Strategies to Improve Synchronous Dataflows Analysis Using Mappings between Petri Nets and Dataflows

José-Inácio Rocha[1,2,3], Octávio Páscoa Dias, and Luís Gomes[1,3]

[1] Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologias, Portugal
[2] Escola Superior de Tecnologia de Setúbal, Setúbal , Portugal
[3] UNINOVA – Centro de Tecnologia e Sistemas, Portugal
jose.rocha@estsetubal.ips.pt, lugo@uninova.pt
octavio.pdias@gmail.com

Abstract. Over the last decades a large variety of dataflow solutions emerged along with the proposed models of computation (MoC), namely the Synchronous Dataflows (SDF). These MoCs are widely used in streaming based systems such as data and video dominated systems. The scope of our work will be on consistent dataflow properties that can be easily demystified and efficiently determined with the outlined mapping approach between Dataflows and Petri nets. Along with this strategy, it is also highlighted that it's of a major relevance knowing in advance the proper initial conditions to start up any SDF avoiding buffer space over dimensioning. The methodology discussed in this paper improves the outcomes produced so far (in Petri net domain) at design stage aiming at knowing the amount of storage resource required, as well as has a substantial impact in the foreseen allocated memory resources by any signal processing system at the starting point and also points out new directions to minimize the buffer requirements at design stage.

Keywords: Dataflows, Petri nets, Synchronous Dataflows, Place and Transition Invariants, Models of Computation.

1 Introduction

Over the last decades a multitude of dataflow solutions emerged along with requirement always supported by the needs of each new Model of Computation subtleties and peculiarities. Researchers have developed a few number of dataflow variants, trying to model the different constraints presented in the firing rules. These constraints about the firings rules are what differentiate several developed MoCs. For example in Cycle-static Dataflow [1], tokens rates can be scheduled in a periodic way, while in Dynamic Dataflow [2] token rates can be changed after each schedule. Another variant called Parameterized Synchronous Dataflow [3] allows the tokens rates to be defined by a parameter, which can be changed only between schedules.

Synchronous Dataflow (SDF) is a Model of Computation where the number of tokens consumed or produced by an actor is known in advance at compile time, i.e. without executing the model. This way, a SDF can be viewed as a statically schedulable model [4]. Every SDF model is composed by nodes and arcs (also usually called edges or channels). The nodes (graphically correspond to circles or ellipses) represent the functional or activity elements, while the edges (symbolized by arrows) represent communication channels between them. These channels are conceptually first-in-first-out queues. Edges are tagged with weights appended on both ends by non-negative integers. Due to their activity nodes consume and produce an establish amount of tokens on each firing based on tagged weights in the arcs. Tokens (represent by black dots) can also be seen as delays; therefore edges may contain initial markings which correspond to an initial number of delays.

SDF model have demonstrate their adequacy in many signal processing and multimedia domains, in which there is no data-dependency in each schedule, i.e. the tokens rates remains constant after each actor firing, therefore the rates of tokens are fixed by nature. But this weakness (lack of expressiveness) can be viewed as its strength, since it's possible to analyze all models for deadlock or bounded buffers.

Several advantages can be outlined to SDFs models, namely the easiness in modeling signal processing multi-rate systems, the ability to generate static schedule at compile time (allowing to know the required resources in advance), and the sequential static schedules can be optimized for buffer memory minimization, as well as allowing efficient simulation and efficient code generation. In a nutshell, SDF are well-matched to signal processing and communications systems. In [5] and [6] the researchers aimed at unveiling the effective and potential maximum number of tokens for each arc in every Synchronous Dataflow to foresee at compile time the necessary amount of storage resources, i.e. at design stage, as well as the study of dataflow models under a cyclic and continuous flow of data, whereas in this paper new developments are highlighted in the scheduling analysis and on reducing the allocated buffer memory by buffer sharing.

The paper is organized in seven topics. After a short introduction and motivation about dataflows, a brief discussion about the issue under investigation in the present paper to the relationship to collective awareness systems is outlined. In a nutshell, the state of the art provides an overview upon the matter under research. Section 4. ("The proposed methodology") presents and depicts the basic ideas behind the mappings outlined between dataflow and Petri net domains. Experimental work and results are shown in section 5. In section 6 it is presented in a summarized way the tools and computational environments used to work out the conclusions. The paper ends up with some guidelines that will rule the further research work on the key area of interest.

2 Relationship to the Collective Awareness Systems

Collective awareness is an issue that is nowadays being addressed in existing literature by several researchers due to the need to improve the knowledge of the impact of the different scenarios that are growing mainly on the side of the technological "footprint". The need to seek for understanding at what surround us, which make us above all a social creature, is demanding a worldwide coordination and awareness to gather in a more fruitful way all these new collaborative emerging realities.

The proposed approach in this document maintains a close regarding to collective awareness and aims its future research to a subject which is a topic being conveyed as a worldwide awareness (a wide collective awareness) in terms of earth's sustainability, the optimization energy consumption in order to fulfill very tight power requirements demanded by a target application or an embedded system. It is a vital subject since our society is founded on energy and due to increasing demanding on portable electronic devices and massive utilization, optimization of power (energy) consumption has become a critical design point. The industry of low power consumer electronics is currently submitted to a huge growth, while the power dissipation of digital systems experiences an increase in device density, complexity and speed, and the last but not the least the growing interest in lower power chips and systems are driven by business and demanding technical needs. Researchers and system developers are aware of the tradeoff between power and performance [7]. Moreover, tools and strategies can be developed or improved to go on even further to pursue the goal of devising optimized applications and embedded systems on different platforms. Memory resources are scarce in every signal processing system, therefore the contribution of tools to ameliorate this topic are of great significance nowadays when used together with the lowest sleep and active power draws hardware techniques. Our approach intends to be a contribution aiming at knowing the minimal allocated resource memory even at and during a system's startup period.

3 Summarized Overview of the Supporting Formalisms

Actually Embedded Systems have a multitude of constraints and non-functional requirements, as a result to perform the synthesis or code generation efficiently of models the practitioner must deal with several issues, ranging from energy constraints, buffer requirements, memory efficiency and performance. To deal with these matters, two modeling formalisms are used in this work: Dataflows and Petri nets. In order to present Dataflows and Petri nets a summarized overview is provided, for a more complete and formal descriptions of Petri nets see for example [8, 9 and 10] and for Dataflows see [11, 12 and 13].

3.1 Dataflows

The basic idea of the principles of a language for parallel processing outlined by Gilles Kahn [11] in his seminal paper was to represent programs using nodes and arcs. Later on, Dennis [12] established the dataflow principles aiming at exploitation of parallelism in programs. In a dataflow, nodes are associated to computational program activity, and arcs connecting nodes convey the flow of information. Arcs are First-In-First-Out queue channels with blocking read operations and non-blocking write operations. Lee et al. [13] presented in 1987 a special kind of dataflow known as Synchronous Dataflow (SDF) where the token rates (incoming and outgoing) are fixed and established by nodes on each arc, thus improving the runtime overhead of the modeled system. Nowadays SDFs are a mature approach and are well disseminate among stream based system, namely parallel and distributed signal processing systems, since one can detect deadlocks at compile time, as well as have demonstrate their adequacy in specifying multirate systems that can be statically scheduled. Figure 1 (a) illustrate a SDF model where some arcs contain black dots representing initial markings (which can also be known as tokens or delays).

3.2 Petri Nets

A Place-Transition Petri net is a special bipartite directed graph (consisting of two kind of nodes, places and transitions), and can be viewed as a graphical and mathematical modeling tool. As a graphical tool, Petri nets are a valuable tool since they can be used as a visual aid to simulate the concurrent and dynamic activities of the systems. As a mathematical tool, it is possible to address algebraic equations, state equations and other mathematical formalisms that govern the system's behavior and structure. The origin theory of Petri nets is based on the work of Carl Adam Petri dissertation's published in 1962. Graphically, places are drawn as circles or ellipses, and transitions as bars or rectangles. Arcs are tagged with positive integers representing weights, and are either from a place (/transition) to a transition (/place). Places may hold tokens defining the state of the system. In the graphical environment tokens assigned to places correspond to black dots or positive integer numbers. The activity of Petri nets is ruled by enabling and firing transitions. In a word, a transition is enabled if each input place is marked with a minimum number of tokens fixed by the arc weight, and whenever enabled the transition will eventually fire. In figure 1. (b) one can see an example of a Petri net model.

4 The Proposed Methodology

The fundamental idea of our methodology is to make use of an inter-relation between a Synchronous Dataflow (SDF) and a transformed equivalent Petri net (PN). The idea was first presented in [14]. This way one can encompasses and take benefit from the well known properties verification capabilities of Petri nets, namely invariants and reachability analysis. Based on this inter-relation between SDF and PN, valuable information is added to the scheduler allowing one to gain insight knowledge about the storage resource in each arc, as well as the resource allocation of the whole SDF by realizing an inverse mapping into dataflow domain.

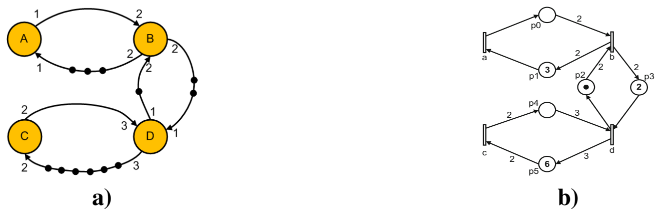


Fig. 1. An example of a Dataflow model, where some arcs contain initial tokens (a) and Petri net model (b)

The complete mapping strategy is depicted in figure 2. In a summarized way, one tries to find a relationship between every element in each domain, keeping in mind that each of these environments due to its nature has passive and active elements.

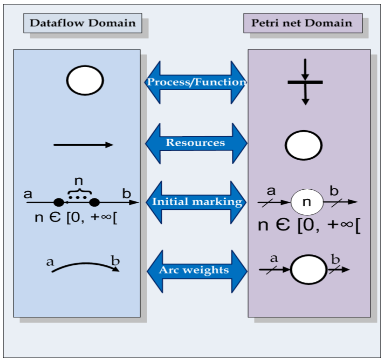
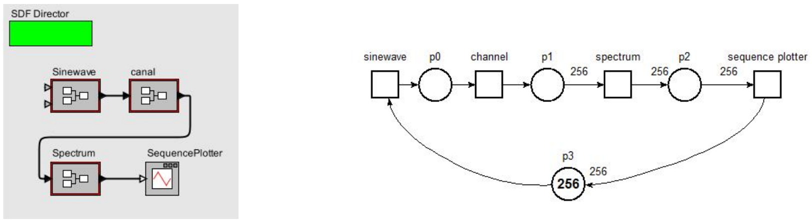


Fig. 2. Proposed mapping between dataflow and Petri net domain

This way, a process/function (represented by a circle) in a dataflow corresponds to a transition in Petri net domain (depicted by a bar). Since arcs in dataflows represent paths that conveys the oriented stream of information, arrows are used between the processes. The matching element on the Petri net side for the oriented path is a circle, embodying normally conditions, states, resources or objects. Arcs in a dataflow also include weights and initial markings denoted by non negative integer tagged at both ends of the arrow and black dots, respectively. The previous two characteristics find their matching elements in the lower part of figure 1, on the side of Petri net domain.

To illustrate our approach, a dataflow is presented in figure 1 (a) and the corresponding equivalent Petri net model is shown in figure 1 (b).

A Petri net with **m** places and **n** transitions is represented by a matrix *(mxn)* called the incidence matrix C. The entries of the incidence matrix are integers representing the token balance associated to the firing of every transition in the Petri net. If there is token gain the numbers are positive, for a loss and no change on the associated transitions, negative and zero entries in C are used respectively. This matrix is the key element to establish the transition or fundamental equation which governs the new markings on the Petri net given an initial marking set. Based on a qualitative analysis of PN two properties support our analysis: (1) Place Invariant (P-Invariant); and (2) Transition Invariant (T-Invariant). Based on the incidence matrix one creates a set of homogeneous linear equations which allows finding the so called P and T-Invariants. The structural domain analysis is aimed at designing systems without any kind of inconsistencies coping with properties which stay constant during model's execution. Maximum potential number of tokens is computed with P-Invariants, as presented in [15], whereas the maximum effective number of tokens on each arc in a dataflow is achieved summing all values presented in the reachability tree. More over with T-Invariants one can foresee the potential initial markings (M_0) for each place. Using the reachability tree one can keep track of the maximum number of tokens present on each arc during a schedule. This way the buffer size for each individual arc is foreseen on the dataflow. Gathering all these values the total buffer requirements for the arcs present in a dataflow is estimated.



a) **b)**

Fig. 3. An implementation of a multi-rate SDF Dataflow model in Ptolemy environment (a) and Petri net equivalent model supported by the translation rules presented in Fig. 2

The spectrum actor in figure 3 requires multiple input tokens to fire. This input multiplicity is established by a parameter called order, which is always a number dependent of power 2. The icon "SDF Director" governs the proposed dataflow model by defining the number of iteration to be performed and additionally determines the number of times each actor fires in one iteration using the balance equations. The multi-rate SDF model presented in figure 3. (a) is called well ordered chain structured SDF, and for these cases it is necessary to add an extra place (p_3) to make the whole Petri net reversible and live (for further reading see [9]), because at the end of each periodic schedule the digital signal processing system exhibits cyclic behavior since there is always an imbalance in the number of tokens. Besides this condition is central to evolve our methodology which is based on the P/T-Invariants in Petri net domain (see [6]).

5 Experimental Work and Results

One may argue that the reachability tree is a bottleneck due to the state space explosion problem; however, it is possible to distribute the state space in a set of computing machines [9] limiting this drawback. The experimental work was carried on two kinds of (well-known) problems, the dining philosophers and distributed database, besides the process is still dependent on the choice of having a good hash function to guarantee also a good state's distribution among the processors.

5.1 Scheduling Analysis of Synchronous Dataflow

Analysis of ill-specified SDF at the design time or even the misbehaved ones is an issue of major relevance. In a SDF our concern is related to attain a static schedule (if one exists), where a finite periodic list of actors is schedule in a sequential way. A SDF can evidence either rate-inconsistency (motivated by unbounded memory problems), or inability to run, or even deadlock caused by lack of initial buffering. As stated in Lee et al. [4] "a periodic admissible sequential schedule (PASS) is a periodic and infinite admissible sequential schedule", one concludes that after a period has been executed, the size of each buffer returns to its initial state, i.e. or to its home state (as it is known in Petri net domain).

To illustrate our idea considers the following flowchart depicted in figure 4. Taking the equivalent Petri net model one can identify easily if either a SDF model is consistent or inconsistent, or has deadlocks, using the P-T Invariants (structural analysis) and the reachability tree (behavioral analysis). Therefore, it is possible to draw a methodology (on the side of Petri net domain) for each and every SDF to find out their runnability feature. Figure 5 presents all those cases of runnability in SDFs models.

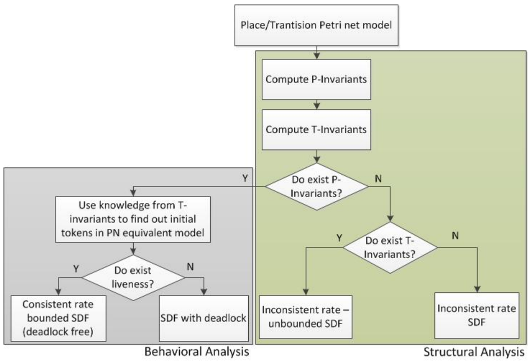

Place/Trantision Petri net model
Compute P-Invariants
Compute T-Invariants
Do exist P-Invariants?
Y
N
Use knowledge from T-invariants to find out initial tokens in PN equivalent model
Do exist T-Invariants?
Y
N
Do exist liveness?
Y
N
Consistent rate bounded SDF (deadlock free)
SDF with deadlock
Inconsistent rate – unbounded SDF
Inconsistent rate SDF
Behavioral Analysis
Structural Analysis

Fig. 4. Flowchart to identify the runnability (or rate consistency) of SDFs

As another example consider the simplified spectrum analyzer dataflow presented in figure 6 (a) and based on the translation rules outlined in section 4. one can get the Petri net model shown in figure 6 (b). Using the Petri net structural analysis, it is possible to know the maximum potential and effective amount of storage space for the static schedule identified, as stated before. The amount of storage space for each arc ($M(p_i)$ stands for the number of tokens in place p_i) are shown in figure 7 (a) and the associated static schedule in figure 7 (b). The total buffer requirement for the spectrum analyzer is 17 units, where each arc in dataflow domain maps to a place in Petri net domain. In the reachability tree achieved under Petri net domain one must find the home state in order to know the execution time of the periodic static schedule and at the same time can observe the execution pattern of each process in a single schedule as shown in figure 7 (b). If every process (transition) takes 2 time units, the entire schedule execution time will take place in 18 time units. From figure 7 (b), one can observe that in every periodic static schedule the Adaptive LPF (ALPF) process executes 4 times, FFT Zoom (FFT Z), Peak Detector (PD), Interpolator (INT) and Zoom Control (ZC) executes one time and that Decision (DECI) executes 2 times.

The use of the reachability tree can be seen as a bottleneck by some researchers due to the state space explosion problem; however it is possible to distribute the state space in a set of computing machines [16] limiting this drawback. Additionally to know the real impact of the initial conditions in the spectrum analyzer multiple simulations were performed using the computational environments to estimate the number of states generated, as well as the number of fired transitions in the foreseen static schedule.

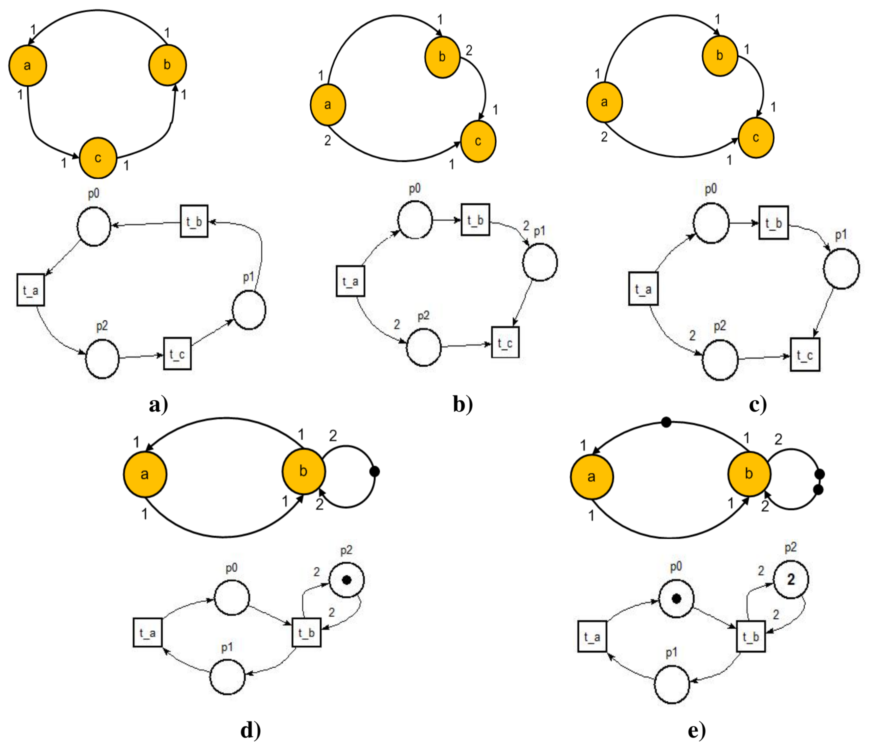


Fig. 5. Synchronous Dataflows and their equivalent Petri net models for different runnability cases; (a) rate consistent with bounded memory; (b) rate-inconsistent with unbounded memory; (c) rate-inconsistent – cannot run; (d) rate consistent with deadlock; (e) rate-consistent without deadlock

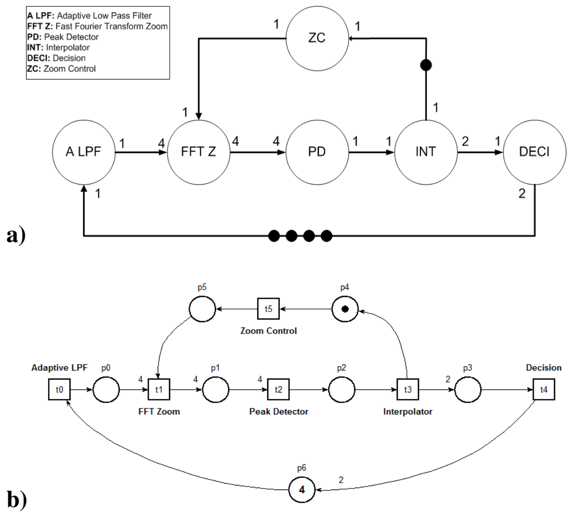


Fig. 6. SDF of spectrum analyzer (a) and Petri net equivalent model (b)

Observing *Table 1* among the various cases identified one (**case 1**) conducts the system to a minimum in the number of states and fired transitions. Therefore from the other cases in *Table 1* one can conclude that initial conditions (in place p_1 and p_4) will make the modeled system progress to some extra states that are not relevant for the periodic static schedule, thus augmenting the state space and at the same time unnecessarily over dimensioning the system. Thus it is important to develop an algorithm to find out the initial conditions that conduct the modeled system to a minimal state space and also reduces the required storage space.

5.2 Reducing Buffer Memory by Buffer Sharing

In order to reduce even further the allocated buffer space among the arcs it is important to unveil those that don't overlap their live ranges stressing that each arc is committed to a First-In-First-Out (FIFO) buffer. The strategy to follow is to observe the way each place in each state behaves traversing the state space of the schedule and then, decide what are the places that don't overlap at each state by grouping those that are disjoint in state space schedule.

Table 1. State space and fired transitions for several cases for different initial marking on places p_1 and p_4 ($M_0(p_1)$ and $M_0(p_4)$), for figure 6 (b)

Case	M0(p1)	M0(p4)	# states	# fired transitions
1	4	1	20	32
2	4	2	31	56
3	8	1	68	144
4	8	2	114	276
5	5	1	28	50
6	7	1	52	106

Amount of storage space per arc							**Total**
M(p0)	M(p1)	M(p2)	M(p3)	M(p4)	M(p5)	M(p6)	
4	4	1	2	4	1	1	**17**

a)

Schedule	Time stamp								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	ALPF, ZC	ALPF	ALPF	ALPF	FFTZ	PD	INT	DECI	DECI	
2										ALPF ,ZC

b)

Fig. 7. Buffer requirement for a periodic static schedule of spectrum analyzer dataflow (a), and the corresponding static schedule (b)

After this identification one is able to state which buffers can be fully shared. To illustrate our methodology, an example is used, the spectrum analyzer. The corresponding SDF is shown in figure 6. (a). The equivalent Petri net model of the spectrum analyzer (in figure 6. (b)) is used as an example to illustrate the issue of the buffer sharing. This way the following buffer sharing is possible: places p_0, p_1, p_2 and p_3 can share the same buffer and requires a buffer size of 4 units, which is the maximum amount of storage in this subset of places; places p_4 and p_5 can also share the same buffer and require a buffer with unitary dimension; and finally place p_6 needs a buffer of 4 units. Summing the contributions of each group of shared places one attains the total amount of buffer space of 9 units, representing a space memory saving of 47 percent in comparison to the initial solution.

6 Tools and Computational Environments

The development of the equivalent transformed Petri nets was performed by two non commercial tools that are freely available in the web: (1) Integrated Net Analyzer INA [17], a non graphical Petri net editor, that requires a previous knowledge of the INA syntax language to perform a net description; (2) Still TINA - TIme Petri Net Analyzer [18], a graphical editor with a stepper simulator to perform a step by step animation in a Petri net. The invariant analysis (used in section 5) was performed in both environments.

7 Conclusions and Further Work

The proposed methodology allows one start from a dataflow model, a Synchronous Dataflow, and based on a set of mapping rules achieve an equivalent Place – Transition Petri net model exploring the well-known verification properties: Behavioral (Structural) properties which are dependend (independent) on the initial marking. In this paper the focus is on the behavioral properties: reachability, boundedness, deadlocks, liveness, reversibility and home state. Using reachability tree one can identify if from a initial marking it is always possible to reach that initial state (referred in this case as home state), or know the firing sequence that brings the equivalent Petri net to the home state. Boundedness was useful to get the required buffer space for each Synchronous Dataflow arc and reveal if no design errors were committed at design stage. Liveness is a major property for the stream based SDF models since data flows sequentially, continuously and iteratively since ones tries to find a schedule, i.e., guarantee the absence of deadlocks.

On the side of structural properties the analysis is focused on the invariants (P and T) to achieve the buffer (token) capacity concerning each arc in SDF models. Initial conditions in dataflow models play an important role since with the proper initial conditions the system will not be over dimensioned, which are not desirable and should be avoided. Moreover, choosing different initial conditions one can state and conclude that the modeled system will evolve to some states that are not relevant for the periodic static schedule, thus augmenting the state space will jeopardize (more time consuming task) the search for specific states, such as for example the home state. In resume, the goal is to look for initial markings that make system runnable to complete periodic static schedules. As future work, one should find an algorithm to identify amongst all firing conditions those that lead the system to a minimal state space encompassing a finite periodic static schedule using the modeling formalism of PN. Also outline an algorithm (or an optimization algorithm) to find regions (or sub-regions) in the state space where a buffer sharing may be possible. This strategy will allow reducing to a great extent the buffer requirements.

Acknowledgments. This work was partially financed by Portuguese Agency "FCT - Fundação para a Ciência e a Tecnologia" in the Framework of projects PEst-OE/EEI/UI0066/2011 and PTDC/EEI-AUT/2641/2012 and by Instituto Politécnico de Setúbal (IPS) in the framework of projects of "3º Concurso de Projectos do IPS", reference 3CP-IPS-8-2009.

References

1. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cycle-static dataflow. IEEE Transactions on Signal Processing 44(2), 397–408 (1996)
2. Buck, J.T., Lee, E.A.: Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 1993, vol. 1, pp. 429–432 (April 1993)
3. Bhattacharya, B., Bhattacharyya, S.S.: Parameterized Dataflow Modeling for DSP Systems. IEEE Transactions on Signal Processing 49, 2408–2421 (2001)
4. Lee, E., Messerschmitt, D.G.: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Transactions on Computers C-36(1), 24–35 (1987)
5. Rocha, J.-I., Gomes, L., Dias, O.: Petri net verification techniques on synchronous dataflow models. In: IECON 2011 37th Annual Conference on IEEE Industrial Electronics Society, pp. 3792–3797 (2011)
6. Rocha, J.-I., Páscoa Dias, O., Gomes, L.: Exploiting Dataflows and Petri Nets Mappings. In: 2013 11th IEEE International Conference on Industrial Informatics (INDIN), pp. 590–595 (2013)
7. Yeap, G.K.: Practical Low Power Digital VLSI Design. Springer (1997)
8. Giraud, C., Valk, R.: Petri Nets for Systems Engineering. A Guide to Modeling, Verification, and Applications. Springer, Heidelberg (2003)
9. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)
10. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River (1981)
11. Kahn, G.: The Semantic of a Simple Language for Parallel Programming. In: Proc. of the IFfP Congress 74, vol. 74, North-Holland Publishing Co. (1974)
12. Dennis, J.: First version of a data flow procedure language. In: Robinet, B. (ed.) Programming Symposium. LNCS, vol. 19, pp. 362–376. Springer, Heidelberg (1974)
13. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE 75(9), 1235–1245 (1987)

14. Rocha, J.-I., Gomes, L., Páscoa Dias, O.: Dataflow Model Property Verification Using Petri net Translation Techniques. In: INDIN'2011 - 9th IEEE International Conference on Industrial Informatics, pp. 783–788 (2011), doi:10.1109/INDIN.2011.6034993, ISBN 978-1-4577-0434-5
15. Rocha, J.-I., Gomes, L., Páscoa Dias, O.: Analysing Storage Resources on Synchronous Dataflows using Petri Net Verification Techniques. In: IECON'2012 – The 38th Annual Conference of the IEEE Industrial Electronics Society (2012)
16. Boukala, M.C., Petrucci, L.: Towards distributed verification of petri nets properties. In: Barkaoui, K., Ioualalen, M. (eds.) Proceedings of the First International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS 2007), pp. 13–24. British Computer Society, Swinton (2007)
17. INA - Integrated Net Analyser, Humboldt – Universita zu Berlin, `http://www2.informatik.hu-berlin.de/starke/ina.html`
18. TINA-TIme Petri Net Analyzer, Laboratoire d'Analyse et d'Architecture des Systémes, `http://homepages.laas.fr/bernard/tina/`