Scheduling of a LQ Control Algorithm for Efficient FPGA Implementation ⋆

Přemysl Šůcha * **Zdeněk Hanzálek** *

* *Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic (Tel: 420-2-2435-5714; e-mail: {suchap,hanzalek}@fel.cvut.cz)*

Abstract: This paper deals with the speed optimization of iterative algorithms with matrix operations or nested loops for hardware implementation in Field Programmable Gate Arrays (FPGA). The presented scheduling algorithm use Integer Linear Programming (ILP) while complex algorithm structure is modeled by system of linear inequalities. The method is demonstrated on a LQ control algorithm. An advantage of the presented scheduling method is its suitability for algorithms with longer iteration period.

1. INTRODUCTION

This paper deals with the scheduling of iterative control algorithms, where K iterations have to be executed each sampling period in order to achieve the desired sampling period. Such control algorithms usually contain rather complex data computations, usually expressed in the form of matrix operations and implemented as nested loops. In this paper we deal with a target hardware based on FPGA (constituted by several pipelined specialized arithmetic units). The presented method is aimed to schedule described control algorithms on FPGA architectures. We solve the scheduling problem by Integer Linear Programming (ILP), while achieving the optimal schedule, i.e. schedule with minimal sampling period. Thanks to an efficient representation of imperfectly nested loops we are able to handle scheduling problems of significant size as demonstrated on LQ control algorithm.

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 Cel [2004]. 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 Matoušek et al. [2002]. 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.

1.1 Related Work

An iterative algorithm can be seen as a computation 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 (within sampling period). If operations belonging to different iterations interleave, the schedule is called *overlapped* Sindorf and Gerez [2000]. 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 Hanen and Munier [1995]. For a fixed number of processors the problem becomes NP–hard Hanen and Munier [1995]. The general solution to this problem is shown e.g. in Fimmel and Müller [2001]. Cyclic scheduling presented in Šůcha et al. [2004] is not dependent on the period length and with respect to Fimmel and Müller [2001] 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 Pohl et al. [2005]. *Modulo scheduling* and *software pipelining* [Rau and Glaeser, 1981] are related terms to *cyclic scheduling*, which are usually used in the compiler community.

For practical reasons, we usually do not want to expand matrix operations inside iterative loop 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

⋆ This work was supported by the Ministry of Education of the Czech Republic under research programme MSM6840770038 and by the ARTIST2 Network of Excellence on Embedded Systems Design IST-004527.

shifting (operations from one iteration of the loop body are moved to its previous iteration) was recently extended in Gupta et al. [2004], Darte and Huard [2000]. Another approach is the unroll-and-jam (also called unfolding or unwinding) Carr et al. [1996], 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 Petkov et al. [2002].

In the case of LQ controller, 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 Wolfe [1995]. The tiling method, extended for imperfectly-nested loops, is discussed in Ahmed et al. [2000]. 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 Wolfe [1995]. 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.

In paper Šůcha et al. [2007], we presented a 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 "processing time fusion". The method is based on the construction of an abstract model, which models the nested loops, and which is optimally scheduled using integer linear programming (ILP).

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 (time-indexed) ILP formulations of cyclic scheduling (e.g. Sindorf and Gerez [2000]), 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.

1.2 Outline

In this paper we show, how to apply our method Šůcha et al. [2007] on the LQ control algorithm. 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) Cel [2004] and High-Speed Logarithmic Arithmetics (HSLA) Matoušek et al. [2002]) and we show that by using the presented method, the optimal architecture can be chosen for a given algorithm prior to its implementation.

The paper is organized as follows: in the next section the LQ control 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. Section 4 presents scheduling of iterative algorithms with imperfectly-nested loops, illustrated by the parallelization of the LQ control algorithm. Section 5 presents experimental results and the last section concludes the work.

2. LQ CONTROL PROBLEM

The LQ (linear-quadratic) control problem, initiated by Kalman [1960], is one of the most important classes of optimal control problems, in both theory and applications. The problem, finding the optimal infinite-horizon LQ controller, lies in determination of the optimal state-feedback gain (Kalman gain $\mathbf{K}$) through the discrete-time algebraic Riccati equation. The Kalman gain can be evaluated as shown in Figure 1.

for t=K **downto** 1 **do**
$\mathbf{K}(t) = \left(\mathbf{R} + \mathbf{B}^T \cdot \mathbf{P}(t-1) \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{P}(t-1) \cdot \mathbf{A}$;
//Kalman gain
$\mathbf{P}(t) = \mathbf{A}^T \cdot \mathbf{P}(t-1) \cdot \mathbf{A} + \mathbf{Q} - \mathbf{A}^T \cdot \mathbf{P}(t-1) \cdot \mathbf{B} \cdot \mathbf{K}(t)$;
//Riccati equation
end

Fig. 1. LQ control algorithm.

T is time horizon of control and t is discrete time. Matrices $\mathbf{A}$ and $\mathbf{B}$ describes the state-space representation of the controlled system. The weighting matrices $\mathbf{Q}$ and $\mathbf{R}$ are user specified and define the trade-off between regulation performance (how fast goes to zero) and control effort. Matrix $\mathbf{P}$ is a solution of the discrete-time algebraic Riccati equation.

3. FORMULATION AND SOLUTION OF THE SCHEDULING PROBLEM

The iterative procedure, as the one mentioned in the previous section, can be implemented as a computation loop performing an identical set of operations repeatedly. Therefore our work, dealing with an optimized implementation of such procedures, is based on cyclic scheduling.

3.1 Cyclic Scheduling Problem

Operations in a computation loop can be considered as a set of n *generic tasks* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed K times where K is usually very large. 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 T_i Hanen and Munier [1995]. Figure 2 shows an illustrative example of a simple computation loop with corresponding processing times of operations executed using HSLA arithmetic library (see table in Figure 2(b)).

Data dependencies of this problem can be modeled by a directed graph $G = (\mathcal{T}, \mathcal{E})$. Each task $T_i \in \mathcal{T}$ (node in G) is characterized by the processing time p_i. Edge $e_{ij} \in \mathcal{E}$ 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.

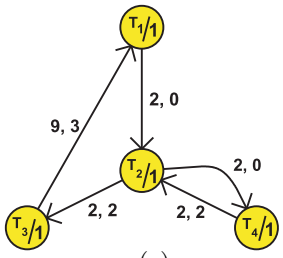
The notions of the length l_{ij} and the processing time p_i are useful when we consider the pipelined processors used in

for k=1 **to** K **do**
T_1: $a(k) = c(k-3) \cdot IN(k)$
T_2: $b(k) = a(k) \cdot OUT(k-2)$
T_3: $c(k) = b(k-2) + 1$
T_4: $OUT(k) = b(k) \cdot 0.5$
end

(a)

Unit	HSLA / FP32 (XCV2000E-6) Max clock frequency: 50 / 93 MHz	
	Processing time [clk]	**Input-output Latency [clk]**
ADD	1 / 1	9 / 11
MUL	1 / 1	2 / 8
DIV	1 / 1	2 / 28
SQRT	1 / 1	2 / 27

(b)



(c)

Fig. 2. (a) Illustrative example of the iterative algorithm. (b) Parameters of HSLA and FP32 arithmetic library (c) The corresponding data dependency graph G.

both arithmetic libraries HSLA and FP32. The processing time p_i represents the time to feed the processor (i.e. new data can be fed to the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of computation (i.e. the input–output latency). Therefore, the result of a computation is available after l_{ij} clock cycles. On the other hand, the *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.

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

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}, \tag{1}$$

where s_i denotes the start time of task T_i in the first iteration. Figure 2(c) shows the data dependence graph of the computation loop shown in Figure 2(a).

The aim of cyclic scheduling Hanen and Munier [1995] is to find a periodic schedule while minimizing the period length w. The scheduling problem is simply solved when the number of processors is not limited, i.e. is sufficiently large. Thereafter, the minimal feasible period w is given by the *critical circuit* c in graph G, maximizing the ratio

$$w = \max_{c \in C(G)} \frac{\sum_{e_{ij} \in c} l_{ij}}{\sum_{e_{ij} \in c} h_{ij}}, \tag{2}$$

where $C(G)$ denotes a set of cycles in G. Critical circuit can be found in polynomial time, $O(n^3 \cdot \log(n))$ Hanen and Munier [1995], and we use this value to determine lower bound of the period length w in our scheduling problem.

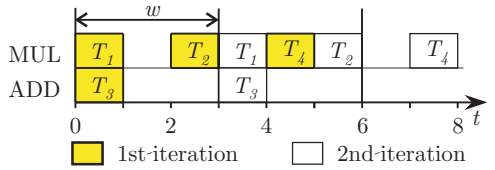


Fig. 3. An optimal schedule.

Two iterations of the optimal schedule to the loop from Figure 2(a) are shown in Figure 3.

3.2 Solution of Cyclic Scheduling on Dedicated Processors by ILP

When the number of processors m is restricted, the cyclic scheduling problem becomes NP–hard Hanen and Munier [1995]. Unfortunately, in our case the number of processors is restricted and the processors are dedicated to execute specific operations (see table in Figure 2). Due to the NP–hardness it is meaningful to formulate the scheduling problem as a problem of Integer Linear Programming (ILP), since various ILP algorithms solve instances of reasonable size in reasonable time.

The aim of this subsection is to outline ILP formulation for simple loops. Fore more details see Šůcha et al. [2004]. This approach is used in next section to schedule nested loops.

The ILP model introduce new variables marked by "ˆ". Let $\hat{s}_i$ be the remainder after division of s_i by w and let $\hat{q}_i$ be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \quad \hat{s}_i \in \langle 0, w-1 \rangle, \quad \hat{q}_i \geq 0. \tag{3}$$

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* (4) corresponding to Inequality (1).

The second kind of restriction are *processor constraints* (5). 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. The constraint uses a 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).

Using ILP formulation we are able to test the schedule feasibility for a given value of w. In addition, we minimize one of the following objective criterions. One of the simplest objectives is to minimize the iteration overlap by the objective function $\min \sum_{i=1}^{n} \hat{q}_i$.

The summarized ILP model, using variables $\hat{s}_i, \hat{q}_i, \hat{x}_{ij}$ is shown in Figure 4. It contains $2n + \sum_{d=1}^{m}(n_d^2 - n_d)/2 + n_e$ integer variables and $n_e + \sum_{d=1}^{m}(n_d^2 - n_d)$ constraints (where n_e is the number of edges in graph G and n_d is the number of tasks on one dedicated processor P_d).

Please notice that w is assumed to be a constant in the ILP model (in order to avoid multiplication of two variables). Optimal period w^*, the shortest period resulting in a feasible schedule, can be found by formulating one ILP model for each w between the lower and upper bound (explained in Šůcha et al. [2004]). Moreover, the interval bisection method can be used, since w^* is not preceded by any feasible solution (i.e. no $w \leq w^* - 1$ results in a feasible solution). Therefore, there are, at maximum, $\log_2(w_{upper} - w_{lower})$ iterative calls of ILP, where w_{lower}

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

subject to

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l_{ij} - w.h_{ij}, \quad \forall e_{ij} \in \mathcal{E} \quad (4)$$

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i \ , \quad \forall i,j : i < j \text{ and } T_i, T_j \in \mathcal{T}_d \quad (5)$$

where

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

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

Fig. 4. ILP model.

and w_{upper} are the lower and upper bound of w, respectively.

4. PARALLELIZATION OF ALGORITHMS WITH MATRIX OPERATIONS

The data dependencies corresponding to LQ control algorithm (see Section 2) represented as *condensed graph* G^c are shown in Figure 5. A single node of the condensed graph G^c 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 presented approach is based on the expansion of G^c to graph G (see Figure 6), where the first level of nesting is modeled by, so called, *united edges* and the second and higher level of nesting (for matrix operations) is modeled by the *processing time fusion* while fully utilizing the corresponding arithmetic unit. Furthermore, G can be scheduled by ILP from Figure 4 while partially fixing the precedence constraints.

4.1 Processing Time Fusion

For example, task T_1 in G^c computes matrix-matrix multiplication $\mathbf{M} = \mathbf{P} \cdot \mathbf{A}$. When the multiplication is evaluated in a common form, i.e. row-wise with respect to matrix $\mathbf{P}$, 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 Heřmánek et al. [2004], where all the elements of the k^{th} column of $\mathbf{P}$ are multiplied by a k^{th} row of $\mathbf{A}$. The partial sums are stored in the memory.

```
for k=1 to 3 do
    for i=1 to 3 do
        for j=1 to 3 do
            M_ij = P_ik · A_kj
        end
    end
end
```

Processing time fusion models second and higher level of nesting (loops over i and j in this example). With respect to implementation in FPGAs (complexity of control logic) the elementary operations of these loops should be processed sequentially without a preemption. Therefore these loops are modeled by one task ($T_{1,1}$ in G in Figure 6) with processing time equal to length of serialized elementary operations.

When the operations on second and higher level of nesting requires different types of processors (arithmetic units) the elementary operations related to one processor are fussed into one task. Then precedence constraints between these tasks are not longer given by the inequality (1), but they are given by the equation $s_j - s_i = l_{ij} - w \cdot h_{ij}$. Such an edge, represented by a dashed line, is called a *fixed edge* in the rest of this text.

First level of nesting is modeled using united edges, explained below.

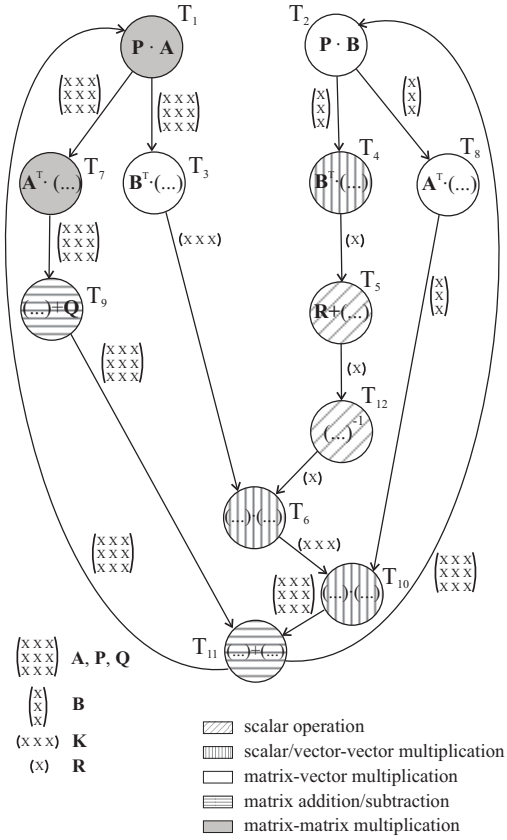


Fig. 5. Data dependencies of the LQ control algorithm represented by the condensed graph G^c.

4.2 United Edges

The first level of nesting is modeled by loop expansion. In order to keep regularity of the loop implementation, the time delay between consequent iterations of the column loop must be the same. Therefore, we use a new kind of edge - *united edges*. For example, when the united edge from $T_{1,1}$ to $T_{1,2}$ belongs to the same group g as the united edge from $T_{1,2}$ to $T_{1,3}$ then $s_{1,2} - s_{1,1} = s_{1,3} - s_{1,2}$. The equivalent precedence constraints are expressed by equalities:

$$s_{1,2} - s_{1,1} - z_g = l_g,$$
$$s_{1,3} - s_{1,2} - z_g = l_g,$$

where z_g is one variable of ILP, such that $z_g \geq 0$, and constant l_g is the length of the united edge. Both z_g and l_g

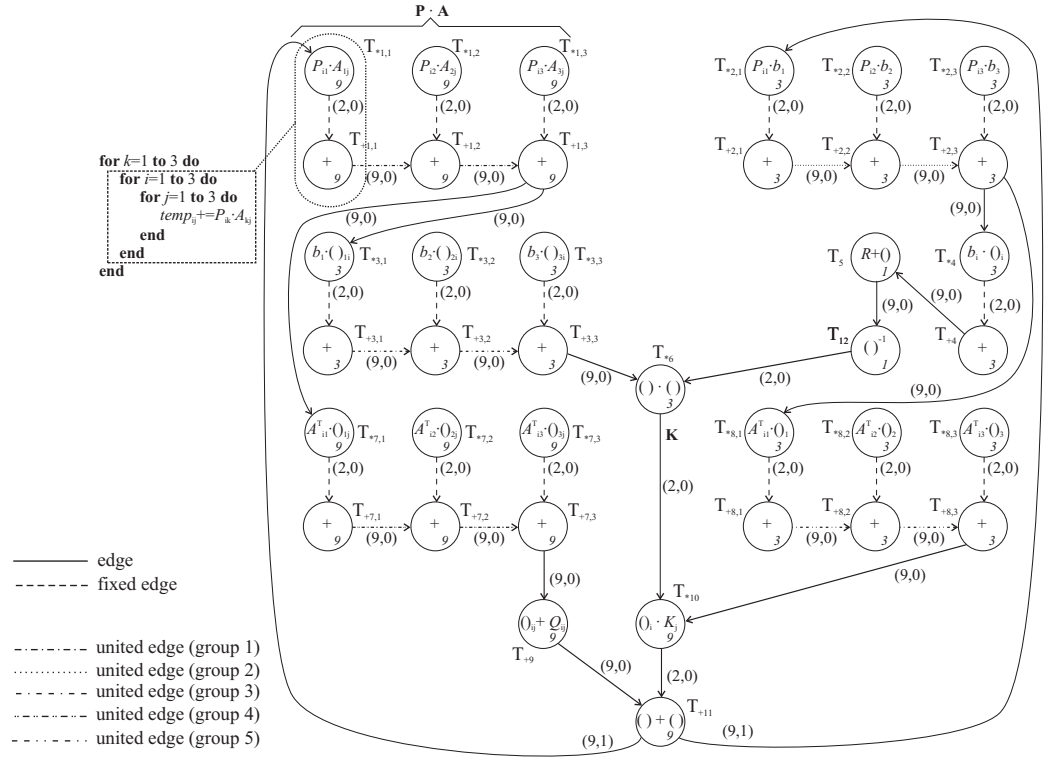


Fig. 6. Abstract model of the LQ algorithm by the graph G with fixed edges and united edges on HSLA library.

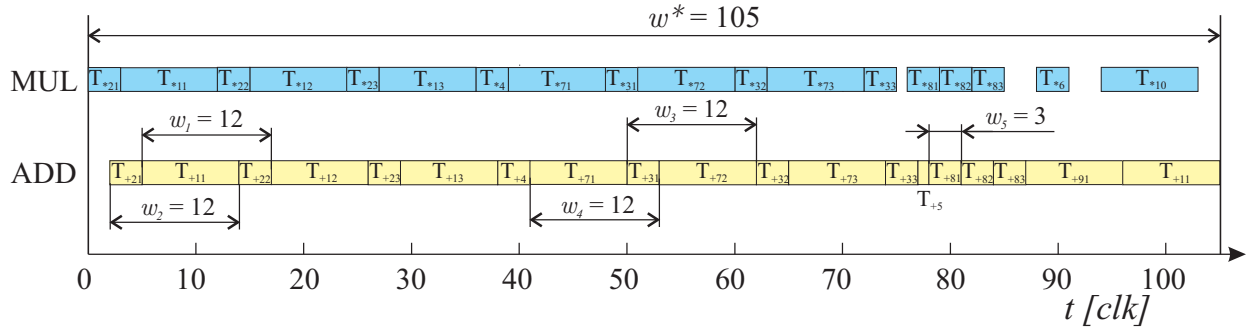


Fig. 7. Resulting schedule of the LQ algorithm ($w^* = 105$; inner periods $w_1 = 12$, $w_2 = 12$, $w_3 = 12$, $w_4 = 12$, $w_5 = 3$) on two dedicated processors (adder and multiplier).

are common for all united edges belonging to the group g. Then $w_g = z_g + l_g$ is a period of the column loop belonging to group g.

4.3 Paralelization of the LQ control algorithm

In the example of LQ control we consider linear SISO (single-input single-output) controlled system of third order. This system can be represented by a three-by-three matrix **A** and a column vector **B** of length three. Then weight matrix **Q** is three-by-three matrix and **R** is a scalar. Kalman gain **K** is a row vector of length three and **P** is three-by-three matrix.

The data dependencies of the LQ control algorithm are shown in Figure 5. Single nodes in the figure represent the set of tasks (performing e.g. scalar-vector multiplication, matrix-matrix multiplication, ...) processed in nested loops. This condensed graph G^c is further expanded by the method shown in two previous subsections, i.e. the first level of nesting is modeled by using *united edges* and second and higher levels are modeled by *processing time fusion*. The resulting *abstract model* of the LQ control algorithm is shown in Figure 6. Finally, the optimal schedule with respect to the abstract model was obtained by ILP from Section 3.2.

5. RESULTS OF SCHEDULING

The resulting schedule assuming HSLA architecture is shown in Figure 7. The presented scheduling technique was tested on an Intel Pentium 4 at 2.4 GHz using the commercial ILP solver CPLEX ILOG Inc. [2005]. The abstract model of the LQ algorithm in Figure 6 consists of 38 tasks on two dedicated processors (19 on the ADD unit, 18 on the MUL unit). DIV unit is not considered as a shared resource since there is only one task T_{12} processed on this unit.

The upper bound of $\hat{q}_i$ was given *a priory* equal to 1 for all tasks. The lower bound of period w, w_{lower} =

103 [*clock cycles*] is given by the maximum of sum of processing time on each processor. The upper bound $w_{upper} = 160$ [*clock cycles*] was calculated from condensed graph G^c (in Figure 5), where all tasks in the inner loops are supposed to be executed simply, in sequence.

The optimal schedule with period $w^* = 105$ [*clock cycles*] was obtained by the interval bisection method in 7 iterative calls of the ILP. The corresponding ILP model contains 366 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 36.1 s. 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 Šůcha et al. [2007] was 21.4 s. In this case, as many as 148 variables were eliminated (the number of eliminated redundant inequalities is twice as big).

HSLA library and FP32 library differ only in input-output latency of used units. The scheduling model stays the same (except constants corresponding to input-output latency) and scheduling results are similar to results with HSLA library.

6. CONCLUSION

The objective of the paper is to demonstrate the use of the scheduling method Šůcha et al. [2007] on a control algorithm. We show, how a scheduling method can be used to optimize the computation speed of real-time iterative algorithms with matrix operations running on FPGA architectures. The method is demonstrated on LQ control algorithm. An advantage of the method is that a suitable hardware architecture can be chosen prior to time consuming implementation.

Important advantage of the scheduling algorithm is its independence of period length. Common approaches based on ILP (e.g. K.-I. Kum [2001]) uses a time indexed binary decision variables x_{it} determining whether a computation of node i starts at time step t. Unfortunately, the size of such an ILP model (corresponding to time complexity) depends on the period length.

REFERENCES

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

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.

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

A. Darte and Guillaume Huard. Loop shifting for loop compaction. *International Journal of Parallel Programming*, 28(5):499–534, 2000. ISSN 0885-7458.

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

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.

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

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.

ILOG Inc. *CPLEX Version 9.1*, 2005. http://www.ilog.com/products/cplex/.

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

R. E. Kalman. Contributions to the theory of optimal control. *Bol. Soc. Mat. Mexicana*, 5(2):102–119, 1960.

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.

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.

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.

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.

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.

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.

P. Šůcha, Z. Hanzálek, A. Heřmánek, and J. Schier. Scheduling of iterative algorithms with matrix operations for efficient FPGA design–implementation of finite interval constant modulus algorithm. *Journal The Journal of VLSI Signal Processing*, 46(1):35–53, January 2007. Springer.

M. J. Wolfe. *High Performance Compilers for Parallel Computing.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0805327304.