)$$

Exclusive OR relation between first case and second case, i.e. either (10) holds or (11) holds, disables to formulate the problem directly as ILP program, since there is AND relation among all inequalities in ILP program. In other words the state space of  $\hat{s}_j, \hat{s}_i$  is not convex even for continuous values of  $\hat{s}_j$  and  $\hat{s}_i$  (see two polytopes in Figure 8a, upper-left one corresponding to 11 and lower-right one corresponding to (10)).

The first case, constrained by (10), differs from the opposite second case, constrained by (11), only in  $w$  in the middle of double-inequality. This term

signals whether  $T_i$  is before  $T_j$  within execution period or not. Therefore (10) and (11) can be reduced into one double-inequality while using 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  $T_i$ ):

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i \quad (12)$$

The processor constraint restrictions for two tasks  $T_i$  and  $T_j$  is illustrated in Figure 8. All feasible start times on monoprocessor are marked in Figure 8a. Other couples of start times cause overlap of tasks. Introduction of  $\hat{x}_{ij}$  realizing exclusive OR between (10) and (11) is demonstrated graphically by cuts of the polytop in Figure 8b in the planes  $\hat{x}_{ij} = 1$  and  $\hat{x}_{ij} = 0$ . Therefore we are able to formulate our problem using ILP program (AND relation among inequalities and integer restriction on variables).

To derive feasible monoprocessor schedule double-inequality (12) 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 reduced graph  $G'$ ), i.e. there are  $(n'^2 - n')$  inequalities specifying the processor constraints.

### 5.3 Objective function

Using ILP formulation we are able to test the schedule feasibility for given  $w$ . In addition we can minimize the iteration overlap by formulation the objective function as  $\min \sum_{i=1}^n \hat{q}_i$ .

The summarized ILP program, using variables  $\hat{s}_i, \hat{q}_i, \hat{x}_{ij}$ , is shown in Figure 9. It contains  $(2n' + (n'^2 - n')/2)$  variables and  $(n'_e + n'^2 - n')$  constraints.

If needed, this problem can be reformulated to minimize  $C_{max}$  by adding one variable  $c_{max}$  and  $n'$  constraints:

$$\hat{s}_j + \hat{q}_j \leq c_{max}, \quad T_j \in \mathcal{T} \quad (13)$$

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

### 5.4 Period minimizations

Nevertheless the aim of the cyclic scheduling is to find a feasible schedule with minimal period  $w$ . Therefore  $w$  is not constant but due to the periodicity of the schedule it is positive integer value. Lower bound of period  $w$  is given by equation (3) related to critical circuit of  $G'$  (identical with circuit of  $G$ ).

The schedule found by BCS of  $G'$  (assuming unlimited number of processors) enables two tasks

of  $G'$  to be processed at the same time, which results in the conflict on monoprocessor. But BCS schedule can be used to derive upper bound of period  $w$ , by serializing conflicting tasks. Such schedule, with serialized conflicting tasks, does not need to be optimal but it is feasible, therefore it gives upper bound on  $w$ .

Period  $w^*$ , the shortest period resulting in feasible schedule, can be found iteratively by formulating one ILP program for each iteration. These ILP iterations need not to be performed for all  $w$  between the lower and upper bound, 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 feasible solution). Therefore there are at maximum  $\log_2(\text{upperbound} - \text{lowerbound})$  iterative calls of ILP.

The above mentioned method gives the feasible schedule of  $G'$  on monoprocessor. Corresponding schedule of  $G$  is also feasible, since the tasks executed on unlimited number of processors obey only to the precedence relation constraints, that are already included in precedence delays of  $G'$ . Figure 10 shows the schedule of the example depicted in Figure 2 where the dedicated processor is shown on the bottom line.

## 6 Results

Presented scheduling technique was implemented in Matlab language using ILP solver tool LP-SOLVE [14]. Specific inner loop of RLS filter described in section 2 is shown in Figure 11a where the corresponding task label is above of each arithmetic operation. Figure 11b shows corresponding  $G'$ , the graph after reduction. The schedule presented in Figure 12 was found by the first call of ILP program for  $w^* = 26$  (the same period as lower bound of  $w$  given by the critical circuit). ILP program from Figure 9 for this instance was calculated in 2.09s on Intel Pentium 4 running at 2.4GHz.

Real-time demo application implemented in Celoxica rc200e development board (Chip xc2v1000-4, design clock 50MHz, audio input and output sampling at 44100Hz) using 19-bit logarithmic number system arithmetic HSLA reached order of filter  $N = 129$  on sampling frequency 44100Hz (i.e. 129 iterations of inner loop executed each 1/44100 s).

Figure 13 shows results of filtering consisting of four diagrams. The horizontal axis of each diagram represents the running time (corresponding to a time interval of 5s), the vertical axis represents the signal frequencies (up to 22kHz) and the color represents the signal amplitude. The lower-right diagram presents the original sound, the upper-left diagram presents the noise, the upper-right diagram presents corrupted sound assuming sinu-

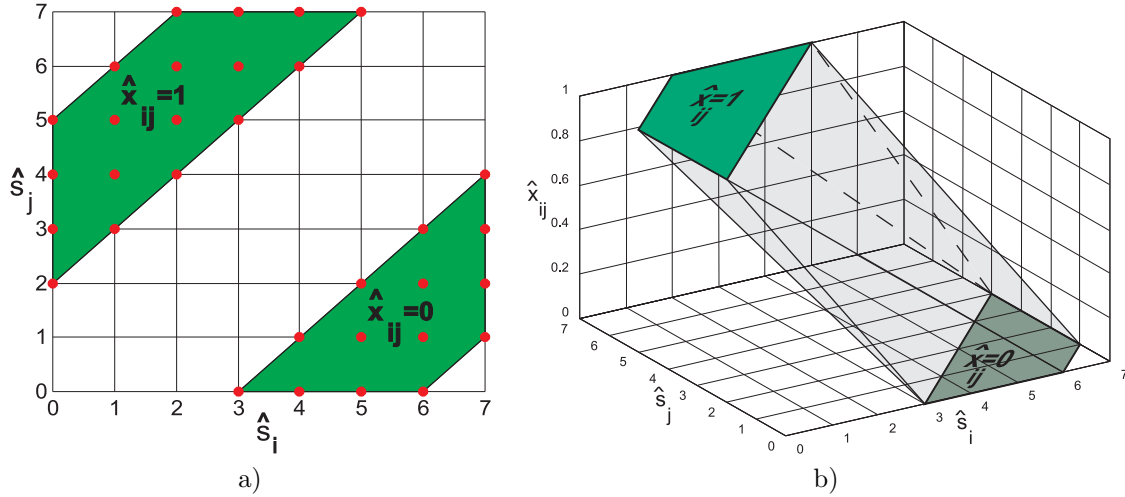


Figure 8: Statespace of feasible schedules given by double-inequality 12 of the example from Figure 7. a) Statespace of feasible start times  $\hat{s}_i, \hat{s}_j$ . b) Equivalent convex continuous statespace.

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

Subject to:

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

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i, \quad \forall i \neq j \text{ and } T_i, T_j \in \mathcal{T}$$

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

Figure 9: ILP programme.

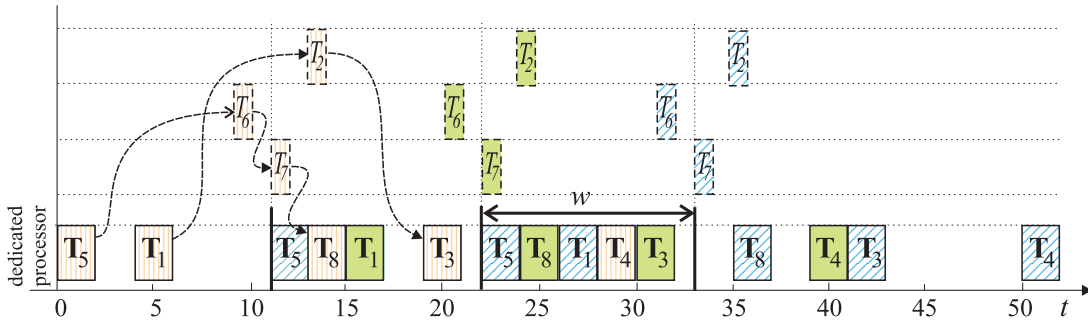


Figure 10: The schedule ( $w^* = w(G) = 11$ ) including inverse reduction to multiprocessor.

soidal changes of the estimated channel parameters, and finally the lower-left diagram presents reconstructed sound. Real-time demonstration is ready for presentation at the conference.

## 7 Conclusion

This paper presents ILP based cyclic scheduling method used to optimize computation speed iterative algorithms running on HSLA [13]. The approach is based on transformation to monoproc-

essor cyclic scheduling with precedence delays. Then optimal periodic solution is searched iteratively using interval bisection.

The advantage of ILP program presented in Figure 9 in comparison with common ILP programs used for similar problems is that the number of variables is independent of period length. Another property of the formulation is arbitrary processing time and precedence delay of each task. Moreover ILP approach enables to incorporate additional

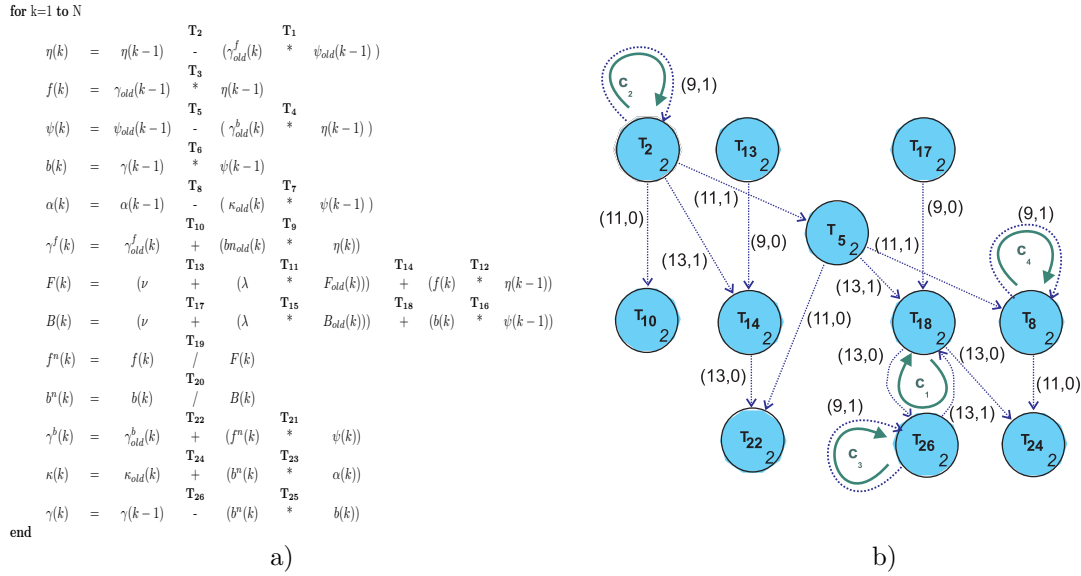


Figure 11: a) The inner loop of RLS filter. Constant  $N$  determines the filter order. b) Corresponding reduced graph  $G'$  contains four circuits  $c_1$ ,  $c_2$ ,  $c_3$  and  $c_4$  with critical circuit  $c_1$  since  $w = \max\{26/1, 9/1, 9/1, 9/1\} = 26$ .

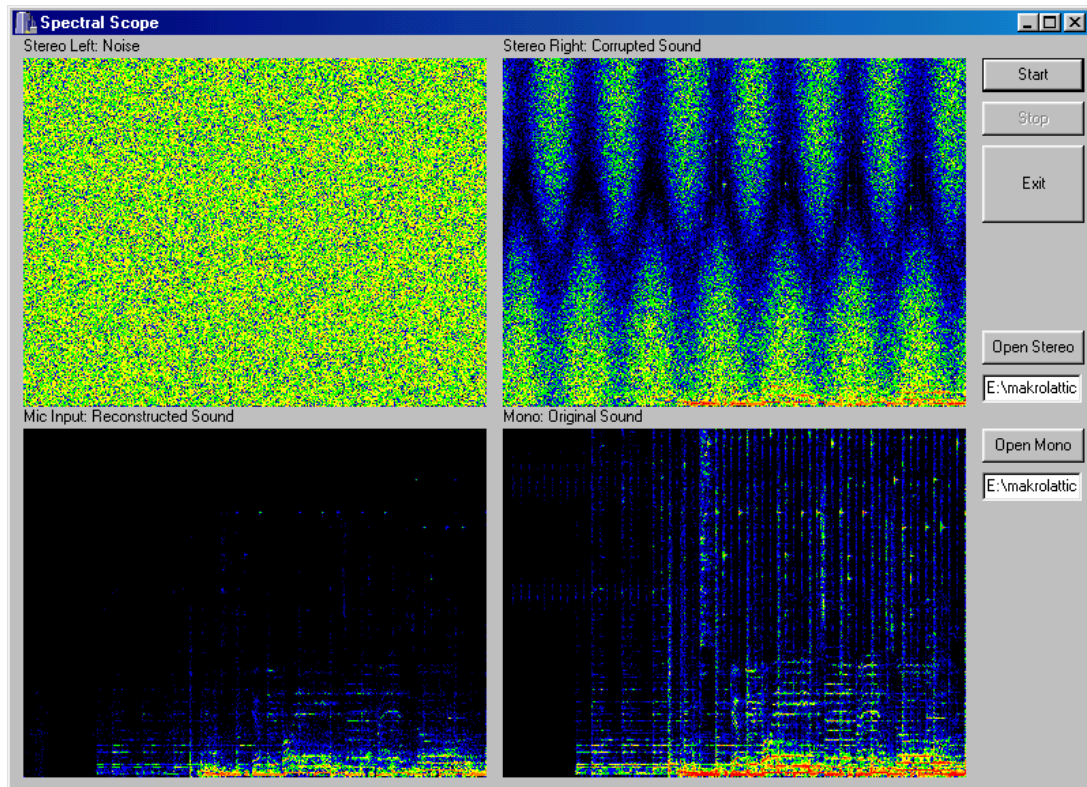


Figure 13: Active noise cancellation

constraints. Therefore the tasks can be also constrained by release dates and deadlines related to beginning of the period (this features were not explained in this paper since they are not exploited in RLS filter application).

Results of the scheduling applied on RLS filter automated design are better than the ones achieved by experienced FPGA programmer. For given sampling period the filter order achieved by our method is 129, in contrast to manual design achieving order 75. This acceleration by 70% is



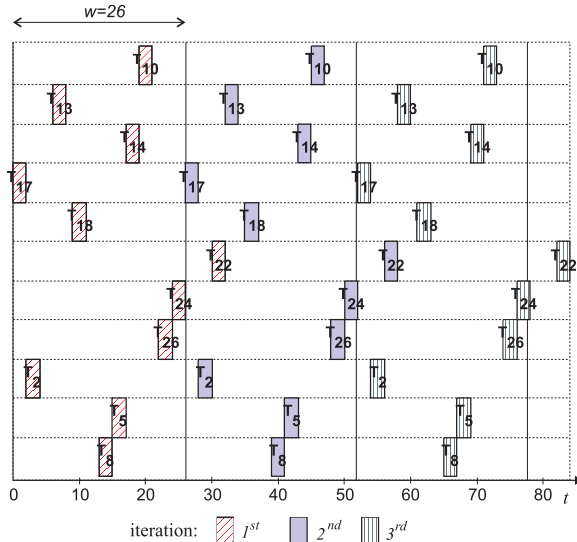


Figure 12: The schedule ( $w = 26$ ) of RLS filter inner loop.

due to the schedule overlap (operations belonging to different iteration are executed simultaneously), which is rather difficult for manual design but it is straight for cyclic scheduling.

Our method of algorithm modeling, transformation, scheduling is fully automated therefore it can be easily incorporated in design tools while processing considerable simplification for rapid prototyping.

In future work we would like to develop more general technique for another FPGA libraries differing from HSLA in the number of dedicated processors. For the ILP acceleration it is needed to study discrete convex optimization in order to be able to remove integer constraints on  $\hat{s}_i$ , which seems to be feasible.

## References

- [1] J. Błazewicz, K. Ecker, G. Schmidt and J. Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer, second edition, 2001.
- [2] P. Bratley, M. Florian and P. Robillard. Scheduling with earliest start and due date constraints. *Naval Res. Logist. Quart.* 18, 1971.
- [3] N. Chabini and Y. Savaria. Methods for optimizing register placement in synchronous circuits derived using software pipelining techniques. In *ISSS*, pages 209–214, 2001.
- [4] C. M. Chang, C. M. Chen and C. T. King. Using integer linear programming for instruction scheduling and register allocation in multiissue processors. *Computers and Mathematics with Applications*, 1997.
- [5] P. Chrétienne. List schedules for cyclic scheduling. In *Proceedings of the third international conference on Graphs and optimization*, pages 141–159. Elsevier Science Publishers B. V., 1999.
- [6] P. Chrétienne. On graham’s bound for cyclic scheduling. *Parallel Comput.*, Volume 26, Number 9, pages 1163–1174, 2000.
- [7] P. Chrétienne, E. G. Coffman, J. K. Lenstra and Zhen Liu. *Scheduling Theory and its Application*. John Wiley and Sons, 1995.
- [8] B. D. Dinechin. Simplex scheduling: More than lifetime-sensitive instruction scheduling. *Proceedings of the International Conference on Parallel Architecture and Compiler Techniques*, 1994.
- [9] D. Fimmel and J. Müller. Optimal software pipelining under resource constraints. *Journal of Foundations of Computer Science*, 2001.
- [10] M. Gondrant and M. Minoux. *Graphes et algorithmes*. Eyrolles, 1985.
- [11] R. Govindarajan, E. R. Altman and G. R. Gao. A framework for resource-constrained rate-optimal software pipelining. *IEEE Transactions on Parallel and distributed systems*, vol. 7, no. 11, 1996.
- [12] C. Hanen and A. Munier. A study of the cyclic scheduling problem on parallel processors. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, Volume 57, 1995.
- [13] A. Heřmánek, Z. Pohl and J. Kadlec. Fpga implementation of the adaptive lattice filter. In *Proc. FPL2003*, Springer, Berlin, 2003.
- [14] J. C. Kantor. *LP-SOLVE 2.3*. [ftp://ftp.es.ele.tue.nl/pub/lp\\_solve/](ftp://ftp.es.ele.tue.nl/pub/lp_solve/), 1995.
- [15] I. Kazuhito, E. Lucke and K. Parhi. Ilp based cost-optimal dsp synthesis with module selection and data format conversion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1999.
- [16] J. K. Lenstra, A. R. Kan and P. Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.* 1, 1977.
- [17] A. Leung, K. V. Palem and A. Pnueli. A fast algorithm for scheduling time-constrained instructions on processors with ilp. *IEEE PACT*, 1998.

- [18] R. Matoušek, M. Tichý, A. Z. Pohl, J. Kadlec and C. Softley. Logarithmic number system and floating-point arithmetics on fpga. Field-Programmable Logic and Applications: Reconfigurable computing Is Going Mainstream. Lecture notes in Computer Science A 2438, Springer, Berlin, 2002.
- [19] S. Megerian, M. Drinic and M. Potkonjak. Watermarking integer linear programming solutions. Proceedings of the 39th conference on Design automation, New Orleans, Louisiana, USA, 2002.
- [20] A. Munier. The complexity of a cyclic scheduling problem with identical machines. Rapport masi, Institut Blaise Pascal, 1990.
- [21] G.-J. Olsder, K. Roos and R.-J. Egmond. An efficient algorithm for critical circuits and finite eigenvectors in the max-plus algebra. *Linear Algebra and its Applications*, Volume 295, Number 1–3, pages 231–240, 1999.
- [22] H.-J. Park and B. K. Kim. An efficient optimal task allocation and scheduling algorithm for cyclic synchronous applications. Proceedings 6th International Conference on Real-Time Computing Systems and Applications (RTCSA '99), 1999.
- [23] S. L. Sindorf and S. H. Gerez. An integer linear programming approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays. PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands, 2000.