Proceedings of the International Conference on Wireless Networks & Embedded Systems (WECON 2009)

Vol. 2of 2

ISBN-13:9788185266657
ISBN-10:81-85266-65-4

[With contribution and editing by Prof. T.L.Singal and Prof. Isha Verma]

Published by: *Mohindra Capital Publishers, 25/7, Industrial Area, Phase-II, Chandigarh-160002. Ph.No. 0172-5077345 on behalf of Chitkara Institute of Engineering and Technology, Chandigarh-Patiala National Highway, Tehsil-Rajpura, Distt. Patiala-140 401(INDIA)*

Implementation of Real Time Scheduling in MATLAB

Mr. Vishal S.Vora- PG Studen, tvsvora@aits.edu.in, Mr. Yagnesh N. Makawana- PG Student, ynmakwana@aits.edu.in
Mr. A. M. Kothari- PG Student, amkothari@aits.edu.in, Prof. A.C.Suthar- Prof. acsuthar@yahoo.co.in
Dept of E&C, CCET, Wadhwan

Abstract- **This paper presents a Matlab based Scheduling toolbox TORSCHE (Time optimization of Resources, Scheduling). The toolbox offers a collection of data structures that allow the user to formalize various off-line and online scheduling problems. Algorithms are simply implemented as Matlab functions with fixed structure allowing users to implement new algorithms. A more complex problem can be formulated as an Integer Linear Programming problem or satisfiability of boolean expression problem. The toolbox is intended mainly as a research tool to handle control and scheduling co-design problems. Therefore, we provide interfaces to a real-time Matlab/Simulik based simulator TrueTime and a code generator allowing generating parallel code for FPGA.**

I. INTRODUCTION

A. Tool Overview

TORSCHE (Time Optimization of Resources, SCHEduling) is a MATLAB-based toolbox including scheduling algorithms that are used for various applications such as high level synthesis of parallel algorithms or response time analysis of applications running under fixed-priority operating system. Using the toolbox, one can obtain an optimal code of computing intensive control applications running on specific hardware architectures. The tool can also be used to investigate application performance prior to its implementation. These values (e.g. the shortest achievable sampling period of the filter implemented on a given set of processors) can be used in the control system design process performed in Matlab/Simulik. The main contribution of the toolbox, which is built on well-known disciplines of the graph theory and operation research, is to make it easy to apply this type of reasoning to a wide range of problems. Many of them are combinatorial optimization problems, and as such they are Challenging from the theoretical point of view.

The toolbox offers a collection of Matlab routines that allow the user to formalize the scheduling problem, while considering appropriate configuration of resources (e.g. Field Programmable Gate Arrays (FPGA) based architecture [1] or micro controllers with real-time operating system [2]), task parameters (e.g. deadlines, release dates, preemption) and optimization criterion (e.g. makespan minimization, maximum lateness minimization, the task completion prior its deadline). The toolbox enables to solve these optimization and decision problems by their reformulation (e.g. to Integer Linear Programming (ILP) or satisfiability of boolean expression problem (SAT)) or to solve them directly while choosing appropriate scheduling algorithm. The input data of the problem instance are typically represented by a set of tasks, set of resources and optimization criterion. The output data of the optimization problems are typically represented by a Gantt chart. The input data might be automatically generated from the problem description (e.g. equations of the filter algorithm) and output data, the schedule, may be used to automatically generat e an implementation of embedded system (e.g. parallel code for dedicated processing units implemented on FPGA).

B. Motivation

The toolbox is intended mainly as a research tool to handle control and scheduling co-design problems. The objective of these problems is to design a set of controllers and schedule them as real-time tasks, such that the overall control performance is optimized for given set of controlled systems and limited computational resources. In some cases, such optimization problem can be formulated analytically. Unfortunately, real applications are more complex and therefore design process cannot be fully automated. In such cases a simulation environment such as Matlab/Simulink presents excellent environment for rapid prototyping of new concepts, simulation and elaboration of design methodologies that are tailored to a specific class of applications and computational resources. For a given control algorithm and computational resources the toolbox makes it possible to derive such real-time parameters as sampling period and jitter. These real-time parameters are further used to derive the control performance (e.g. using TrueTime [3]) and to optimize the controller parameters or to choose another control algorithm and to repeat the design process.

C. Related Work

The toolbox is mostly based on existing well-known scheduling algorithms. In part it contains our previous [4] and current research work [5]. It is very convenient platform to share ideas and tools among researchers and students. Several traditional off-line scheduling algorithms [6] and their extensions represent the basis of the toolbox. In additional, these algorithms can be simply used for scheduling of operations on specific hardware architectures, e.g. FPGAs with arithmetic modules [7]. On-line scheduling algorithms are based on proven approaches from real-time community [8],[9] and on the schedulability analysis for tasks with static and dynamic offsets [10]. In contrast to the MAST tool [11] built to support mainly timing analysis of realtime applications, TORSCHE is not as profound in this area, but covers also off-line scheduling algorithms and due to its implementation in Matlab it is suited to handle control and scheduling co-design problems. TORSCHE is focused on the schedule synthesis and

schedulability analysis. Therefore it is complementary to TrueTime, which is a Matlab/Simulink based simulator.

D. Outline

This paper is organized as follows: Section II presents the tool architecture and basic notation. Section III presents offline scheduling algorithm for the set of tasks with precedence constraints running on parallel identical processors. The problem of makespan minimization is solved via formulation to the satisfiability of boolean expressions problem (SAT). Section IV presents another off-line scheduling problem, cyclic scheduling aiming to find a periodic schedule with a minimum period. The next section describes on-line scheduling problems, that should be solved when the tasks are executed under real-time operating system based on the fixedpriority scheduler. In particular we show the schedulability analysis algorithm for tasks with offsets. All the algorithms are accompanied by illustrative examples including the use of the toolbox functions (for more details see the toolbox manual [12]). Section VI concludes the work.

II. Tool Architecture And Basic Notation

TORSCHE is written in Matlab object oriented programming language and it is used in Matlab environment as a toolbox. Main objects are *Task*, *TaskSet* and *Problem*. Object *Task* is a data structure including all parameters of the task as processing time, release date, deadline etc. These objects can be grouped into a set of tasks with other related information as precedence constraints into a *TaskSet* object.

Object *Problem* is a small structure describing classification of deterministic scheduling problems in Graham and Bła˙zewicz notation [6]. These objects are used as a basis that provide general functionality and make the toolbox easily extensible by other scheduling algorithms. In off-line scheduling problems, the task is given by the following parameters

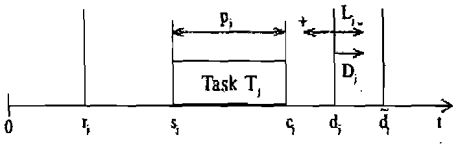

Fig. 1. Task parameters

Processing time, pj , is time necessary for task execution (also called computation time).

Release date, rj , is the moment at which a task becomes ready for execution (also called arrival time, ready time, request time).

Deadline, _ dj , specifies a time limit by which the task has to be completed, otherwise the scheduling is assumed to fail.

Due date, dj , specifies a time limit by which the task should be completed, otherwise the criterion function is charged by penalty.

Weight expresses the priority of the task with respect to other tasks (also called priority).

Processor specifies dedicated processors at which the task must be executed. Resulting schedule is represented by the following parameters:

• *Start time*, sj , is the time when the execution of the task is started.

• *Completion time*, cj , is the time when the execution of the task is finished.

• *Lateness*, Lj = cj − dj .

• *Tardiness*, Dj = max{cj − dj , 0}.

The task is represented by the object data structure with the name *Task* in Matlab. This object is created by the command with the following syntax rule:

```
t1 = task([Name,]ProcTime[,ReleaseTime ...
[,Deadline[,DueDate[,Weight[,Processor]]]]])
```

Command task is a constructor for object of type *Task* whose output is stored into a variable (in the syntax rule above it is variable *t1*). Properties contained inside the square brackets are optional. The object *Problem* is used for classification of deterministic scheduling problems in Graham and Bła˙zewicz notation.

This notation consists of three parts. The first part describes the processor environment, the second part describes the task characteristics of the scheduling problem as the precedence constrains, or the release time. The last part denotes an optimality criterion. An example of its usage is shown in the following code:

```
prob = problem('P|prec|Cmax')
```

Most of all algorithms use the following syntax:

```
tasksetWS = algorithmname(taskset,prob,procesors[,param])
```

Where

• *tasksetWS* is the input *taskset* with an added schedule, • *algorithmname* is the algorithm command name, • *taskset* is the set of tasks to be scheduled, • *prob* is the object of type *problem*,

• *procesors* is the number of processors to be used, • *param* denotes additional parameters, e.g. algorithm strategy etc.

III. Scheduling On Parallel Identical Processors

This section presents the SAT based approach to the scheduling problems. The main idea is to formulate a given scheduling problem in the form of CNF (conjunctive normal form) clauses (for more details see [13]). TORSCHE includes the SAT based algorithm for P|prec|Cmax problem, i.e. scheduling of tasks with precedence constraints on the set of parallel identical processors while minimizing the schedule makespan. is a function of Boolean variables in the form xijk. If task Ti is started at time unit j on the processor k then xijk = true, otherwise xijk = false. For each task Ti, where i = 1 . . .n, there are S ×R Boolean variables, where S denotes the maximum number of time units and R denotes the total number of processors.

The Boolean variables are constrained by the three following rules (modest adaptation of [14]): **1.** For each task, exactly one of the S×R variables has to be equal to 1. Therefore two clauses are generated for each task Ti. The first

guarantees having at most one variable equal to 1 (true):(¯xi11 □ ¯xi21)□···□(¯xi11 □ ¯xiSR)□···□(¯xi(S−1)R □ ¯xiSR).

The second guarantees having at least one variable equal to 1: (¯xi11 □ ¯xi21 □ ··· □ ¯xi(S−1)R □ ¯xiSR). **2.** If there is a precedence constrains such that Tu is the predecessor of Tv, then Tv cannot start before the execution of Tu is finished. Therefore, xujk → ((¯xv1l □ ··· □ ¯xvjl □ ¯xv(j+1)l □ ··· □ ¯xv(j+pu−1)l) for all possible combinations of processors k and l, where pu denotes the processing time of task Tu. 3. At any time unit, there is at most one task executed on a given processor. For the couple of tasks with a precedence constrain this rule is ensured already by the clauses in the rule number 2. Otherwise the set of clauses is generated for each processor k and each time unit j for all couples Tu, Tv without precedence constrains in the following form: (xujk → ¯xvjk) □ (xujk → ¯xv(j+1)k) □ ··· □ (xujk → ¯xv(j+pu−1)k).

In the toolbox we use a *zChaff* [15] solver to decide whether the set of clauses is satisfiable. If it is, the schedule within S time units is feasible. An optimal schedule is found in iterative manner. First, the List Scheduling algorithm is used to find initial value of S. Then we iteratively decrement value of S by one and test feasibility of the solution. The iterative algorithm finishes when the solution is not feasible.

As an example we show a computation loop of a Jaumann wave digital filter [16]. Our goal is to minimize computation time of the filter loop, shown as directed acyclic graph in Fig. 2. Node in the graph represent the tasks and the edges represent precedence constrains. The nodes are labeled by the operation type and processing time pi. We look for an optimal schedule on two parallel identical processors.

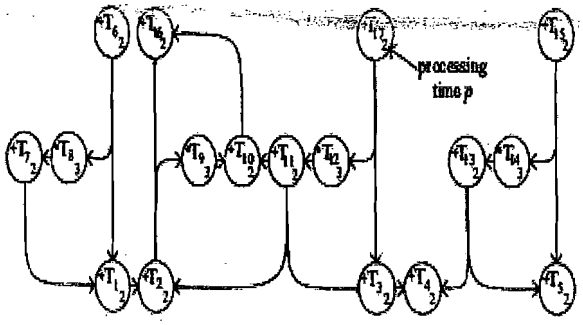

Fig. 2. Jaumann wave digital filter

Fig. 3 shows consecutive steps performed within the toolbox. First, we define the set of task with precedence Finally we plot the Gantt chart.

```
>> procTime = [2,2,2,2,2,2,2,3,3,2,2,3,2,3,2,2,2];
>> prec = sparse(...
[6,7,1,11,11,17,3,13,13,15,8,6,2,9 ,11,12,17,14,15,2
,10],...[1,1,2,2 ,3 ,3 ,4,4 ,5 ,5 ,7,8,9,10,10,11,12,13,14,16,16],...
[1,1,1,1 ,1 ,1 ,1,1 ,1 ,1 ,1,1,1,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1],...
17,17);
>> jaumann = taskset(procTime,prec);
>> jaumannSchedule =
satsch(jaumann,problem('P|prec|Cmax'),2)
Set of 17 tasks
There are precedence constraints
There is schedule: SAT solver
SUM solving time: 0.06s
MAX solving time: 0.04s
Number of iterations: 2
>> plot(jaumannSchedule)
```

The *satsch* algorithm performed two iterations. In the first iteration 3633 clauses with 180 variables were solved as satisfiable for S = 19 time units. In the second iteration 2610 clauses with 146 variables were solved with unsatisfiable result for S = 18 time units. The optimal schedule is depicted in Fig. 4.

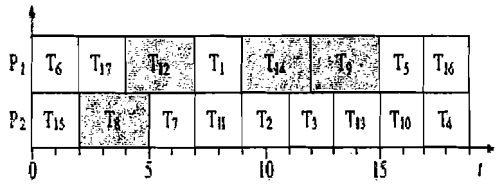

Fig. 4. The optimal schedule of Jaumann filter.

IV. CYCLIC SCHEDULING ON DEDICATED PROCESSORS

Cyclic scheduling deals with a set of operations (generic tasks) that have to be performed an infinite number of times [17]. This approach is also applicable if the number of loop repetitions is large enough. If execution of operations belonging to different iterations can interleave, the schedule is called *overlapped*. An overlapped schedule can be more effective especially if processors are pipelined hardware units or precedence delays are considered. The *periodic schedule* is a schedule of one iteration that is repeated with a fixed time interval called a *period* (also called *initiation interval*). The aim is then to find a periodic schedule with a minimum period [17]. As an example, we show a computation loop of a wave digital filter (WDF) [18] consisting of eight tasks. It is extended to five channels by assuming five clock cycles processing time of each task (i.e. single channels are shifted by one clock cycle). Fig. 5 shows the filter with corresponding processing times of operations executed using HSLA The scheduling problem is to find a start time si(k) of every occurrence Ti [17]. arithmetic library on FPGA (input-output latency of ADD (MUL) unit is 9 (2) clock cycles, respectively [7]). Operations in a computation loop can be considered as a set of n *generic tasks* T = {T1, T2, ..., Tn} to be performed K times where K is usually very large.

One execution of

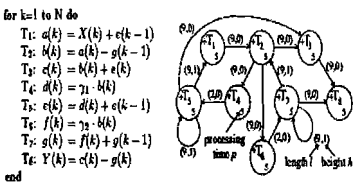

Fig. 5. (a) An example of a computation loop of wave digital filter (WDF). (b) Corresponding data dependency graph G of WDF.

Data dependencies of this problem can be modeled by a directed graph G. Each task (node in G) is characterized by the processing time pi. Edge eij from the node i to j is weighted by a couple of integer constants lij and hij . Length lij represents the minimal distance in clock cycles from the start time of the task Ti to the start time of Tj and is always greater than zero (corresponds to input-output latency in our example). On the other hand, the height hij specifies the shift of the iteration index (dependence distance) related to the data produced by Ti and read (consumed) by Tj. Assuming a *periodic schedule* with the *period* w (i.e. the constant repetition time of each task), each edge eij in graph G represents one precedence relation constraint $s_j - s_i \geq l_{ij} - w \cdot h_{ij}$, (1) where si denotes the start time of task Ti in the first iteration. Fig. 5(a) shows the data dependence graph of the computation loop shown in Fig. 5(b). When the number of processors m is restricted, the cyclic scheduling problem becomes NP–complete [17]. Unfortunately, in our case the number of processors is restricted and the processors are dedicated to execute specific operations.

In the toolbox we formulate the scheduling problem as a problem of Integer Linear Programming (ILP). For more detail about the scheduling algorithm see [12], [4]. The schedule of the WDF example, obtained as outlined in Fig. 7, is shown in Fig. 6. The toolbox is interconne cted with Matlab/Simulink based simulator TrueTime [3] which facilitates cosimulation of realtime task execution and continuous plant dynamics. Arbitrary schedule can be directly transformed to a model used by TrueTime. This function allows to design complex control systems and simulate influence of external events on the schedule. Furthermore, the scheduling results can be used to generate parallel code in Handel C [19] for FPGA. It allows to design time critical algorithms especially for FPGA as is shown in WDF example in this section. The toolbox provides to designer full control over the scheduling algorithm.

```
>> load wdf
>> UnitProcTime=[5 5];
>> UnitLattency=[9 2];
>> G=cdfg2LHgraph(wdf,UnitProcTime,UnitLattency);
>> t=taskset(G);
>> prob=problem('m-DEDICATED');
>> schoptions=schoptionsset('ilpSolver','glpk', ...
'cycSchMethod','integer','varElim',1);
>> taskset_sch=mdcycsch(t, prob, 1, schoptions)
Set of 8 tasks
There are precedence constraints
There is schedule: MONOCYCSCH - ILP based algorithm
Tasks period: 31
Solving time: 0.094s
Number of iterations: 5
>> plot(taskset_sch)
```

Fig. 7. Solution of a cyclic scheduling problem in the toolbox.

V. REAL-TIME SCHEDULABILITY ANALYSIS

Real-Time scheduling is usually used in Real-Time operating systems for scheduling a set of periodic tasks. Simple scheduling algorithms such as *fixed-priority scheduling* are usually used since they need to be executed on-line. Given a system comprising of a set of real-time tasks and a scheduling algorithm, a verification algorithm can determine whether all the tasks in the system meet their real-time constraints deadlines). Besides basic *response-time analysis* for rate monotonic algorithm [9], TORSCHE contains a more advanced technique: *schedulability analysis for tasks with offsets*. This technique was firstly introduced by Tindell in [20], and later further formalized and enhanced by Palencia and Harbour in [10]. In both papers, authors designed exact algorithm for this NP-hard problem (determining response times of tasks in the system) as well as polynomial approximate analysis that finds upper bound to the task response times. Currently, TORSCHE contains only the exact algorithm. Note: This section uses notation different from the rest of this paper. The reason is that this notation is common in real-time community.

A. Computational Model

The real-time system considered for analysis is composed of tasks executing in the same processor, but the analysis can be easily extended for multiprocessor systems. Tasks are grouped to *transactions*. Each transaction Γi is activated by a periodic sequence of external events with period Ti. The relative phasing between the different external events is arbitrary. Each task will be identified with two subscripts: the first one identifies the transaction to which it belongs and the second one the position that the task occupies within the tasks in its transactions, when they are ordered by increasing offsets. Task τij is activated (released) when a relative time—called the *offset*, Φij—elapses after the arrival of the external event. The offset can be static or dynamic. Dynamic offsets are represented by a value of jitter Jij , which specifies the length of the interval in which a task can be activated. Cij is the worst-case execution time. It is assumed that each task has its unique priority and all the tasks are scheduled using a preemptive fixed priority scheduler. If tasks synchronize for using shared resources in a mutually exclusive way they will be using a hard real-time synchronization protocol such as the priority ceiling protocol [9]. The *response time* of each task τij is defined as a difference between its completion time and the instant at which the associated external event arrived. The worst-case response time is denoted by Rij . Each task may have associated global deadline Dij , which is also relative to the arrival of external event. A system is said to be schedulable if for each task $R_{ij} \leq D_{ij}$.

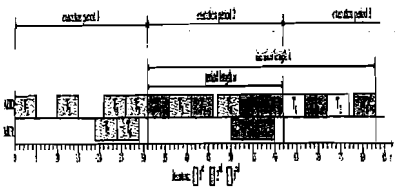

Fig. 8 shows an example of a real-time system. The horizontal axis represents time. Down-pointing arrows represent periodic external events, filled boxes represent task execution. Dashed lines below each transaction axis represent task jitter values. The system in this figure is composed of three transactions.

The first and the third ones group two tasks, the second one groups three tasks. No blocking is considered in this example and tasks have assigned priority corresponding to their subscript. The smaller the number the higher the priority. Note that task τ12 has its offset greater than completion time of the previous task in the transaction and that tasks not initiating the transactions has non-zero jitter (dotted line). In this way, offsets and jitters can be used to represent self-suspending tasks (e.g. tasks calling sleep() OS service or waiting for data from a periphery). *B. Exact Response-Time Analysis Algorithm* The calculation of the exact worst-case response-time (WCRT) for any task in the system is NP-hard problem [21] and therefore the complexity of the exact algorithm is exponential to the size of the system. In [10] and [22] approximate methods that calculates an upper bound to the WCRT are developed. These polynomial time algorithms are based on simplification of the exact algorithm. TORSCHE currently implements only the exact algorithm.

To find the worst-case response time of a task under analysis (τab), it is necessary to build the worst-case scenario for this task. Finding this scenario rests in finding such a combination of higher priority tasks having the highest contribution to τab response time. The time when this combination occurs is called the *critical instant* of task τab. In the case where all tasks are independent (without offsets), the critical instant occur at the time when all higher priority tasks are activated simultaneously with τab. This no longer holds for tasks with offsets, as it might be impossible for some sets of task to be activated at the same time.

The complexity of the algorithm rests in the fact that we have to explore all possible combinations of tasks from each transaction and find which combination produces the worstcase response time. Fig. 9 shows the scenario of the system from Fig. 8, for which the response-time of task τ21 reaches its worstcase value. This situation happen when tasks τ00, τ12 and τ21 are activated simultaneously after being delayed by their maximum jitter. The phasing of external events as well as activation times of all tasks in this scenario are depicted in the three top-level axes of the picture. The last axis shows the schedule as produced by the fixed-priority scheduler. The critical instant is at time 0 of this schedule and the length of busy period Lab = 150. For this scenario, (the worst-case) response-time R21 = L21 +Φ21 + J21 = 200. C. *Matlab Session Example* The model of a real-time system is entered into Matlab by creating appropriate *Task* and *Transaction* object instances. The response-time analysis can be performed by calling taskoffsanal function. Fig. 10 shows the steps needed to perform response-time analysis within TORSCHE. Computed response times are listed as respTime values. Other toolbox functions can be used to automatically draw figures like the ones in this section.

```
>> t00=task(10, 0);
>> t01=task(10,[10 15], 100);
>> G0=transaction([t00 t01], 100);
>> t10=task(25,0);
>> t11=task(10, [25 30]);
>> t12=task(20, [70 80], 100);
>> G1=transaction([t10 t11 t12], 130);
>> t20=task(30, 0);
>> t21=task(35, [30 50], 250);
>> G2=transaction([t20 t21], 300);
>> system = [G0 G1 G2]; setprio(system, 'rm');
>> taskoffsanal(system)
System:
Transaction: per=100.0
Task t00: prio=7 wcet=10.0 o=0 j=0 respTime=10.0
Task t01: prio=6 wcet=10.0 o=10.0 j=5.0 respTime=25.0
Transaction: per=130.0
Task t10: prio=5 wcet=25.0 o=0 j=0 respTime=45.0
Task t11: prio=4 wcet=10.0 o=25.0 j=5.0 respTime=60.0
Task t12: prio=3 wcet=20.0 o=70.0 j=10.0 respTime=120.0
Transaction: per=300.0
Task t20: prio=2 wcet=30.0 o=0 j=0 respTime=145.0
Task t21: prio=1 wcet=35.0 o=30.0 j=20.0 respTime=200.0
```

Fig. 10. Offset-based response-time analysis in a Matlab session

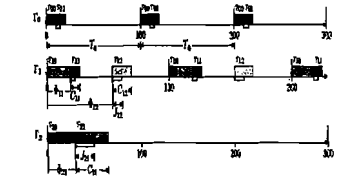

Fig. 8. Graphical representation of the real-time system.

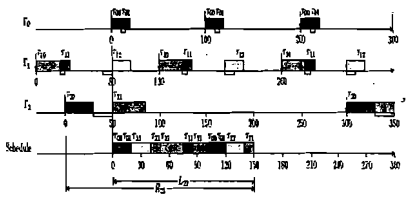

Fig. 9. The schedule that produces the worst-case response time for τ_{21}.

VI. Conclusions And Future Work

This paper presents TORSCHE Scheduling Toolbox for Matlab for off-line and on-line scheduling. The toolbox includes scheduling algorithms, that are used for various applications as high level synthesis of parallel

algorithms or response time analysis of applications running under fixed priority operating system. The main objective of this project is to facilitate design of real-time applications mainly in control domain where the Matlab is frequently used. In this paper, we have shown the applicability on three examples. In the future work we will focus on interconnections to another designs tools and simulators and we will incorporate new scheduling algorithms. Actual version of the toolbox is freely available.

VII. REFERENCES

[1]. J. Kadlec, *FloatPipe1v35 Modules for Virtex and Virtex 2*, February 2004. http://www.celoxica.com.

[2]. L. Waszniowski and Z. Hanz´alek, "Analysis of OSEK/VDX based automotive applications," in *IFAC Symposium on Advances in Automotic Control, Salerno*, Elsevier, April 2004.

[3]. M. Andersson, D. Henriksson, and A. Cervin, *TrueTime 1.3Reference Manual*. Lund University, Sweden, 2005. http://www.control.lth.se/□dan/truetime/.

[4]. P. ˇS°ucha, Z. Pohl, and Z. Hanz´alek, "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.

[5]. A. He˘rm´anek, J. Schier, P. ˇS°ucha, and Z. Hanz´alek, "Optimization of finite interval CMA implementation for FPGA," in *In IEEE 2005 Workshop on Signal Processing Systems (SIPS'05)*, pp. 75–80, Piscataway:IEEE, 2005.

[6]. J. Bła˙zewicz, K. Ecker, G. Schmidt, and J. We¸glarz, *Scheduling Computer and Manufacturing Processes*. Springer, second ed., 2001.

[7]. R. Matou˘sek, M. Tich´y, A. Z. Pohl, J. Kadlec, and C. Softley, "Logarithmic number system and floating-point arithmetics on FPGA." Field-Programable Logic and Applications: Reconfigurable computing Is Going Mainstream. Lecture notes in Computer Science A 2438, Springer, Berlin, 2002.

[8]. G. C. Buttazzo, *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, second ed., 2005.

[9]. J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ, USA: Prentice Hall, 2000.

[10]. J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets.," in *Proceedings of the 19th Real Time Systems Symposium*, pp. 26–37, IEEE Computer Society Press, December 1998.

[11]. "MAST (Modeling and Analysis Suite for Real-Time Applications)."http://mast.unican.es/.

[12]. M. Kutil, P. ˇS°ucha, M. Sojka, and Z. Hanz´alek, *TORSCHE:Scheduling Toolbox Manual*, February 2006. http://rtime.felk.cvut. cz/scheduling-toolbox/.

[13]. Y. Crama and P. L. Hammer, "Boolean functions: Theory, algorithms and applications," 2006. ttp://www.rogp.hec.ulg.ac.be/ Crama/Publications/BookPage.html.

[14]. S. O. Memik and F. Fallah, "Accelerated SAT-based Scheduling of Control/Data Flow Graphs.," in *ICCD*, pp. 395–400, IEEE Computer Society, 2002.

[15]. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[16]. S. H. de Groot, S. Gerez, and O. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 39, pp. 351–364, 1992.

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