Optimisation of Applications for FPGAs with PowerPC Processor Using Priced Timed Automata

Jan Krákora and Zdeněk Hanzálek
Czech Technical University in Prague,
Faculty of Electrical Engineering, Department of Control Engineering
Karlovo náměstí 13, Prague 2, 121 35, Czech Republic
Email: {krakorj, hanzalek}@fel.cvut.cz

***Abstract*— Some digital signal processing applications can be executed faster by moving parts of application implementation into hardware. Platforms, like Xilinx Virtex-4 4VFX12, allow a user to run software in embedded processor and offload computations to the set of hardware modules. The article deals with optimal schedule synthesis techniques for tasks executed on such platform using Priced Timed Automata and UPPAAL CORA tool. It shows a schedule synthesis techniques minimising makespan or sum of completion times criterion. Moreover, it presents a synthesis methodology considering a fraction of resource capacity, called resource budget and maximization of processor utilization for tasks with bounded period. Case studies and FPGA experiments are finally presented.**

I. Introduction

Some software applications can be executed faster by moving of application implementation into hardware. There are some platforms, like Xilinx Virtex-4 4VFX12 [16], that allow user to run software in embedded processor and offload computations to the set of hardware modules in user defined coprocessors. These platforms are typically used for digital signal processing (DSP) applications, were complex computation is required [7].

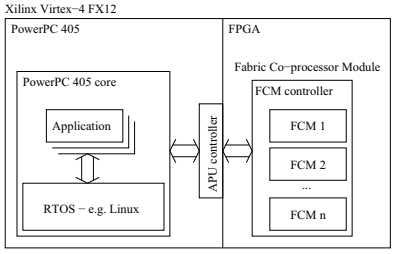


Fig. 1. An application under consideration

Let us suppose the software application running on the embedded processor, that shares a set of co–processors implemented in the hardware. The problem is how to effectively implement the set of tasks in that generic resource.

The article deals with schedule synthesis techniques that find an optimal task assignment on the Field-Programmable Gate Array (FPGA) platform using timed automata. We show a schedule synthesis techniques minimising schedule length Λ_{max}, called makespan, and sum of completion times $\sum \Lambda_j$ criterion (denoted C_{max} and $\sum C_j$ in [6] notation). Moreover, we present a synthesis methodology to guarantee the resource budget [11], [10] and a methodology to maximize the processor utilization for tasks with bounded period.

The scheduling problems considered in the article are characterised by set $\mathcal{S} = \{t_1, t_2, ...t_n\}$ of n tasks with precedence constraints, a set of dedicated processors $\mathcal{P}$. The computation time of task t_j is C_{t_j}. The study proposes both sporadic and periodic task scheduling synthesis.

For the schedule synthesis techniques UPPAAL-CORA tool is used [5]. It implements a branch and bound algorithm [6] for cost–optimal reachability analysis using Linearly Priced Timed Automata (LPTA). LPTA is a priced transition system and it is an extension of timed automata (TA) to optimal scheduling and planing problems.

The article is structured as follows. Subsection I-A surveys the related work. The proposed schedule synthesis techniques are presented in Section II. Schedule synthesis that minimise makespan criterion is described in Subsection II-A, minimisation of the sum of completion times is shown in Subsection II-B. How to find the optimal schedule of periodic tasks, sharing coprocessor resources, with respect to a processor budget is given in Subsection II-C. Finally, the maximization of processor utilization for tasks with bounded period is shown in Subsection II-D. An experiment is shown in Section III.

A. Related work

Authors in [12] show a methodology for time optimal production scheme generation using timed automata and OpenKronos [14]. The optimal production schemes are generated using the algorithms for reachability analysis of timed automata implemented in OpenKronos. In compare to our approach the application paper does not propose periodic task scheduling synthesis.

Job-shop scheduling problem can be modelled as a special class of acyclic timed automata, see [1]. Finding an optimal schedule corresponds to the search of the shortest (in term of elapsed time) path in the timed automaton state space. The approach of the finding of the shortest paths in timed automata is similar to the approach presented in our article.

This article is supported by Research Programme of the Ministry of Education of the Czech Republic No. MSM6840770038: Decision Making and Control for Manufacturing III and by the European Commission under project FRESCOR IST-034026.

Nevertheless, our case is not job–shop problem, since some of the application tasks are processed in more than one processor simultaneously.

Similar comparison can be provided in the case of the non-preemptive scheduling problem for job-shop scheduling, solved in [8], [9]. These techniques specify pre-defined goal locations of an automaton and use reachability analysis to construct traces leading to the goal locations. The traces are used as schedules.

TIMES tool [4] based on reachability analysis of timed automata supports the following scheduling policies: rate monotonic, deadline monotonic, fixed-priority scheduling (with user defined priorities), earliest deadline first (EDF), and first come first served (FCFS). All scheduling policies support preemptive or non-preemptive task sets. In our case we generate an off–line schedule.

A scheduling framework integrating specification and schedule generation for real–time systems is presented in [3]. It proposes cyclic or sporadic task scheduling analysis using timed Petri nets and timed automata. It includes precedence constraints and resource sharing. Using the framework, an on–line non–preemptive schedule can be generated. Our approach is the timed automata based only, we are able to produce off-line scheduler.

From the budget reservation point of view of, [2] proposes a Constant Bandwidth Server (CBS). It is an on-line scheduling methodology reserving a fraction of the processor bandwidth to each task. It integrates hard and soft real-time computing in a single system, under the EDF scheduling algorithm. Method to resource budget bound in rate monotonic scheduling environment has been proposed in [10].

B. Contribution

The article contributions are the following: (a) Implementation of schedule synthesis techniques based on timed automata into FPGA platform, (b) approach to maximization of processor utilization for tasks with bounded period and (c) schedule synthesis technique of periodic tasks, sharing a co-processor resources, with respect to a processor budget.

II. Schedule synthesis techniques

This section presents a set of scheduling techniques, based on timed automata, for a processor-based system presented above. For better understanding of the problem under consideration the target platform and the software configuration should be explained.

As presented above we are interested to the platforms that combine embedded processor and FPGA. The Xilinx Virtex-4 4VFX12 platform, see Figure I, provides one PowerPC 405 32-bit RISC processor core [15] and FPGA in single device both interconnected via special interfaces as Auxiliary Processor Unit interface (APU) [15].

The APU enables tight integration between an application-specific function and the processor pipeline. It allows a designer to extend native PowerPC 405 instruction set with a custom instruction called Fabric Co–processor Module (FCM).

The FCM interacts with PowerPC through the APU controller using a set of PowerPC-FCM instructions. APU controller decodes those instructions, it determines what resources are needed and passes required information from PowerPC to FCM and vice versa. For example, the APU controller determines if an instruction is a read, a write, if it needs source data from General Purpose Register, etc.

The PowerPC instruction set is extended. There are three different instruction classes defined by their interaction with the normal processor pipeline execution: *Autonomous*, *Non–Autonomous Blocking* and *Non–Autonomous Non–blocking* instruction class. In our article the *Non–Autonomous Blocking* instruction class si supposed. This class of instructions stall the pipeline until the PowerPC-FCM instruction is done. For example an instruction that takes data from the FCM and writes it to a PowerPC register. Such instruction that cannot be predictably aborted and later re-issued must be blocking.

Our system configuration consists of PowerPC running a multi–task application sharing a set of FCM in FPGA, e.g. see Figure 2.

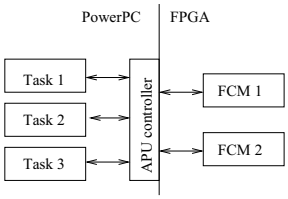


Fig. 2. Problem structure

The problem under consideration is to find the optimal schedule of a given task set interacting with a set of co–processor modules. The optimal schedule result is derived from the best cost diagnostic trace given by timed automata verification using UPPAAL CORA tool.

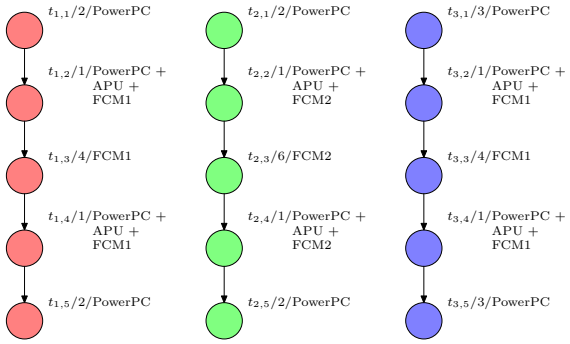


Fig. 3. Set of 3 tasks executed on dedicated processors

A. Minimisation of makespan

The subsection shows how to find the optimal schedule with respect to makespan criterion. Let us have a processor running multi–task application consisting of 3 non–periodic tasks. Each task is sharing two FCMs via an APU controller, see Figure 2.

Set of 3 tasks executed on dedicated processors is shown in graph in Figure 3.

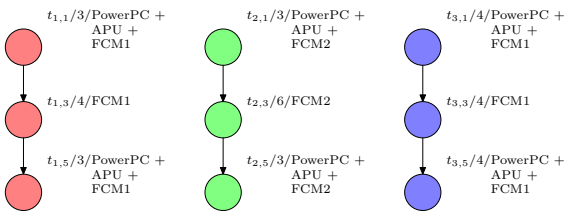


Fig. 4. Abstracted set of 3 tasks executed on dedicated processors

Execution of each task, identified by an id, is divided into five parts. Each part notation is $part\ label/C_{t_{id}}/processor$. At the beginning, the data to the FCM are pre–processed in PowerPC, represented by part labelled $t_{\{id\},1}$. Further the processor interacts with the FCM via APU controller, the FCM decodes the instruction and the data are loaded into FCM, part label is $t_{\{id\},2}$. When the APU controller operation is finished the FCM function is executed, $t_{\{id\},3}$ label. When the FCM function is finished, the result data are prepared to be stored back to the processor application via APU, $t_{\{id\},4}$ label. The resulting data are post–processed in processor, label $t_{\{id\},5}$.

Due to Non–Autonomous Blocking instruction, used for an interaction with FCM, the set of task can be abstracted, see Figure 4. Since the FCM instruction can not be aborted by any processor activity, the APU operation $t_{\{id\},2}$ and $t_{\{id\},4}$ are integrated in part $t_{\{id\},1}$ and $t_{\{id\},5}$ respectively. For that, the possible APU model is neglected in the system.

Timed automaton of the *processor*, the *task*, and the *FCM* are presented in Figure 5 a), b) and d). Due to the makespan criterion the cost is computed separately in *Schedule detector* timed automaton (Figure 5 c)).

To find the optimal schedule with makespan criterion, the CTL* [5] property `E<> SD.ScheduleFound` should be satisfied. The optimal schedule can be derived from the best cost diagnostic trace. The optimal schedule is depicted in Figure 6. The best cost is equal to 22 time units so the makespan is equal to 22.

The proposed result can be directly implemented into PowerPC using e.g ANSII-C or PowerPC assembler. The accordant *dispatcher task* will schedule each application task as presented in experiment section III.

ILP approach: The problem studied in this subsection, minimization of Λ_{max}, can be also conveniently formulated as Integer Linear Programming problem using the approach adopted in [13]. This approach is based on so called scheduling of multiprocessor tasks with general precedence relations.

	Timed automata [sec]	ILP by GLPK solver [sec]
3 tasks	0.2	0.1
12 tasks	120	unsolved

TABLE I

ILP AND TA APPROACH PERFORMANCE COMPARISON OF Λ_{max}

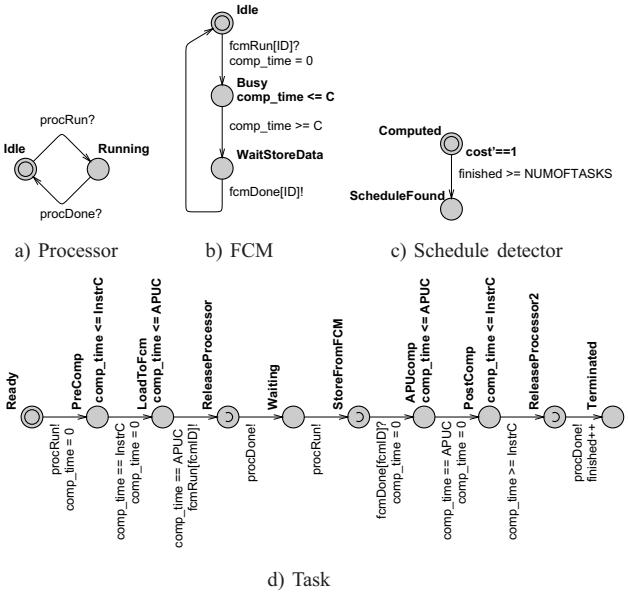


Fig. 5. Λ_{max} problem, timed automata

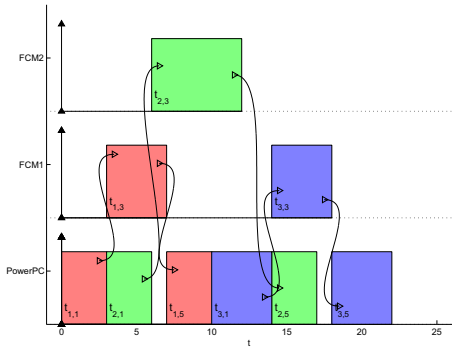


Fig. 6. Optimal schedule for taskset on Figure 4

The optimal schedule is same as presented in Figure 6, nevertheless Table I compares performance of both approaches achieving the same objective criterion. As presented, for 12 tasks the ILP is not able to reach the optimal solution.

Fig. 7. Sum of completion times problem, *task* timed automaton

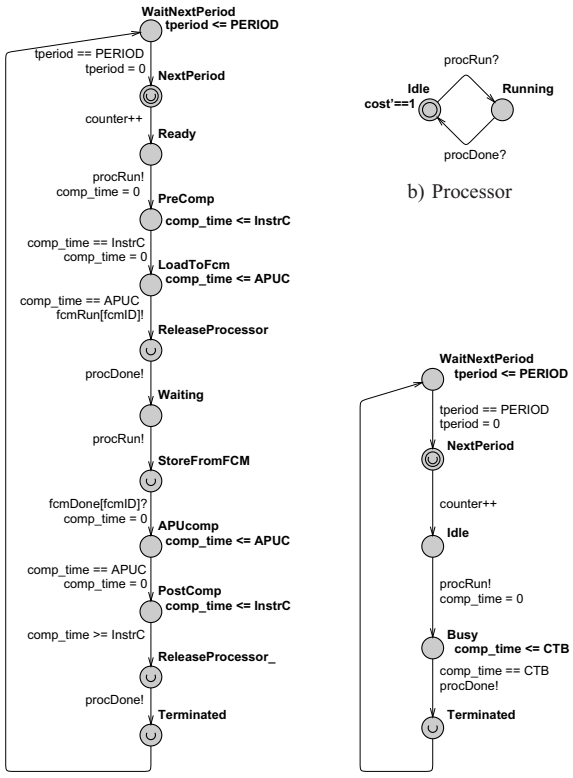
B. Minimisation of sum of completion times

This subsection presents, how to find the optimal schedule with respect to the sum of completion times criterion.

The demonstrated configuration is similar to the one presented for makespan synthesis. The dedicated task set is depicted in Figure 4. Timed automaton of the *processor* and the *FCM* have been presented in Figure 5 a) and b).

Due to the sum of completion times criterion the cost is not computed using *Schedule detector* automaton, but in every *task* location where time behaviour is modelled, except the location Terminated (task is finished there). For that, the *task* automaton has to be modified, see Figure 7.

To find the optimal schedule with sum of completion times criterion, the property `E<> t1.Terminated && t2.Terminated && ... && tn.Terminated` should be satisfied. The optimal schedule can be derived from the best cost diagnostic trace. The optimal schedule is the same to the schedule depicted in Figure 6. The best cost of sum of completion times is 22.



a) Task b) Processor c) Required budget task

Fig. 8. Budget guarantee timed automata

C. Guarantee of budget available on PowerPC

This subsection presents a technique how to find the optimal schedule of periodic tasks, sharing the FCM resources, with respect to PowerPC processor budget.

First we will describe, how to model periodic tasks and then, the scheduling synthesis follows.

Periodic task modelling: An example of the periodic task is presented in Figure 8 a). The period is measured by clock variable `tperiod` and by constant `PERIOD`, see locations WaitNextPeriod and NextPeriod. The periodical task behaviour is given by the transition Terminated → WaitNextPeriod.

Schedule synthesis: Due to periodic behaviour of each task it is supposed that the schedule is the same for every *macro period* (T_{macro}). The T_{macro} is given by interval from the time when all tasks are in their location NextPeriod to the next closest time when all tasks are again in the same location.

To identify an existence of a T_{macro}, the CTL property `E<> t1.NextPeriod && ... && tn.NextPeriod && global_timer > 0` should be satisfied. (`global_timer` is a clock variable used as global timer). The T_{macro} is given as $T_{macro} = T_{t_{\{id\}}} * counter_{t_{\{id\}}}$ where id is a task identifier, the $T_{t_{\{id\}}}$ is task period, given by `PERIOD` constant, and the $counter_{t_{\{id\}}}$ is value of task cycles, derived from the TA `counter` variable.

Case study: The demonstration configuration consists of a processor running two tasks t_1, t_2 each using its FCM. The periods are $T_{t_1} = 14$ and $T_{t_2} = 56$. The required budget is given by its period $T_{t_b} = 7$ and the computation time $C_{t_b} = 2$. The set of tasks is shown in Figure 9.

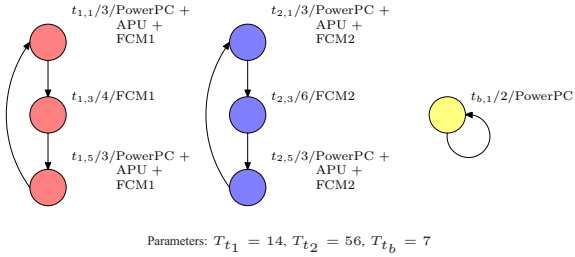


Fig. 9. Set of 3 tasks executed on dedicated processors, including required budget task t_b

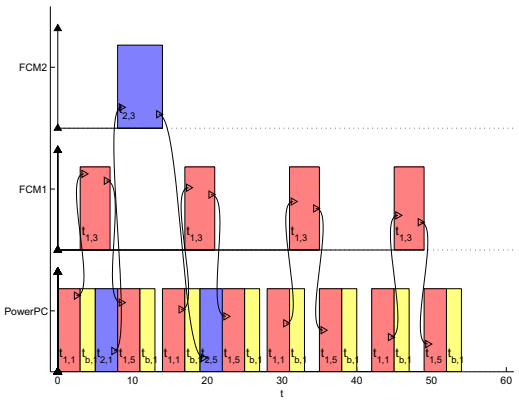


Fig. 10. Optimal schedule for taskset on Figure 9

The required budget is modelled as a periodic task t_b, called *required budget task*, that occupies the processor for computation time `comp_time` equal to constant `CTB`. The T_{t_b} is equal to constant `PERIOD`. The *task* model and the *budget*

model are presented in Figure 8 a) and c). The *processor* automaton is shown in Figure 8 b). The `cost` variable is used to define the processor utilization.

To find the optimal schedule, the property `E<> t1.NextPeriod && t2.NextPeriod && ReqBudget.NextPeriod && t2.counter==1` should be satisfied. The optimal schedule can be derived from the best cost diagnostic trace. The optimal schedule is depicted in Figure 10. Also this schedule can by directly implemented to PowerPC.

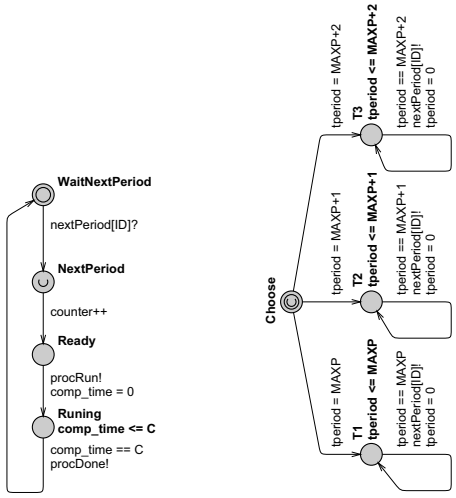
D. Maximization of processor utilization for tasks with bounded period

This subsection presents a technique that finds the optimal schedule for periodic task set, sharing one processor, where period of each task is given by lower and upper bound. The aim is to find each task period so the schedule is feasible and to maximize the shared processor utilization, within the meaning of $\sum \frac{C}{T}$.

Case study: Let us suppose a system consisting of set of periodic tasks t_1, t_2 and t_3. Tasks computation times are $C_{t_1} = 4$, $C_{t_2} = 3$ and $C_{t_3} = 3$. Admissible tasks integer value of periods are defined as follows $T_{t_1} = \langle 5; 7 \rangle$, $T_{t_2} = \langle 13; 15 \rangle$ and $T_{t_3} = \langle 54; 56 \rangle$.

Timed automaton of *Task* is presented in Figure 11 a). It consists of WaitNextPeriod, NextPeriod, Ready and Running locations. WaitNextPeriod, NextPeriod locations set the task period using `nextPeriod[taskID]` channel. Ready location waits for the processor timed automata when it is idle (channel `procRun`). The Running location models the task computation time given by constant `C` and its processor release (channel `procDone`).

The *processor* automaton is the same the one presented in Figure 8 b).



a) Task b) Observer for interval $\langle m; m+2 \rangle$

Fig. 11. Optimal period timed automata

The *observer* model (see Figure 11 b)) is used to set one of the possible task periods from the given period interval. Each observer belongs to one task model. It consist of the initial–committed location Choose with several output transitions that are not constrained. Therefore, the model can choose one of the possible transitions at the system initialisation. Locations T1 ... Tn and subsequent self–loop transition set the related task period. The period in this structure is set by clock variable `tperiod`. Each loop period is given by an integers from task period interval, the loop activates corresponding task using the `nextPeriod[taskID]` channel.

The depicted example of the observer is defined for interval $\langle m; m+2 \rangle$. If the m is equal to 5, the possible modelled period is $T_{t_{id}} \in \{5, 6, 7\}$.

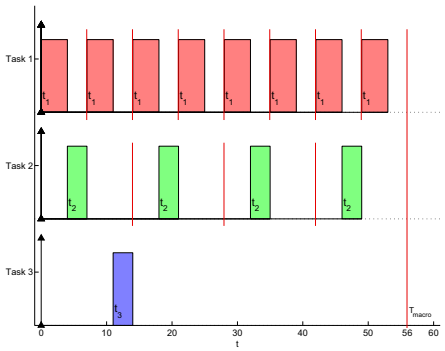


Fig. 12. Optimal schedule for taskset with bounded periods

To find the optimal schedule, the property `E<> t1.NextPeriod && t2.NextPeriod && t3.NextPeriod && global_timer > 0` has to be satisfied. Each optimal period value is given by the selection of period branch of the accordant *observer* timed automaton, where the selection is based on the result of the best cost diagnostic trace. For the case study it is $T_{t_1} = 7$, $T_{t_2} = 14$ and $T_{t_3} = 56$.

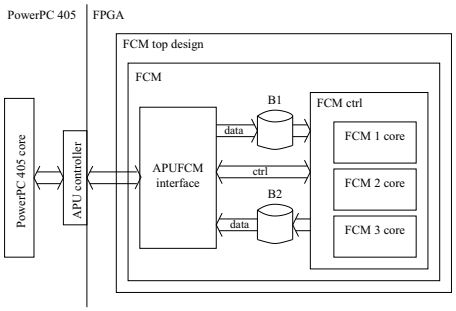


Fig. 13. Hardware structure

The optimal schedule is depicted in Figure 12. The processor utilization mentioned is 94.96% because the the sum of the processor idle time $cost = 9$, and the makespan is 56. For that $100 - 9 \cdot (56/100) = 94.96\%$.

III. Experiments on FPGA platform

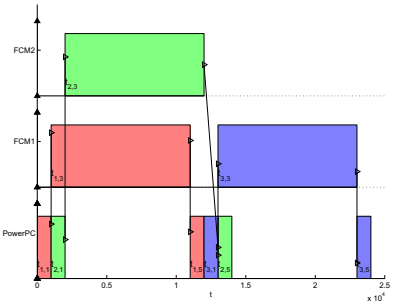
The section shows the use of the minimisation of makespan schedule technique directly at Xilinx Virtex-4 4VFX12 plat-

form. Due to the FPGA architecture the application is divided into FCMs in FPGA (hardware) and dispatcher task running on PowerPC (software).

The hardware structure (Figure 13) includes 3 FCM arithmetic units. Each unit increments an input number received from PowerPC and delays the output for user defined time. The input data from APU are stored into buffer B1. The output data to be uploaded by APU are stored in one-place buffer B2. Each FCM works as a delay module set to 10000 ns.

The application in PowerPC uses the scheduling technique presented in subsection II-A. The system parameters are $C_{t_{1,1}} = C_{t_{2,1}} = C_{t_{3,1}} = 1000\ ns$, $C_{t_{1,3}} = C_{t_{2,3}} = C_{t_{3,3}} = 10000\ ns$ and $C_{t_{1,5}} = C_{t_{2,5}} = C_{t_{3,5}} = 1000\ ns$.

Optimal schedule with makespan 24000ns and the *dispatcher task* code, derived from the schedule, are depicted in Figure 14. The FPGA behavior is shown in Figure 15. Each FCM occupation is visible in *fcmc1_busy* and *fcmc2_busy* line.




```
1  void dispatcherTask(){
2    // Reset all FCM units
3    fcm_reset();
4    // Write data into FCM1 and FCM2
5    fcm_write(FCM1, src[0]);     //t_{1,1}
6    fcm_write(FCM2, src[0]);     //t_{2,1}
7    fcm_wait(FCM1);              //t_{1,5}
8    dst[FCM1] = fcm_read(FCM1); //
9    fcm_write(FCM1, src[0]);     //t_{3,1}
10   fcm_wait(FCM2);              //t_{2,5}
11   dst[FCM2] = fcm_read(FCM2); //
12   fcm_wait(FCM1);              //t_{3,5}
13   dst[FCM1] = fcm_read(FCM1); //
14 }
```

Fig. 14. Optimal schedule and *dispatcher task* listing

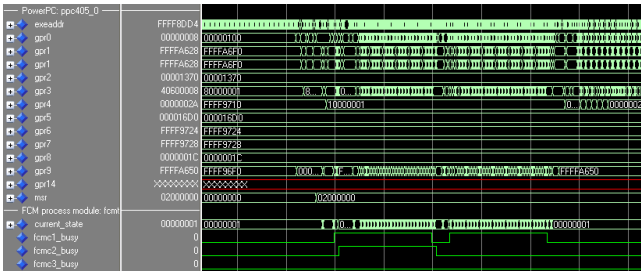


Fig. 15. Modelsim FPGA simulation

IV. Conclusion

Based on Priced Timed Automata (UPPAAL CORA tool), the article shows a schedule synthesis techniques minimising makespan and sum of completion times criterion. Moreover, it presents a synthesis methodology to guarantee the budget available on processor and maximization of processor utilization for tasks with bounded periods. The study considers that tasks are either executed once or they are periodic. A simple experiment, that shows an implementation of a given task schedule directly into presented FPGA platform, is finally presented.

Complexity of the state space given by timed automata is well known drawback. It can be solved by simplification of the models as presented in subsection II-A.

In this work, the real-time operating system behaviour is neglected. In further work we would concentrate on synthesis and implementation of applications running on a real-time operating system (e.g. Linux with fully preemptive patch) with the FCM co–processor.

References

[1] Yasmina Abdeddaim and Oded Maler. Job-shop scheduling using timed automata. In *Computer Aided Verification*, pages 478–492, 2001.

[2] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 4, Washington, DC, USA, 1998. IEEE Computer Society.

[3] Karine Altisen, Gregor Gler, Amir Pnueli, Joseph Sifakis, Stavros Tripakis, and Sergio Yovine. A framework for scheduler synthesis. In *IEEE Real-Time Systems Symposium*, pages 154–163, 1999.

[4] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464, London, UK, 2002. Springer-Verlag.

[5] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, 2005.

[6] Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Gunter Schmidt, and Jan Weglarz. *Scheduling computer and manufacturing processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[7] Muhammad Omer Cheema and Omar Hammami. Application-specific simd synthesis for reconfigurable architectures. *Microprocessors and Microsystems*, 30(6):398–412, 2006.

[8] Ansgar Fehnker. Scheduling a steel plant with timed automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press, 1999.

[9] Thomas Hune, Kim Guldstrand Larsen, and Paul Pettersson. Guided synthesis of control programs using UPPAAL. In *ICDCS Workshop on Distributed System Validation and Verification*, pages E15–E22, 2000.

[10] Chang-Gun Lee and Lui Sha. Enhanced processor budget for qos management in multimedia systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 123.2, Washington, DC, USA, 2003. IEEE Computer Society.

[11] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CS-93-157, 1993.

[12] Peter Niebert and Sergio Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In *HSCC*, pages 338–351, 2000.

[13] P. Šůcha and Z. Hanzálek. Scheduling of tasks with precedence delays and relative deadlines framework for time-optimal dynamic reconfiguration of fpgas. In *The 14th International Workshop on Parallel and Distributed Real-Time Systems 2006*, Island of Rhodes, Greece, April 2006.

[14] Stavros Tripakis. *L'analyse formelle des systemes temporiss en pratique*. PhD thesis, UNIVERSITE JOSEPH-FOURIER - GRENOBLE I, 12 1998.

[15] Inc. Xilinx. *PowerPC 405 Processor Block Reference Guide*. Xilinx, Inc., 2005.

[16] Inc. Xilinx. *ML40x EDK Processor Reference Design*. Xilinx, Inc., June 2006.