Scheduling of Iterative Algorithms with Matrix Operations for Efficient FPGA Design - Implementation of Finite Interval Constant Modulus Algorithm

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

Antonín Heřmánek, Jan Schier
Institute of Information Theory and Automation,
Academy of Sciences of the Czech Republic
{hermanek,schier}@utia.cas.cz

July 18, 2006

Abstract

This paper deals with the optimization of iterative algorithms with matrix operations or nested loops for hardware implementation in Field Programmable Gate Arrays (FPGA), using Integer Linear Programming (ILP). The method is demonstrated on an implementation of the Finite Interval Constant Modulus Algorithm. It is an equalization algorithm, suitable for modern communication systems (4G and behind). For the floating-point calculations required in the algorithm, two arithmetic libraries were used in the FPGA implementation: one based on the logarithmic number system, the other using floating-point number system in the standard IEEE format. Both libraries use pipelined modules. Traditional approaches to the scheduling of nested loops lead to a relatively large code, which is unsuitable for FPGA implementation. This paper presents a new high-level synthesis methodology, which models both, iterative loops and imperfectly nested loops, by means of the system of linear inequalities. Moreover, memory access is considered as an additional resource constraint. Since the solutions of ILP formulated problems are known to be computationally intensive, an important part of the article is devoted to the reduction of the problem size.

Keywords: High-level synthesis, cyclic scheduling, iterative algorithms, imperfectly nested loops, integer linear programming, FPGA, VLSI design, blind equalization, implementation.

1 Introduction

In modern digital communication systems an estimator of transmitted symbols represents one of the critical parts of the receiver. The estimator consists typically of an equalizer and of a decision device. Recent systems (such as the GSM system) use methods based on training sequences, where part of signal is known and repeated. The equalizer is based on matching its output to the reference signal, by adapting its parameters to minimize some criterion. Unfortunately the training sequence consumes a considerable part of the overall message (approx. 25% in GSM). For this reason, much research effort has been devoted to blind deconvolution algorithms, i.e., algorithms with no training sequence. Perhaps the most popular blind algorithm (performing estimation without any training sequence) is the Constant Modulus Algorithm (CMA), originally proposed by Godard [1]. The Finite Interval Constant Modulus Algorithm (FI-CMA) [2] is a windowed version of CMA where a time-window operator is applied to the received data. Compared to other algorithms in the class of blind deconvolution algorithms , it is characterized by fast convergence (see Section 2, Figure 1(b)) at acceptable computational complexity. Nevertheless FI-CMA requires up to 8 iterations operating on matrices of sampled data to be performed each $3.7\mu s$.

The FPGA technology supports massive fine tuned parallelism and high data throughput. On the other hand, the FPGA design gets complicated, due to the representation of real numbers. One solution is to use an arithmetic library implementing a 32-bit floating point number system, compliant with IEEE standards [3]. An alternative solution is to use the logarithmic number system arithmetic, where a real number is represented as the fixed-point value of base two logarithm of its absolute value [4]. In each case, rather complex arithmetic is 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.

This paper deals with the scheduling of digital signal processing algorithms, where at least K iterations have to be executed each sampling period in order to achieve the desired algorithm convergence. Moreover, the iteration may contain rather complex data computations, usually expressed in the form of matrix operations and implemented as nested loops. The scheduling method shown in this paper models both, iterative loops and imperfectly nested loops, by means of the system of linear inequalities. The target hardware based on FPGA, a set of pipelined dedicated processors, is formalized as a system of linear inequalities with integer variables. The scheduling problem is then solved using the Integer Linear Programming (ILP), resulting in the optimal schedule. Thanks to an efficient representation of imperfectly nested loops and to a polynomial reduction of decision variables we are able to handle scheduling problems of significant size as demonstrated on applications (hardware implementation of FI-CMA shown in this article; FI-CMA on different hardware architecture shown in internal report [5]). We show how, given the hardware resources, the mathematical programming can be used to optimize the execution time of the implemented algorithm.

1.1 Related work in Parallel Implementation of Signal Processing Algorithms

The signal processing algorithms typically rely on loops, either performing vectormatrix computations, or performing iterative computations. In the signal processing field, the concept of loop unrolling closely relates to the so-called systolic array – regular mesh of synchronous "processing elements", which however does not account for the time optimization issues. Between 1980s and mid 1990s, there were indeed many works in this field treating transformation of algorithm to the systolic arrays. Examples range from simple systolic arrays for the FIR filters [6] to the recursive QR algorithms [7, 8] and to the QSVD update [9]. They become again topical with the recent advances of the FPGA technology, with embedded multipliers and DSP units.

Practical FPGA realizations first involved simple algorithms (FIR and LMS filters), implemented using fix-point arithmetic. Later, from the end of 90', implementations of the recursive QR algorithm were published (see [10]). It should also be noted, though, that probably all published designs treated the problem of the *recursive* algorithm, where rotation matrix Q is computed only implicitly, while in our paper, block version of the algorithm is used and it is this matrix that is of interest. An interesting design has been published in [11], where floating point cores are used to implement a high-speed QR processor in the Virtex-E device. We are not aware of any FPGA designs that would implement complete blind equalization algorithm. Moreover, major development in the floating point libraries has happened only in the recent years and they still have to be considered with hardware demands in mind.

1.2 Related Work on Scheduling of Iterative Algorithms

An iterative algorithm can be seen as an *iterative loop* performing an identical set of operations repeatedly (one repetition of the loop is called an *iteration*). When the number of iterations is large enough, the optimization can be performed by *cyclic scheduling* while minimizing the completion time of the set of K iterations. If operations belonging to different iterations interleave, the schedule is called *overlapped* [12]. Efficient exploitation of the schedule overlap and pipelining is rather difficult to achieve in manual design.

The *periodic schedule* is given by a schedule of one iteration that is repeated with a fixed time interval called a *period* (also called *initiation interval*). The aim is to find a periodic schedule with a minimum period. If the number of processors is not limited, a periodic schedule can be built in polynomial time [13]. For a fixed number of processors the problem becomes NP–complete [14]. The general solution to this problem is shown e.g. in [15].

Cyclic scheduling presented in [16] is not dependent on the period length and with respect to [15] it leads to simpler problem formulation with less integer variables. Moreover, this model allows to reduce number of interconnections by minimization of the data transfers as shown in [17]. *Modulo scheduling* and *software pipelining* [18, 19], usually used in the compiler community, are alternative terms to *cyclic scheduling*, used in the scheduling community.

1.3 Related Work on Scheduling of Nested Loops

For practical reasons, we usually do not want to expand matrix operations into scalar operations, since we want to achieve regularity of the schedule (efficiently implemented in the form of the nested loops) and we want to prevent enormous growth in the number of the scheduled tasks (i.e. to prevent the growth of computation time required by the scheduling algorithm). Therefore, we intend to schedule matrix operations in the form of nested loops.

A great deal of work in this field has been focused on *perfectly-nested* loops (i.e. all elementary operations are contained in the innermost loop). For example, one of the often used optimization approaches called loop shifting (operations from one iteration of the loop body are moved to its previous iteration) was recently extended in [20, 21]. Another approach is the unroll-and-jam (also called unfolding or unwinding) [22], which partially unrolls one or more loops higher in the nest than the innermost loop, and fuses the resulting loops back together. Improvement by unroll-and-squash technique has been shown in [23].

In our case the loops are *imperfectly-nested* (i.e. some elementary operations are not contained in the innermost loop). Existing compilers usually use heuristics transforming them into perfectly-nested loops [24]. The tiling method, extended for imperfectly-nested loops, is discussed in [25]. Loop tiling is a transformation technique, which divides the iteration space of loop computations into tiles (or blocks) of some size and shape, so that traversing the tiles results in covering the whole iteration space [24]. The tiling method is also used in PICO-NPA [26], a tool for high-level synthesis of nonprogrammable hardware accelerators. Nevertheless, the tool is limited to perfectly-nested loops. However, such approaches can greatly expand the code size, which is unacceptable with respect to the limited size of FPGAs as well as to the number of interconnections in the design.

1.4 Outline and Contribution

In this paper, we propose a new method for scheduling of iterative algorithms with imperfectly-nested loops on the set of pipelined dedicated processors. The method is based on cyclic scheduling of iterative algorithms where matrix operations are modeled by “united edges” and “approximated expansion”. The method is based on the construction of model, which models the nested loops, and which is optimally scheduled using integer linear programming (ILP). Moreover, access into the memory is considered as an additional resource constraint.

A lot of research has been done in scheduling of nested loops as mentioned above. Our method differs in mathematic formulation of the scheduling problem which leads to simpler code and therefore more efficient FPGA implementation. Applications requiring matrix operations usually lead to schedules with long period. Therefore classical ILP formulations of cyclic scheduling (e.g. [12]), where the number of integer/binary variables (and also time complexity) depends on the period length, are inconvenient in this framework. Number of integer/binary variables of our ILP model is independent of period length. Further to improve time performance of the scheduling algorithm we show a polynomial method reducing the size of the ILP model. Primarily, we look for the shortest feasible period. For such period we find a schedule minimizing the number of interconnections (related to the data transfers).

The proposed scheduling method is demonstrated on the Finite Interval Constant Modulus Algorithm (FI-CMA). To achieve an acceptable computational performance and precision, the iterative algorithm has to be implemented using floating-point arithmetics. We consider two libraries of arithmetic units (Celoxica pipelined floating-point library (FP32) [3] and High-Speed Logarithmic Arithmetics (HSLA) [4]) and we show that by using the presented method, the optimal architecture can be chosen for a given algorithm prior to its implementation.

Starting from the algorithm representation and hardware parameters, the presented method shows, how to represent the problem by a graph and how to find an optimal schedule. The schedule, containing start times of operations, is further used to automatically generate a state machine in Handel-C or VHDL, controlling a parallel execution of the algorithm on FPGA.

The main contributions of this paper are: (a) a novel scheduling method for iterative algorithms with imperfectly-nested loops on the set of dedicated processors leading to efficient design on FPGA (see Section 4). (b) a representation of imperfectly-nested loops in the form of linear inequalities (see Subsection 4.1.1). (c) a polynomial algorithm for elimination of redundant decision variables which considerably accelerates the scheduling algorithm (see Subsection 3.4). (d) an implementation of the Finite Interval Constant Modulus Algorithm (see Section 5) considered for 4G communication systems. While using our scheduling method, we reached a considerable speedup in comparison with [27], the first implementation of FI-CMA in FPGA.

The paper is organized as follows: in the next section the FI-CMA algorithm is briefly outlined. Section 3 surveys the scheduling of iterative algorithms (considering only scalar variables) on the set of dedicated processors by Integer Linear Programming (multiprocessor extension of [16]). Section 4 presents the main contribution of the paper, scheduling of iterative algorithms with imperfectly-nested loops, illustrated by the parallelization of the FI-CMA algorithm. The construction of the model, respecting the regularity of algorithm data dependencies, is described and the resulting schedule is shown. Parallelization of the QR decomposition, used in FI-CMA algorithm, is described in Section 4.2. Section 5 presents the HW implementation of FI-CMA algorithm on FPGA using two different libraries of arithmetic units (HSLA or FP32). The last section concludes the work.

2 Constant Modulus Algorithm

In this section, the system model and constant modulus (CM) criterion will be briefly reviewed, followed by derivation of the FI-CMA algorithm.

Let $\{s_n\}$ be an IID symbol sequence to be transmitted, adhering to a constant modulus (CM) constellation, such as PSK. The data symbols are transmitted over a Single-Input Multiple-Output (SIMO) discrete channel with a finite length impulse response matrix $\mathbf{H}$. The received signal has the form

$$\mathbf{u}_n = \mathbf{H}\mathbf{s}_n + \mathbf{b}_n. \qquad (1)$$

where $\mathbf{s}_n = [s_n\ s_{n-1}\ \ldots\ s_{n-M_c}]^T$ collects the M most recent input symbols, $\mathbf{b}_n$ is a background noise vector, $\mathbf{H}$ is the $P \times M_c$ channel impulse response matrix, and P is the number of antennas or the oversampling factor. An equalizer can be seen as a linear combiner of order M with output y_n,

$$y_n = \sum_{k=0}^{M} \mathbf{g}_k^T \mathbf{u}_{n-k} = \mathbf{g}^T \mathbf{U}_n. \qquad (2)$$

where n is the discrete baud rate time and $\mathbf{g}$ and $\mathbf{U}$ are defined as

$$\mathbf{g} = \begin{bmatrix} \mathbf{g}_0^T & \mathbf{g}_1^T & \ldots & \mathbf{g}_M^T \end{bmatrix}^T,$$
$$\mathbf{U}_n = \begin{bmatrix} \mathbf{u}_n^T & \mathbf{u}_{n-1}^T & \ldots & \mathbf{u}_{n-M}^T \end{bmatrix}^T.$$

The CMA algorithm is based on minimization of a cost function defined by the constant modulus (CM) criterion. This criterion penalizes deviation in magnitude of the equalizer output from a fixed value. It has the following form

$$J_{CMA}(\mathbf{g}) = \frac{1}{4} E\left[(|y_n|^2 - \gamma)^2\right]. \qquad (3)$$

where $E[\cdot]$ is the expectation operator and γ is a constant chosen as a function of the source alphabet.

The main advantage of this criterion is that the resulting gradient descent algorithm is very similar to the well known Least Mean Square (LMS) algorithm, hence, it can be implemented in a very simple way. On the other hand, the relatively slow convergence of CMA ($\approx 10^4$ iterations), as well as its dependence on the initialization and on the step-size parameter, are recognized drawbacks.

2.1 Finite Interval CMA algorithm

The FI-CMA algorithm [2] is a windowed version of (3) where a time-window operator is applied to the received data. The cost function has the following form

$$\begin{aligned} J(\mathbf{g}) &= \sum_{n=1}^{N} \left(|y_n|^2 - 1\right)^2 \\ &= \sum_{n=1}^{N} \left(\left|\mathbf{g}^T \mathbf{U}_n\right|^2 - 1\right)^2. \qquad (4) \end{aligned}$$

Constant γ used in (3) is replaced by 1 without loss of generality (the value does not change the position of the local extrema points).

It follows from (2) that N successive equalizer outputs can be rewritten in matrix form as

$$\begin{aligned} \mathbf{y} &= \begin{bmatrix} y_1 & \ldots & y_N \end{bmatrix}^T \\ &= \underbrace{\begin{bmatrix} \mathbf{U}_1 & \ldots & \mathbf{U}_N \end{bmatrix}^T}_{\mathcal{U}} \mathbf{g} \\ &= \mathbf{QRg} = \mathbf{Qw}. \qquad (5) \end{aligned}$$

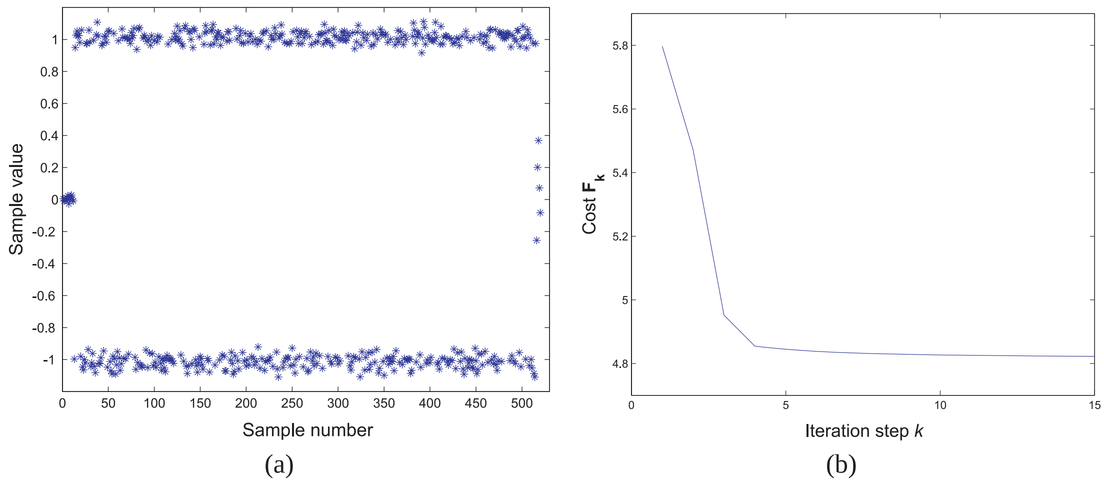


Figure 1: (a) Equalizer output after convergence (i.e. 5 iterations in that case). (b) Evolution of the cost function F_k over iteration k.

where the QR-decomposition of matrix $\mathcal{U}$ is used to obtain an orthonormal matrix $\mathbf{Q}$.

The optimal equalizer coefficients are reached by the following iterative procedure

$$\begin{aligned} \mathbf{v}_{k+1} &= \mathbf{w}_k - \mu_k \mathbf{Q}^T \mathbf{y}_k^{\odot 3}/F_k \\ \mathbf{w}_{k+1} &= \mathbf{v}_{k+1}/\|\mathbf{v}_{k+1}\| \end{aligned} \tag{6}$$

where k is the iteration counter and $(\cdot)^{\odot n}$ is an element-wise power operator. F_k is defined as

$$F_k = |\mathbf{y}_k|_4,$$

$\mathbf{v}_k$ is an auxiliary variable, $\mathbf{w}_k$ is a transformed parameter vector and $\mathbf{y}_k = \mathbf{Q}\mathbf{w}_k$ (see Equation 5). The equalizer output after convergence is shown in Figure 1(a).

The decomposition used in (5) is calculated using Givens rotations: let $\mathbf{R}_j$ be a matrix obtained by triangularization of sub-matrix $\mathcal{U}_j$ containing first j rows of matrix $\mathcal{U}$ (see Equation 5). Let $\mathbf{Q}_j$ be the corresponding orthonormal matrix. Matrices $\mathbf{Q}_{j+1}$ and $\mathbf{R}_{j+1}$, which represent decomposition of matrix $\mathbf{U}_{j+1}$, can be calculated as follows

$$\begin{bmatrix} \mathbf{R_{j+1}} \\ \mathbf{0} \end{bmatrix} = \mathbf{G}_M \mathbf{G}_{M-1} \dots \mathbf{G}_1 \begin{bmatrix} \mathbf{R}_j \\ \mathbf{u}_{j+1} \end{bmatrix}, \tag{7}$$

$$\begin{bmatrix} \mathbf{Q}_{j+1} \mid \mathbf{q} \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{Q}_j & 0 \\ \hline 0 & 1 \end{array}\right] \mathbf{G}_1^T \dots \mathbf{G}_{M-1}^T \mathbf{G}_M^T. \tag{8}$$

where Givens-rotation matrix $\mathbf{G}_i$ eliminates the j-th element of the data vector $\mathbf{u}_{j+1}$. As it is well known [28], the matrix is composed of sine and cosine values (c_j,s_j)

$$G_i = \begin{bmatrix} I & & & \\ & \cos\theta_i & & \sin\theta_i \\ & & I & \\ & -\sin\theta_i & & \cos\theta_i \end{bmatrix}.$$

where I is identity matrix of appropriate size. The sine and cosine parameters are computed as follows:

$$\cos\theta_i = \frac{r_{i,i}}{\sqrt{r_{i,i}^2 + u_{i,j+1}^2}} \tag{9}$$

$$\sin\theta_i = \frac{u_{i,j+1}}{\sqrt{r_{i,i}^2 + u_{i,j+1}^2}} \tag{10}$$

Note: The initial values of matrices $\mathbf{R}$ and $\mathbf{Q}$ are $\mathbf{R}_0 = \begin{bmatrix} \mathbf{u}_1 & \mathbf{0} \end{bmatrix}^T$ and $\mathbf{Q}_0 = I_{2\times 2}$.

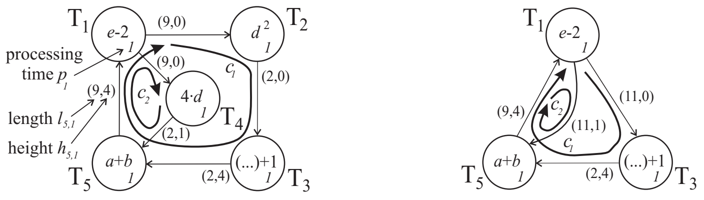
To summarize, the FI-CMA algorithms consist of two successive parts: the QR decomposition of the input data matrix (equations (7) and (8)) and the iterative procedure (6). Compared to the recursive gradient (LMS type) algorithm mentioned earlier, this algorithm converges after few iterations (up to 10) and the optimal step size is analytically known and data dependent only (see [2]).

3 Formulation and Solution of the Scheduling Problem

The iterative procedure, as the one mentioned in the previous section, can be implemented as a com-

```
for k=1 to K do    //iterative loop
    $d(k) = e(k-4) - 2$
    $a(k) = d(k)^2 + 1$
    $b(k) = 4 \cdot d(k)$
    $e(k) = a(k-4) + b(k-1)$
end
```

(a)



(b) (c)

Figure 2: (a) Illustrative example of the iterative algorithm. (b) The corresponding data dependency graph G. G contains two cycles c_1 and c_2 with average cycle times $\{22/8, 20/5\}$. The critical circuit is c_2. (c) The corresponding reduced graph, which is explained in Section 3.2.

putation loop performing an identical set of operations repeatedly. Therefore our work, dealing with an optimized implementation of such procedures, is based on cyclic scheduling (also referenced as *modulo scheduling* or *software pipelining* [18, 19]).

3.1 Cyclic Scheduling Problem

Operations in a *iterative loop* can be considered as a set of n *generic tasks* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed K times. One execution of $\mathcal{T}$ labeled with integer index $k \geq 1$ is called an *iteration*. The scheduling problem is to find a start time $s_i(k)$ of every occurrence of T_i [13]. Figure 2(a) shows an illustrative example of a simple computation loop.

Data dependencies of this problem can be modeled by a directed graph G. Each task (node in G) is characterized by the processing time p_i. Edge e_{ij} from the node i to j is weighted by a couple of integer constants l_{ij} and h_{ij}. Length 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 is always greater than zero.

Considering pipelined processors, 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). Parameter height h_{ij} specifies a shift of the iteration index (dependence distance) related to the data produced by T_i and consumed by T_j.

Figure 2(b) shows the data dependence graph of the iterative loop shown in Figure 2(a). Processing time p_i is equal to 1 for all units of HSLA and length l_{ij} correspond to input–output latency of given unit of HSLA (see Table 1).

Assuming a *periodic schedule* with the *period* τ (i.e. the constant repetition time of each task), each edge e_{ij} in graph G represents one precedence relation constraint

$$s_j - s_i \geq l_{ij} - \tau \cdot h_{ij}, \qquad (11)$$

where s_i denotes the start time of task T_i in the first iteration.

The aim of cyclic scheduling [13] is to find a periodic schedule while minimizing the period τ. The scheduling problem is simply solved when the number of processors is not limited, i.e. is sufficiently large. Given by the *critical circuit* c in graph G, the minimal feasible period is equal to $\max_{c \in C(G)} \left(\sum_{e_{ij} \in c} l_{ij}\right) / \left(\sum_{e_{ij} \in c} h_{ij}\right)$, where $C(G)$ denotes a set of cycles in G. Critical circuit can be found in polynomial time [13], and we use this value to determine lower bound of the period in our scheduling problem.

When the number of processors m is restricted, the cyclic scheduling problem becomes NP–complete [13]. Unfortunately, in our case the number of processors is restricted and the processors are dedicated to execute specific operations (see Table 1).

3.2 Problem with Dedicated Processors

In FPGA, some arithmetic units can be implemented quite cheaply in terms of SLICEs and BRAMs (see MUL, DIV and SQRT units in Table 1 for HSLA parameters). Therefore a designer can admit a high number of these units. Moreover, the algorithm structure (represented by the graph G) permits only limited number of corresponding tasks to be executed concurrently. Consequently these arithmetic units are not critical resources, they will be further called *reduced processors*, and $\mathcal{T}^r$ will be the set of *reduced tasks*, i.e. the ones executed on reduced processors. For example, T_2 and T_4 in Figure 2(b) executed on HSLA, need not be restricted by resource

Table 1: Parameters of the 19-bit (32-bit) HSLA units for XCV2000E-6 FPGA device. Maximal clock frequency is 50 MHz.

Unit	**Input-output Latency [clk]**	**SLICEs [%]**	**BRAMs [%]**
ADD	9	8(13)	3(70)
MUL	2	1	0
DIV	2	1	0
SQRT	2	1	0

constraints, since MUL operation is cheaply implemented in logarithmic arithmetics.

Remaining tasks, denoted by the set $\mathcal{T}' = \mathcal{T} \setminus \mathcal{T}^r$, must be considered in the scheduling algorithm, since they are executed on m' dedicated processors (the processors that can not be reduced). The input-output latency of the reduced tasks is reflected in the lengths of arcs in the reduced graph G'. When the tasks on dedicated processors are scheduled, the reduced tasks are simply executed as soon as possible, while satisfying precedence relations (given by the original graph G).

Therefore, we perform the following reduction of graph G to the *reduced graph* G' while using the calculation of the longest paths (solved e.g., by the Floyd's algorithm). All nodes (tasks), except the ones running on the dedicated processors, are eliminated (see Figure 2(c)). Therefore, tasks in $\mathcal{T}'$ running on the dedicated processors 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 a path from T_i to T_j in G of height h'_{ij} such that this path goes only through the reduced tasks. The value of length l'_{ij} is the longest path from T_i to T_j in G of height h'_{ij}. Therefore, the input-output latency of the reduced tasks is merged into l'_{ij} and their iteration shift into h'_{ij}.

Such reduction allows to formulate our scheduling problem as a problem with set of m' dedicated processors, where 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 reduction, considering one dedicated processor (ADD unit) performed on graph G, in Figure 2(b), is shown in Figure 2(c).

3.3 Solution of Cyclic Scheduling on Dedicated Processors by ILP

The approach presented in this subsection is a multiprocessor extension of work shown in [16]. The size of our ILP model (and also time complexity) is independent of the period τ, which is particularly useful in our application since matrix operations and nested loops lead to schedules with a long period.

Let $\hat{s}_i$ be the remainder after division of s_i by τ and let $\hat{q}_i$ be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot \tau, \;\; \hat{s}_i \in \langle 0, \tau - 1 \rangle \,, \;\; \hat{q}_i \geq 0. \quad (12)$$

This notation divides s_i into $\hat{q}_i$, the index of the *execution period*, and $\hat{s}_i$, the index within the execution period. The schedule has to obey two kinds of restrictions. The first kind of restriction is given by the *precedence constraint* corresponding to Inequality (11). It can be formulated using $\hat{s}$ and $\hat{q}$ as

$$(\hat{s}_j + \hat{q}_j \cdot \tau) - (\hat{s}_i + \hat{q}_i \cdot \tau) \geq l'_{ij} - \tau \cdot h'_{ij}. \quad (13)$$

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

The second kind of restriction are *processor constraints*. They are related to the processor restriction, i.e. at maximum one task is executed at a given time on the dedicated processor P_d, therefore

$$p_j \leq \hat{s}_i - \hat{s}_j + \tau \cdot \hat{x}_{ij} \leq \tau - p_i, \quad (14)$$

where $\hat{x}_{ij}$ is a binary decision variable ($\hat{x}_{ij} = 1$ when T_i is followed by T_j and $\hat{x}_{ij} = 0$ when T_j is followed by T_i).

To derive a feasible schedule, Double–Inequality (14) must hold for each unordered couple of two distinct tasks $T_i, T_j \in \mathcal{T}'_d$. Therefore, there are $(n'^2_d - n'_d)/2$ Double–Inequalities related to processor P_d (where n'_d is the number of tasks on one dedicated processor P_d in the reduced graph G'), i.e. there are ${n'_d}^2 - n'_d$ inequalities specifying the processor constraints.

Using ILP formulation we are able to test the schedule feasibility for a given τ. Please notice that τ is assumed to be a constant in the ILP model (in order to avoid multiplication of two variables). The optimal period τ^*, the shortest period resulting in a feasible schedule, can be found iteratively by formulating one ILP model for each τ between the lower and upper bound (explained in Subsection 5.2). Consequently, for a given τ^* we can minimize the data transfers among the tasks (i.e. the objective function of the ILP model minimizes the number of registers used to store intermediate results) as a secondary objective.

Minimization of data transfers considerably simplifies interconnections on FPGA. To achieve this we add one slack variable Δ_{ij} to each precedence constraint (13) resulting at

$$(\hat{s}_j + \hat{q}_j \cdot \tau) - (\hat{s}_i + \hat{q}_i \cdot \tau) + \Delta_{ij} = l_{ij} - \tau \cdot h_{ij}. \quad (15)$$

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^b_{ij} is equal to 0 otherwise. This relation is formulated as

$$(\tau \cdot (\hat{q}_{max} + 1)) \cdot \Delta^b_{ij} \geq \Delta_{ij}, \quad \forall e_{ij} \in G, \quad (16)$$

where $(\tau \cdot (\hat{q}_{max} + 1))$ represents an upper bound on Δ_{ij} and the objective is to minimize $\sum \Delta^b_{ij}$. Such a reformulated problem not only decides the feasibility of the schedule for the given period τ as a primary objective, but if such a schedule exists, it also finds the one with minimal data transfers among the tasks as a secondary objective.

The summarized ILP model, using variables $\hat{s}_i, \hat{q}_i, \hat{x}_{ij}, \Delta^b_{ij}, \Delta_{ij}$, is shown in Figure 3. It contains $2n' + \sum_{d=1}^{m'} (n'^2_d - n'_d)/2 + n'_e$ integer variables and $2n'_e + \sum_{d=1}^{m'} (n'^2_d - n'_d)$ constraints.

$\min \sum \Delta^b_{ij}$

subject to

$\hat{s}_j + \hat{q}_j \cdot \tau - \hat{s}_i - \hat{q}_i \cdot \tau + \Delta_{ij} = l'_{ij} - \tau \cdot h'_{ij},$
$\forall e'_{ij} \in G'$

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

$(\tau \cdot (\hat{q}_{max} + 1)) \cdot \Delta^b_{ij} \geq \Delta_{ij},$
$\forall e'_{ij} \in G'$

where

$\hat{s}_i \in \langle 0, \tau - 1 \rangle;\; \hat{q}_i, \Delta_{ij} \geq 0;\; \hat{x}_i, \Delta^b_{ij} \in \langle 0, 1 \rangle,$
$\hat{s}_i, \hat{q}_i, \hat{x}_{ij}, \Delta^b_{ij}$ are integers.

Figure 3: ILP model.

3.4 Elimination of Redundant Processor Constraints

The time requirements to solve the generated ILP model roughly corresponds to the number of integer variables, therefore, it is meaningful to reduce this number as much as possible. Not all processor constraints (14) are necessary when considering precedence constraints (13). Keeping in mind that the tasks from distinct iterations can be executed in one period, we set up $\hat{q}_{max}$. This value, calculated *a priori* in polynomial time using evaluation of the longest paths, is an upper bound on difference of two tasks iteration indeces executed in the same period. The necessity of Double–Inequality (14) for the couple of tasks T_i and T_j dedicated to the same processor could be decided e.g. while using polynomial linear programming (LP) composed of constraints (19), (17) and (18). If any feasible solution of this LP problem exists, then T_i and T_j can be in conflict and x_{ij} has to remain in the ILP model.

Tasks T_i and T_j are in conflict, if and only if, $s_j + p_j > s_i$ and $s_i + p_i > s_j$. Moreover, in this cyclic scheduling case, we have to investigate a possible conflict of tasks from different iterations within one period. That is why we investigate a separate LP problem for each possible δ, the *difference between iteration indexes* of T_i and T_j, as formalized in Inequalities (17) and (18). Inequality (19) represents all precedence constraints given by graph G'.

If the LP finds a feasible solution for any integer $\delta \in \langle -\hat{q}_{max}, \hat{q}_{max} \rangle$, the tasks T_i and T_j can eventually cause conflict on their dedicated processor and then the corresponding Double–Inequality (14) is necessary.

$$\begin{aligned} s_i - s_j + \delta \cdot \tau &< p_j, & (17)\\ s_j - s_i - \delta \cdot \tau &< p_i, & (18)\\ s_k - s_l &\geq l'_{kl} - \tau \cdot h'_{kl},\ \forall e'_{kl} \in G' & (19) \end{aligned}$$

where $s_i \in \langle 0, (\hat{q}_{max} + 1) \cdot \tau + p_i \rangle$

This elimination leads to a decrease in the number of binary decision variables $\hat{x}_{ij}$.

4 Parallelization of FI-CMA

As already mentioned, the dynamic range of data in FI-CMA algorithm requires calculations in the floating-point. That is, however, rather costly for an FPGA implementation. In this work, we consider two arithmetic libraries HSLA and FP32.

The High-Speed Logarithmic Arithmetic (HSLA) library, implementing the logarithmic arithmetics, consists of fully pipelined blocks, implementing basic arithmetic operations. Multiplication, division and square-root are – in logarithmic representation – implemented as simple fixed-point addition, subtraction and right shift. Addition and subtraction transform to non-linear functions that require more complicated evaluation using look-up tables.

These tables are stored in a dual-ported Block RAM (BRAM). With the necessary additional logic, the unit consumes – from all log-operations – most logic cells (slices) and BRAMs of the FPGA device. In order to utilize the memory access efficiently, the addition and subtraction macros have been implemented as a twin-adder. For more details see [4]. The HSLA library has been implemented with 19- and 32-bit wordlength, the parameters of both versions are summarized in Table 1.

The 32-bit pipelined floating-point library from Celoxica (FP32) [3], uses the widely known IEEE format to store the data. It has comparable precision with the 32-bit version of HSLA, but different timing parameters (see Table 4 in Appendix C). Namely, HSLA uses very simple (and cheap in resources) implementation of the MUL, DIV and SQRT units, while the ADD unit is expensive in clock cycles, SLICEs and BRAMs. Therefore, HSLA can be modeled by one dedicated processor, since the MUL, DIV and SQRT units are not considered critical resource and they are not subject of processor constraints (see Subsection 3.2). The FP32 library, on the other hand, has all units of non negligible size (but it can be run at a higher clock frequency than HSLA). Therefore, it has to be modeled by several dedicated processors.

To demonstrate the scheduling method, we use FI-CMA algorithm. But this method is universal for all iterative algorithms with matrix operations or imperfectly nested loops.

According to the results obtained by the scheduling method presented in this paper we decided to implement FI-CMA algorithm with HSLA library, as explained in Section 5.1. Therefore, the parameters of HSLA are used in all examples even though we have modeled and calculated both cases (see internal report [5]).

The FI-CMA algorithm consists of two successive parts: the input data matrix is processed by the *QR-decomposition algorithm* to obtain matrix $\mathbf{Q}$ (see equation (5)), which is further used by the *equalizer algorithm* (based on equations (6)). Matrix $\mathbf{Q}$ of N rows and M columns is known after the last iteration of the QR-decomposition, therefore, both parts are scheduled separately.

Both parts are iterative algorithms, and in this section we show how the cyclic scheduling (given in Section 3) can be used for their parallelization. The parallelization of QR-decomposition algorithm by processing time fusion is shown in Subsection 4.2. Subsection 4.1 shows the parallelization of the equalizer algorithm by united edges and processing time fusion.

4.1 Parallelization of Equalizer Algorithm

The equations (6) are implemented as an equalizer algorithm, shown in Figure 4(a) and in pseudo code in Appendix A, while denoting $||v_{i+1}||$ by α and $(\mathbf{Q}^T \cdot \mathbf{y}^3)$ by $\mathbf{v}_\Delta$. The parallelism in the loop is increased while substituting $\mathbf{y}$ with $(\mathbf{y}' \cdot \alpha)$. This substitution increases the number of multiplications, but it significantly increases the parallelism among additions. The corresponding data dependencies are shown in Figure 4(b).

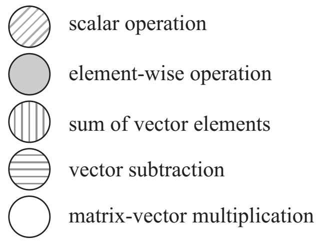
4.1.1 Expansion of Imperfectly Nested Loops

A single node of the *condensed graph* G^c (see Figure 4(b)) represents the set of tasks performing e.g. vector addition, vector multiplication, scalar addition, ... With respect to further efficient implementation, we do not want to simply expand these nodes into scalar operations but we want to keep them in a vector form intending to implement them as computation loops. The reduced condensed graph G'^c, shown in Figure 4(c), is obtained from Figure 4(b) while reducing the nodes not executed on the twin-adder, i.e. the only dedicated processor on HSLA architecture.

Our expansion of imperfectly nested loops is based on the expansion of G'^c to graph G' (see Figure 6). The second level of nesting is modeled by the *processing time fusion* while fully utilizing the twin-adder. The first level of nesting is modeled using *united edges* and an *approximated expansion*. Further, G' can be scheduled by ILP in Figure 3 while partially fixing the precedence constraints.

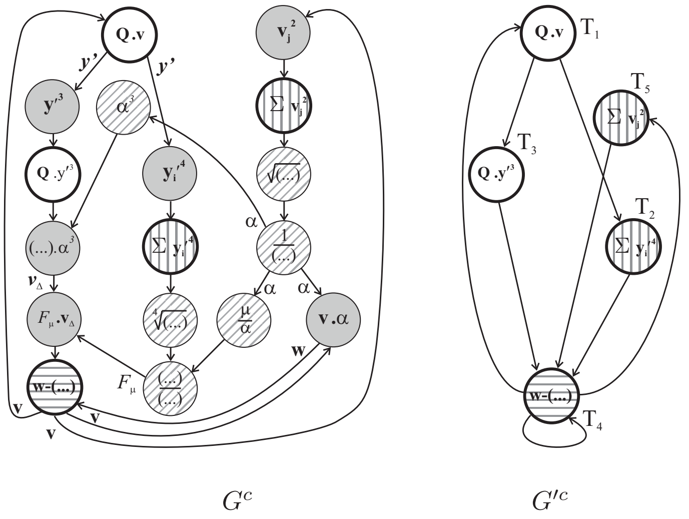
Processing Time Fusion When there are no inter-iteration dependencies inside the nested loop, one can use the following *processing time fusion*: to model each operation as one task with processing time p_i equal to the number of iterations $|\mathscr{I}|$ multiplied by p_i^s, the processing time of single operation. Therefore, different iterations of particular operation are assumed to be executed sequentially within the task. The precedence relations keep the same structure as data dependencies in the loop and the value of the length l_{ij} is increased by $max\{0, (|\mathscr{I}| - 1) \cdot (p_i^s - p_j^s)\}$. The value of the height h_{ij} is equal to zero, since there are no inter-iteration dependencies.

The expansion of nested loops will be illustrated on task T_1 in G'^c, which computes matrix-vector multiplication $\mathbf{y}' = \mathbf{Q} \cdot \mathbf{v}$. When the multiplication is evaluated in a common form, i.e. row-wise with



```
for k=1 to K do        //iterative loop
    $\mathbf{y}'(k) = \mathbf{Q} \cdot \mathbf{v}(k-1)$
    $\alpha(k-1) = \frac{1}{\sqrt{\sum \mathbf{v}(k-1)^2}}$
    $\mathbf{w}(k-1) = \mathbf{v}(k-1) \cdot \alpha(k-1)$
    $F_\mu(k) = \frac{\mu}{\sqrt[4]{\sum \mathbf{y}'(k)^4} \cdot \alpha(k-1)}$
    $\mathbf{v}_\Delta(k) = \mathbf{Q}^T \cdot \mathbf{y}'(k)^3 \cdot \alpha(k-1)^3$
    $\mathbf{v}(k) = \mathbf{w}(k-1) - F_\mu(k) \cdot \mathbf{v}_\Delta(k)$
end
```

(a)



G^c (b)

G'^c (c)

Figure 4: (a) Equalizer algorithm. (b) Data dependencies of the algorithm represented by the condensed graph G^c. (c) Corresponding reduced condensed graph G'^c.

respect to matrix $\mathbf{Q}$, the efficiency of the resource utilization is relatively small. This is caused by the relatively big input-output latency of the twin-adder with respect to the row length. We propose to compute the multiplication in column-wise form, where all the elements of the j^{th} column are multiplied by a j^{th} element of vector $\mathbf{v}$. The partial sums are stored in the memory. For example, the computations of all rows in the 1^{st} column (second level of nesting) are represented by task $T_{1,1}$ in G' in Figure 5. Task $T_{1,1}$ represents the fused ADD operations of the 1^{st} column, and therefore, its processing time $p_{1,1}$ is equal to N.

United Edges The outer loop over the columns, further called *column loop* (first level of nesting), is modeled using *united edges* as follows. Each task T_i in G'^c is expanded to set of tasks $T_{i,j}$ representing single iterations $\mathscr{I}_i$. Index i denotes original index of expanded task T_i and index $j \in \{1, \ldots, |\mathscr{I}_i|\}$ denotes the iterations modeled by loop expansion.

In order to keep regularity of the loop implementation, the time delay between consequent iterations $T_{i,j}$ and $T_{i,j+1}$ must be the same. Therefore, for all $j \in \{1, \ldots, |\mathscr{I}_i| - 1\}$ we introduce a new kind of edge from $T_{i,j}$ to $T_{i,j+1}$. These *united edges*, introduced due to expansion of T_i are grouped in one group o. Thereafter, the new precedence constraints are expressed by equalities

$$s_{i,j+1} - s_{i,j} - z_o = l_o, \quad \forall j \in \{1, \ldots, |\mathscr{I}_i| - 1\}$$

where z_o is one variable of ILP, such that $z_o \geq 0$, and constant l_o is the length of the united edge. Both z_o and l_o are common for all united edges belonging to the group o. Then $\tau_o = z_o + l_o$ is a period of the column loop belonging to group o. This formalism enables to handle inter-iteration dependencies. The height of all united edges belonging to the group o is equal to 0 since each iteration is represented by particular task.

As iterations of the first level nested loop are identical, odd iterations are processed on the first half of the twin-adder and even iterations on the second one. Thereafter, both halves of the twin-adder process the same tasks in the same order, but they are shifted by 9 clocks, i.e. input-output latency of ADD unit in HSLA library. Therefore, the model consists of $M/2$ iterations of the column loop.

Approximated Expansion The scheduling algorithm (in Figure 3) for model with processing time fusion and united edges can find such an inner period τ_o so that the global schedule is optimal. To reduce the number of tasks $(T_{1,1}, T_{1,2}, ..., T_{1,M/2})$ representing expanded iterations of the nested loop processed on the first twin-adder, we propose the following *approximated expansion*. We divide the iterations of the nested loop $\mathscr{I}_i$ into a *prologue* $\mathscr{P}_i$,

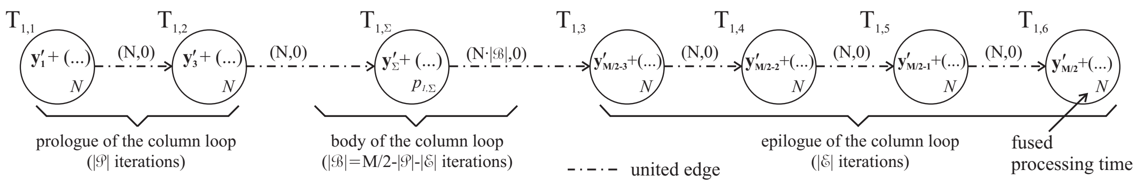


Figure 5: Approximated expansion of the column loop (first level of nesting) of matrix-vector multiplication $\mathbf{y}' = \mathbf{Q} \cdot \mathbf{v}$.

a *body* $\mathscr{B}_i$ and an *epilogue* $\mathscr{E}_i$. The prologue and epilogue contain $|\mathscr{P}_i|$ (respectively $|\mathscr{E}_i|$) iterations, modeled as explained above.

The body contains one task which prevents any other task to be executed. In fact it represents $|\mathscr{B}_i|$ iterations, i.e. $|\mathscr{B}_i| = |\mathscr{I}_i| - |\mathscr{P}_i| - |\mathscr{E}_i|$, where $|\mathscr{I}_i|$ is number of modeled iterations (in case of task T_1 $|\mathscr{I}_1| = M/2$). Processing time of the body task $p_{i,\Sigma}$ is variable and it is equal to $p_{i,\Sigma} = |\mathscr{B}_i| \cdot p_{i,1} + (|\mathscr{B}_i| - 1) \cdot z_o$. United edge from the body task $T_{i,\Sigma}$ to the first task of epilogue $T_{i,\mathscr{E}}$ is given by the following precedence constraint

$$s_{i,\mathscr{E}} - s_{i,\Sigma} - |\mathscr{B}_i| \cdot z_o = |\mathscr{B}_i| \cdot l_o. \qquad (20)$$

The utility of the approximation depends on the estimation of the prologue and the epilogue length permitting sufficient overlap of single operations.

When there are two loops that can be processed in parallel while sharing one dedicated unit, (e.g. expanded tasks T_2 and T_3 in Figure 6) then their interleaving has to be carefully implemented since, in the model, the body of one loop blocks the dedicated unit. Such sharing is possible, when both loops have the same inner period τ_o, i.e. their united edges are grouped in the same group o. Thereafter there is just one body task for both loops, since the role of body task is to block the shared unit.

4.1.2 Simplification of memory access by fixed edges

The previous section described modeling of the first and the second level of nesting by expansion of the condensed graph G'^c to G'. The model considers restriction to the one ADD unit only. In fact such model does not reflect access into memory containing variables of the equalizer algorithm. The only variable, that can cause conflict with respect to access into memory is vector $\mathbf{y}'$ stored in BRAM. Other variables can not be accessed in parallel or they are located in a distributed memory where the number of accesses is not restricted.

Therefore, we assume two dedicated processors. The first one is the twin-adder and the second one is the dual ported BRAM containing vector $\mathbf{y}'$. The accesses into the memory containing $\mathbf{y}'$ are modeled as new tasks labeled "READ" in Figure 6. Data read from the dual ported BRAM are directly processed in the twin-adder in order to avoid temporal storage of data in the register. Therefore, precedence constraints related to this data are no longer given by the inequality (11), but they are given by the equation $s_j - s_i = l_{ij} - \tau \cdot h_{ij}$. Such an edge, represented by a dashed line, is called a *fixed edge* in the rest of this text.

4.1.3 Equalizer Algorithm Schedule

The resulting equalizer algorithm model by the reduced graph G' including accesses into the memory (fixed edges) and expansion of nested loops (united edges) is shown in Figure 6.

The particular constants M and N are chosen to be greater than the input-output latency of the pipelined twin-adder (i.e. $M = 20$ and $N = 24$). The schedule of this instance of cyclic scheduling is shown in Figure 7. While setting greater values of constants M and N, we obtain different parameters of graph G' and an unclear Gantt chart, therefore, we have shown this example.

The model in Figure 6 considers two processors, i.e. the first half of twin-adder and the dual ported BRAM containing vector $\mathbf{y}'$. But it corresponds to the HW design with both parts of the twin-adder, since odd iterations are processed on the first half of the twin-adder and even iterations on the second one. This mapping scheme known from parallel computing can be easily used to extend this model to a different number of ADD units. The advantage of such a representation lies in smaller corresponding ILP model and therefore, lower computing time requirements. Moreover, it leads to a simpler code structure and therefore a smaller FPGA design.

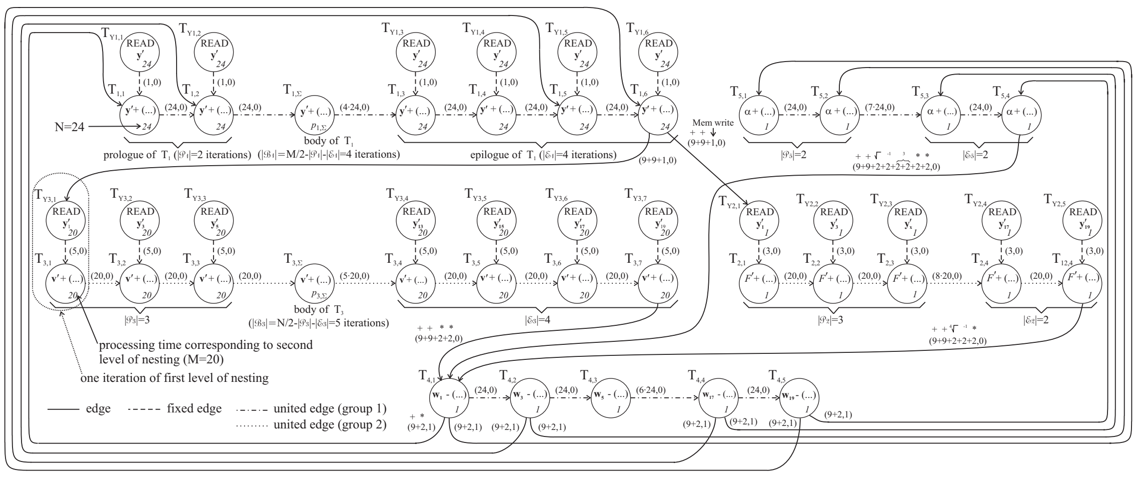


Figure 6: Model of the equalizer algorithm by the reduced graph G' with fixed edges and united edges on HSLA library.

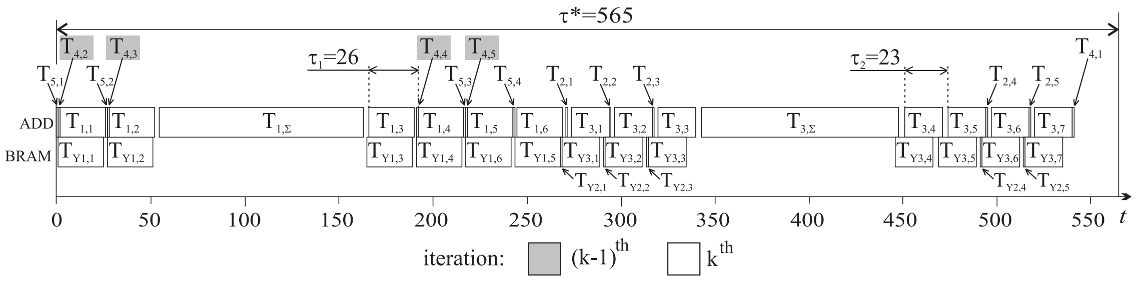


Figure 7: Resulting schedule of the equalizer algorithm ($\tau^* = 565$; inner periods $\tau_1 = 26$, $\tau_2 = 23$) on two dedicated processors (one half of HSLA twin-adder, and BRAM containing $\mathbf{y}'$).

4.2 Parallelization of the QR decomposition

The QR decomposition algorithm (Figure 8(a)) contains an outer loop, an iterative loop and nested loops B,C. The outer loop is repeated for each input data sample and it can not be subject of the cyclic scheduling, since two iterations of the outer loop can not be executed simultaneously. The iterative loop is subject of the cyclic scheduling. The nested loops B,C (both on the first level of nesting) have no inter-iteration dependencies, therefore they are modeled by the processing time fusion while fully utilizing the twin-adder.

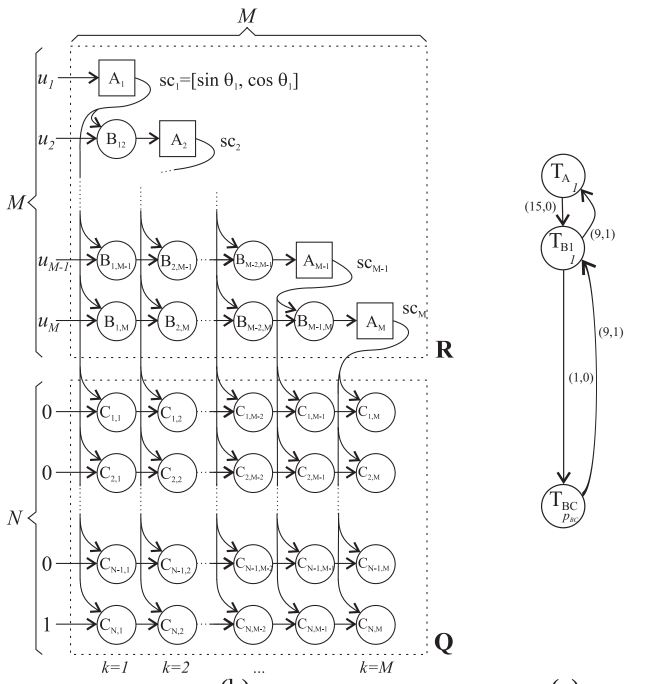
The k-th iteration of the iterative loop computes one diagonal element of $\mathbf{R}$ (single cell A), off diagonal elements of $\mathbf{R}$ (cells B computed in nested loop B) and from one to i elements of $\mathbf{Q}$ (cells C computed in nested loop C). Hence, the number of lines of $\mathbf{Q}$ grows up to i, the iteration index of the outer loop. Figure 8(b) shows the data flow graph of the N-th (the last) iteration of the outer loop, consisting of the cells arranged in the upper triangular matrix $\mathbf{R}$, flipped over the main diagonal, and of the matrix $\mathbf{Q}$.

Assuming HSLA arithmetic library, we propose to model the dataflow in Figure 8(b) by G' in Figure 8(c). The iterative loop (executed over columns in Figure 8(b)) is modeled by the cyclic scheduling. The nested loops B and C (executed over rows in Figure 8(b)) are modeled by the *processing time fusion extended by preemption*. The idea is to combine the operations across loops B and C as one preemptible task, and the scheduling algorithm chooses the optimal preemption point.

However, the number of operations varies with both k (the iteration index of the iterative loop) and i (the iteration index of the outer loop), which precludes direct application of the cyclic scheduling. Therefore G' in Figure 8(c) consists of:

```
for $i = 1$ to $N$ do                              //outer loop
    $u = \text{data}_{i:(i+M)}$;                   //input data
    $t = zeros(i,1)$;
    $t_i = 1$;
    for $k$=1 to $M$ do                            //iterative loop
        [$r^2_{k,k}$, $sc$] = A($r^2_{k,k}$,$u_k$);
            //computation of diagonal element of R
        for $j = k+1$ to $M$ do                    //nested loop B
            [$r_{k,j}$,$u_j$] = B($r_{k,j}$,$u_j$,$sc$);
                //computation of off-diagonal element of R
        end
        for $j = 1$ to $i$ do                      //nested loop C
            [$q_{j,k}$,$t_j$] = C($q_{j,k}$,$t_j$,$sc$);
                //computation of elements of Q
        end
    end
end
```

(a)



(b) (c)

Figure 8: (a) The algorithm of QR decomposition. (b) The data flow graph of the iterative loop (N-th iteration of the outer loop). (c) The reduced graph G' corresponding to the model of QR decomposition.

- task T_A corresponding to cell A
- task T_{B1} corresponding to the first cell B, taking data from the cell A and generating data for the cell A in the next iteration
- task T_{BC} corresponding to the rest of the cells B and to all cells C

For a given column, task T_{BC} represents all cells B (except the first one) and cells C, all sequentially processing data in the nested loop B and nested loop C. No expansion of the nested loop is needed at this point since the sequential processing fully utilizes the twin-adder, therefore it is optimal. The processing time of T_{BC} is given as a sum of the processing times of the corresponding cells.

The processing time of tasks T_A resp. T'_{B1} is $p_A = 1$ resp. $p_{B1} = 1$ since they correspond to one cell. The processing time p_{BC} of T_{BC} depends on the number of iterations of iterative and outer loop therefore $p_{BC} \in \langle 0, M+N-2 \rangle$. Since we consider iterative algorithms with fixed period, task T_{BC} is modeled with $p_{BC} = 42$ ($M = 20$ and $N = 24$).

This graph has critical circuit with $\tau^* = 24$. Further, the optimal schedule of one iteration must contain a time gap of 14 clock cycles between tasks T_A and T_{B1} (since it takes T_{B1} the latency of the pipelined unit to get the result of T_A). In order to be able to use the time gap, the task T_{BC} is divided into tasks T'_{BC} and T''_{BC} with variable processing times satisfying new linear constraint $p'_{BC} + p''_{BC} = p_{BC}$. The division of T_{BC} into T'_{BC} and T''_{BC} can be considered as a preemption of T_{BC}.

The resulting schedule is shown in Figure 9. For the regularity reasons we suppose the order of tasks in the schedule to be the same for all iterations in the iterative loop.

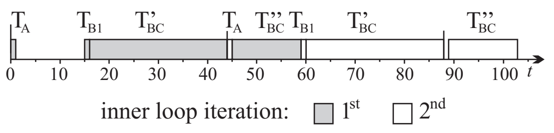


Figure 9: Resulting schedule for two iterative loop iterations of QR decomposition ($\tau^* = 44$).

We are able to consider preemption since the processing times (p'_{BC} and p''_{BC}) can be variables in our ILP model, that is not permitted in common ILP models, e.g. in [12].

5 Experimental Results

5.1 Implementation Results

The FICMA algorithm has been implemented in two FPGA devices: Xilinx Virtex XCV2000E-6 and Xilinx Virtex-II XC2V1000-5. The schedules (in Figure 7 and 9), containing start times of operations, were further used to automatically generate a state machine controlling a parallel execution of the algorithm using Handel-C language. It has been compiled using the compiler from the Celoxica DK3 suite. The resulting VHDL description was processed with Simplify Pro and Xilinx ISE 6.1.03 to get the bitstream for FPGA. The results of the implementation (i.e. equalizer algorithm and QR decomposition) are shown in Table 2.

The choice of architecture to be used was based on two models, one for HSLA (Figure 7) resulting in the period $\tau^* = 565$ and one for the FP32 (see internal report [5]) resulting in the period $\tau^* = 1180$. Considering the clock cycle length (measured in the case of HSLA and estimated in the case of FP32) we were able to determine which architecture leads to faster implementation. Based on these results, we have decided to use the HSLA library. In order to save the on-chip memory, we have decided for the 19-bit version of the library. The performance of the equalizer is, of course, worse than with 32-bit precision, nevertheless still sufficient [29].

The implementation was tested on a data matrix of the size 24×20. The FI-CMA algorithm uses one twin-adder and four MUL units, which results in 300 MFlops for Virtex-II (210 MFlops for Virtex-E). DIV and SQRT operations are neglected, because of their spare use (the resulting error in MFLOPS is 0.3 per mille).

One iteration of equalizer update procedure takes 565 cycles which represents the execution time 11μs for Virtex-II (16μs for Virtex-E). This performance is sufficient for the symbol period 3.7μs (corresponding to 24 symbols processed in one batch in GSM systems), since the implementation of equalizer update is fast enough to perform 8 iterations, needed for the algorithm convergence.

5.2 Results of Scheduling

The presented scheduling technique was tested on an Intel Pentium 4 at 2.4 GHz using the non-commercial ILP solver GLPK [30] and the commercial ILP solver CPLEX [31]. The equalizer algorithm model in Figure 6 consists of 47 tasks on two distinct dedicated processors (29 on the first half of twin-adder and 18 on the dual ported Block RAM containing vector $\mathbf{y}'$). The upper bound of $\hat{q}_i$ was given *a priory* for tasks $T_{1,i}$ and $T_{5,i}$ equal to 0 and for $T_{3,i}$, $T_{2,i}$ and $T_{4,i}$ equal to 1. The lower bound of period τ, $\tau_{lower} = 493$ is given by critical circuit in G'. The upper bound $\tau_{upper} = 774$ was calculated using the ILP model (Figure 3) for the reduced condensed graph G'^c (in Figure 4(c)), where all tasks in the column loop are supposed to be executed simply, in sequence.

The resulting schedule with optimal period $\tau^* = 565$ was obtained by the interval bisection method in 9 iterative calls of the ILP. The corresponding ILP model contains 621 variables. The time required to compute the optimal solution without elimination of redundant processor constraints, given as a sum of iterative calls of the ILP solver was 1634 s with GLPK. This time does not include the construction of the ILP model, since it is negligible from the complexity point of view. The time required to compute the optimal solution with elimination of redundant processor constraints (see Subsection 3.4) was 57.8 s with GLPK and only 3.4 s with CPLEX. In this case, as many as 436 variables were eliminated (the number of eliminated redundant inequalities is twice as big), which is mainly due to the ILP model keeping a regular structure of matrix operations.

The experiments are summarized in Table 3

Table 2: Resource utilization of the FI-CMA implementation with 19-bit HSLA library and the corresponding XC2V1000-5 design.

	XCV2000E-6		**XC2V1000-5**	
Block RAM	16	10 %	16	40%
SLICEs	4349	22%	4222	82%
TBUFs	192	1%	192	7%
Clock Rate	35 MHz		50 MHz	
Performance	210 MFlops		300 MFlops	

Table 3: Overview of scheduling results for equalizer algorithm. Computation times were obtained on CPLEX solver.

	HSLA	**FP32**
n'	47	82
m'	2	4
$\#var$	621	1556
$\#var_{elim}$	436	1038
τ^*	565	1180
$\#iter$	9	11
$CPU\ time$	6.1 s	124.6 s
$CPU\ time_{elim}$	3.4 s	55.1 s

where n' denotes the number of tasks after reduction, m' denotes the number of dedicated processors, $\#var$ denotes number of ILP variables. The algorithm results are given by the shortest period resulting in a feasible schedule τ^*, and by vector s. The row $\#iter$ denotes the number of iterative calls of ILP and $\#var_{elim}$ denotes the number of eliminated variables of ILP for τ^* (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 row $CPU\ time$. $CPU\ time_{elim}$ is the time to compute the optimal solution with eliminated redundant inequalities. The improvement on $CPU\ time$ by using redundant inequalities elimination is 44% in the case of HSLA and 56% in the case of FP32.

Since the model of QR decomposition contains only 4 tasks, time required to compute the optimal solution (without elimination of redundant processor constraints) was $0.001s$. The optimal period of resultant schedule is $\tau^* = \tau_{lower} = 44$ since $\tau_{lower} = 44$ is given as the maximum sum of the processing times of tasks dedicated to one processor.

6 Conclusions

This paper presented a new scheduling method based on mathematical programming used to optimize the computation speed of real-time iterative algorithms with matrix operations running on architectures including distinct dedicated processors with restricted access into memory. The contribution lies in the representation of imperfectly nested loops and memory access in the form of linear inequalities that are incorporated in the cyclic scheduling framework based on ILP. In order to compare two different arithmetic libraries, we have extended our previous work in this paper, cyclic scheduling on one dedicated processor, to its multiprocessor version. Since the solutions of ILP formulated problems are known to be computationally intensive, important part of the article is devoted to the reduction of the problem size. In this article and in internal report [5] we have shown that our elimination of redundant processor constraints plays an important role in the case of the iterative algorithms with matrix operations.

As shown e.g. in Figure 7, this approach leads to a schedule overlap (operations belonging to different iterations are interleaving), which is rather difficult to achieve in a manual design. The cyclic scheduling is not directly applicable to complex parallelization problems containing imperfectly-nested loops, and access to Block RAMs. Therefore, we have proposed a model containing processing time fusion, united edges and fixed edges. Even though our method is not fully automated (i.e. programmer intervention is needed in the nested loop modeling phase), the algorithm modeling, transformation and scheduling could be easily incorporated in design tools while representing considerable simplification for rapid prototyping of real-time applications and evaluating their performance prior to their implementation.

The method was used to find a schedule for FPGA implementation of the Finite Interval Constant Modulus Algorithm (FI-CMA) using Xilinx Virtex-E and Virtex-II devices. The algorithm, which requires floating-point computations to achieve sufficient accuracy, has been implemented using the High Speed Logarithmic Arithmetic (HSLA) library.

The first implementation of the algorithm, has been described in [27]. Using our ILP based scheduling for equalizer order $M = 20$ and size of the block $N = 24$, the number of cycles needed for one iteration of the equalizer update has been improved by approximately 43% (from 994 cycles to 565 cycles). In case of QR-decomposition improvement of one iteration length was about 27.7% (from 10559 cycles to 7633 cycles).

From these numbers, it is clear that the implementation of equalizer update is fast enough to perform 8 iterations, needed for the algorithm convergence. On the other hand the weak point is QR-decomposition, which can be solved by implementing several QR-decomposition blocks operating on different data segments in parallel.

In our current work in the scheduling of algorithms for efficient FPGA design we have solved two related problems: (i) scheduling on dedicated sets of identical processors (e.g. 3 MUL units and 2 ADD units) (ii) variable number of processors constrained by the FPGA resource (e.g. number of MUL and ADD units is also subject of optimization, which is constrained by available SLICEs and BRAMs). Solution to both of these problems has more complex ILP models but the model of nested loops stays the same. We also work on alternative solutions by polynomial heuristic algorithms enabling to solve considerably larger problems. These works are the subject of our ongoing publications.

Acknowledgement

This work been partially supported by the Grant Agency of the Academy of Sciences of the Czech Republic under Project 1ET300750402 and by the Ministry of Education of the Czech Republic under Project 1M0567. The authors would like to thank the anonymous referees for providing many invaluable comments and suggestions that lead to significant improvement of this paper.

References

[1] D. N. Godard. Self-recovering equalization and carrier tracking in two-dimensional data communication systems. *IEEE Trans. Communications*, 28:1867–1875, November 1980.

[2] P. A. Regalia. A finite interval constant modulus algorithm. In *Proc. International Conference on Acoustics, Speech, and Signal Processing(ICASSP-2002)*, volume III, pages 2285–2288, Orlando, FL, May 13-17 2002.

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

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

[5] P. Šůcha and Z. Hanzálek. Optimization of iterative algorithms with matrix operations:case studies. Technical report, CTU FEL DCE, Prague, 2005. http://dce.felk.cvut.cz/sucha/articles/sucha05ficmaCS.pdf.

[6] M.A. Bayoumi, G.A. Jullien, and W.C. Miller. Hybrid VLSI architecture of FIR filters using residue number systems. *Electronics Letters*, 21(8):358–359, January 1985.

[7] J.G. McWhirter. Systolic array for recursive least-squares minimisation. *Electronics Letters*, 19(18):729–730, 1983.

[8] I.K. Proudler, J.G. McWhirter, M. Moonen, and G. Hekstra. The formal derivation of a systolic array for recursive least squares estimation. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(3):247–254, 1996.

[9] M. Moonen, P. Van Dooren, and J. Vandewalle. Systolic algorithm for QSVD updating. *Signal Processing*, 25(2):203–213, 1991.

[10] G. Lightbody, R. Walke, R. Woods, and J. McCanny. Parameterizable qr core. In *Asilomar Conference on Signals, Systems and Computers, Conference Record*, volume 1, pages 120–124, 1999.

[11] R.L. Walke and R.W.M. Smith. 20 GFLOPS QR processor on a Xilinx Virtex-E FPGA. In Franklin T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations X*, volume 4116. SPIE, 2000.

[12] 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. In *PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands*, 2000.

[13] C. Hanen and A. Munier. A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57:167–192, February 1995.

[14] A. Munier. The complexity of a cyclic scheduling problem with identical machines. *European Journal of Operational Research*, 91:471–480, June 1996.

[15] Dirk Fimmel and Jan Müller. Optimal software pipelining under resource constraints. *International Journal of Foundations of Computer Science*, 12(6):697–718, 2001.

[16] P. Šůcha, Z. Pohl, and Z. Hanzálek. Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, Canada*, 2004.

[17] Z. Pohl, P. Šůcha, J. Kadlec, and Z. Hanzálek. Performance tuning of iterative algorithms in signal processing. In *The International Conference on Field-Programmable Logic and Applications (FPL'05), Tampere, Finland*, August 2005.

[18] 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*, pages 318–328, 1988.

[19] B. R. Rau and C. D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *MICRO 14: Proceedings of the 14th annual workshop on Microprogramming*, pages 183–198, Piscataway, NJ, USA, 1981. IEEE Press.

[20] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'04), Paris, France*, February 2004.

[21] A. Darte and Guillaume Huard. Loop shifting for loop compaction. *International Journal of Parallel Programming*, 28(5):499–534, 2000.

[22] S. Carr, C. Ding, and P. Sweany. Improving software pipelining with unroll-and-jam. In *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96)*, January 1996.

[23] D. Petkov, R. Harr, and S. Amarasinghe. Efficient pipelining of nested loops:unroll-and-squash. In *16th International Parallel and Distributed Processing Symposium (IPDPS'02), Fort Lauderdale, California*, April 2002.

[24] M. J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[25] N. Ahmed, N. Mateev, and K. Pingali. Tiling imperfectly-nested loop nests. In *Proceedings of the IEEE/ACM SC2000 Conference, Dallas, Texas*, November 2000.

[26] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. Rau, D. Cronquist, and M. Sivaraman. Pico-npa: High-level synthesis of nonprogrammable hardware accelerators. *The Journal of VLSI Signal Processing*, 31(2):127–142, 2002.

[27] A. Heřmánek, J. Schier, and P. A. Regalia. Architecture design for FPGA implementation of Finite Interval CMA. In *Proc. European Signal Processing Conference*, pages 2039–2042, Wiena, Austria, September 2004.

[28] W. Givens. Computation of plane unitary rotations transforming a general matrix to triangular form. *J. Soc. Ind. Appl. Math.*, 6:26–50, 1958.

[29] A. Heřmánek. *Study of the next generation equalization algorithms and their implementation*. PhD thesis, Université Paris XI, UFR Scientifique d'Orsay, 2005.

[30] A. Makhorin. *GLPK (GNU Linear Programming Kit) Version 4.6*, 2004. http://www.gnu.org/software/glpk/.

[31] ILOG, Inc. *CPLEX Version 8.0*, 2002. http://www.ilog.com/products/cplex/.

A Equalizer Algorithm

```
for $k = 1$ to $K$ do                              //iterative loop

    $y'_i = 0\ \forall i \in \langle 0, N\rangle$
    for $i = 1$ to $M$ do                          //first level of nesting
        for $j = 1$ to $N$ do                      //second level of nesting
            $y'_j = y'_j + Q_{i,j} \cdot v_i$      //$y'(k) = Q \cdot v(k-1)$
        end
    end

    $sum\_v = 0$
    for $i = 1$ to $M$ do
        $sum\_v = sum\_v + v_i^2$
    end
    $\alpha = 1/\sqrt{sum_v}$                      //$\alpha(k) = 1/\sqrt{\sum v(k-1)^2}$

    for $i = 1$ to $M$ do
        $w_i = v_i \cdot \alpha$                   //$w(k) = v(k-1) \cdot \alpha(k-1)$
    end

    $sum\_y' = 0$
    for $j = 1$ to $N$ do
        $sum\_y' = sum\_y' + (y'_j)^4$
    end
    $F_\mu = \mu/(\sqrt[4]{sum\_y'} \cdot \alpha)$
                              //$F_\mu(k) = \mu/(\sqrt[4]{\sum \mathbf{y}'(k)^4} \cdot \alpha(k-1))$

    $vd_i = 0\ \forall i \in \langle 0, M\rangle$
    for $j = 1$ to $N$ do
        for $i = 1$ to $M$ do
            $vd_i = vd_i + Q_{i,j} \cdot (y'_j)^3$
        end
    end
    for $i = 1$ to $M$ do
        $vd_i = vd_i * \alpha^3$     //$\mathbf{v}_\Delta(k) = \mathbf{Q}^T \cdot \mathbf{y}'(k)^3 \cdot \alpha(k-1)^3$
    end

    for $i = 1$ to $M$ do
        $v_i = w_i - F_\mu \cdot vd_i$
                              //$\mathbf{v}(k) = \mathbf{w}(k-1) - F_\mu(k) \cdot \mathbf{v}_\Delta(k)$
    end

end
```

B List of Variables

$\mathbf{b}_n$	background noise vector
e_{ij}	edge e_{ij} from the node i to j
$\mathbf{g}$	CMA equalizer coeffitient vector
h_{ij}	height of e_{ij}
i, j, k	indices
l_{ij}	length of e_{ij}
m	number of processors (arithmetic units)
m'	number of reduced processors
n	number of tasks
n'	number of reduced tasks
o	group of united edges
p_i	processing time
p_i^s	processing time of single operation
$q_{i,j}$	element of matrix $\mathbf{Q}$
$\hat{q}_i$	the index of the execution period
$r_{i,j}$	element of matrix $\mathbf{R}$
s_i	start time of task T_i in the first iteration
$\mathbf{s_n}$	transmited data vector
$\hat{s}_i$	the index within the execution period
t	time
$\mathbf{u}_n$	received signal vector
$\mathbf{v}$	temporary value for equalizer update
$\mathbf{w}$	FI-CMA equalizer coeffitient vector
$\hat{x}_{ij}$	binary decision variable of ILP model
y_n	equalizer output
$\mathbf{y}_k$	equalizer outputs vector at time k
z_o	slack variable in ILP model
F_k	FI-CMA cost function after k iterations
G	graph
G'	reduced graph
G^c	condensed graph
$\mathbf{G}_i$	i^{th} Givens rotation in QR decomposition
$\mathbf{H}$	channel matrix
I	identity matrix
J	CMA cost function
K	number of iterations
M	equalizer order
N	size of the block of FI-CMA
P	oversampling factor
$\mathbf{Q}$	$\mathbf{Q}$ matrix of QR decomposition
$\mathbf{R}$	$\mathbf{R}$ matrix of QR decomposition
T_i	task i
$T_{i,j}$	task representing iteration j of task T_i
$\mathbf{U}$	input data vector of M past samples
$\mathscr{B}_i$	body of the nested loop i
$\mathscr{E}_i$	epilogue of the nested loop i
$\mathscr{I}_i$	iterations of the nested loop i
$\mathscr{P}_i$	prologue of the nested loop i
$\mathcal{T}$	set of tasks
$\mathcal{T}'$	set of tasks on dedicated processors
$\mathcal{T}^r$	set of reduced tasks
$\mathcal{U}$	receiver input data matrix
γ	not important constant
δ	difference between iteration indexes
Δ_{ij}	slack variable of ILP model
Δ_{ij}^b	binary decision variable of ILP model
θ_i	rotation angle in Givens elimination
μ_k	step-size in equalizer update at time k

τ	period
τ^*	optimal period
τ_o	optimal inner period

C Parameters of FP32

Table 4: Parameters of the 32-bit FP32 units for XCV2000E-6 FPGA device. Maximal clock frequency is 93 MHz.

Unit	**Input-output Latency [clk]**	**SLICEs [%]**	**BRAMs [%]**
ADD	11	8	0
MUL	8	4	0
DIV	28	10	0
SQRT	27	6	0

Přemysl Šůcha was born in Prague, Czech Republic, in 1978. He received the Diploma in Electrical Engineering from the Czech Technical University (CTU) in Prague in 2003. He is currently a Ph.D. student at the Czech Technical University. His research interests include scheduling and high-level synthesis.

Zdeněk Hanzálek was born in Tabor, Czech Republic, in 1967. He obtained the Diploma in Electrical Engineering from the Czech Technical University (CTU) in Prague in 1990. He obtained his PhD degree in Control Engineering from the CTU in Prague and PhD degree in Industrial Informatics from the Universite Paul Sabatier Toulouse. He was with LAAS - Laboratoire d'Analyse et d'Architecture des Systemes in Toulouse (1992 to 1997) and with LAG INPG - Institut National Polytechnique de Grenoble (1998 to 2000). In 2005, he obtained Doc. degree at the Czech Technical University in Prague. His research interests include real-time systems and scheduling.

Antonín Heřmánek is a researcher at the Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic. He received his M.S.E.E. and Ph.D. degrees from the Czech Technical University, Faculty of Electrical Engineering (1998) and Universite Paris Sud - Orsay, STITS (2005), respectively. His research interests include blind equalization, MIMO and OFDM communication systems, FPGA implementation of DSP algorithms, array processing and rapid prototyping for signal processing. He has published over 30 papers in these areas.

Jan Schier (M'2005) is a senior researcher at the Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic. He received his M.S.E.E. and Ph.D. degrees from the Czech Technical University, Faculty of Electrical Engineering (1989) and Faculty of Nuclear Sciences and Physical Engineering (1995), respectively. His research interests include FPGA implementation of signal processing algorithms, array processing and rapid prototyping for signal processing. He has published over 20 papers in these areas. Dr. Schier has been a Visiting researcher at the Delft University of Technology and at the Katholieke Universiteit Leuven.