EVOLUTIONARY REQUIREMENTS FOR NEXT-GENERATION DATAFLOW-BASED FPGA SYSTEM DESIGN

John McAllister

Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast
Belfast, Northern Ireland, UK
phone: + (44) 2890971743, email: j.mcallister@ecit.qub.ac.uk
web: www.ecit.qub.ac.uk

ABSTRACT

The use of dataflow digital signal processing system modelling and synthesis techniques has been a fruitful research theme for many years and has yielded many powerful rapid system synthesis and optimisation capabilities. However, recent years have seen the spectrum of languages and techniques splinter in an application specific manner, resulting in an ad-hoc design process which is increasingly dependent on the particular application under development. This poses a major problem for automated toolflows attempting to provide rapid system synthesis for a wide ranges of applications. By analysing a number of dataflow FPGA implementation case studies, this paper shows that despit ethis common traits may be found in current techniques, which fall largely into three classes. Further, it exposes limitations pertaining to their ability to adapt algorith models to implementations for different operating environments and target platforms.

1. INTRODUCTION

The emergence of high-performance digital signal processor chips and dataflow programming languages in the late 1980's and early 1990's resulted in a spate of activity researching the use of dataflow for embedded digital signal processing (DSP) system design. The seminal work on Synchronous Dataflow [3] sparked a rich period in the development on new languages to support everything from cyclically varying actor execution [6] to data-level parallelism [5] amongst others. This has occured in tandem with complementary research into exploiting the semantics of different languages for intelligent multiprocessor system synthesis approaches to produce high-quality embedded implementations [2, 14]. This evolution has yielded extended scope of the dataflow design field to include FPGA technology ([10, 11, 8]), which is currently a significantly less mature research area than DSP-based dataflow research.

In an FPGA system design context, the need to synthesise a processing architecture as well as target the application toward that architecture significantly complicates the system synthesis process when compared with multiprocessor software synthesis. Combined with the increasing diversification and expressivity of current dataflow languages and manipulation techniques, it is simply too complex a problem for a single designer on a specific application design to produce a near-optimal implementation using such techniques. As such, there is a key requirement in this area for system level design tools and methodologies. However, a pre-cursor to such tools are generalised, well-known and well-understood programming languages, language manipulation techniques, and target devices. Given the dizzying array of cutting-edge, application specific dataflow languages, ad-hoc manipulation techniques and open FPGA architectures, this is simply not feasible at present.

This paper analyses for FPGA dataflow system synthesis case studies and attempts to extract common traits. It finds that dataflow graph (DFG) manipulation techniques fall generally into one of three categories: vertex topological, data topological and semantic. In all cases the application of specific DFG manipulation for efficient implementation is driven by one or both of the operating environment and the target platform, yet no research is underway addressing formal integration of these aspects into new dataflow languages.

Whilst this is not a result in itself, this paper acts as a starting point for generalised dataflow design for FPGA, exposing the limitations in current approaches and encouraging research into new classes of dataflow system modelling and manipulation technique.

The remainder of this paper is as follows; Section 2 provides an introduction to the dataflow computational domain and dataflow based system design, whilst Section 3 analyses four design case studies, extracts the major themes in their design processes and attempts to put these in a coherent design framework on which to build a dataflow FPGA design methodology and toolset, and on which to extend current dataflow research in this area.

2. BACKGROUND

The roots of the current dataflow programming languages lie in the Dataflow Process Network (DPN) model [4], which describes parallel processes or actors communicating via unidirectional first-in first-out (FIFO) queues. Actors map data token on input FIFOs to tokens on output FIFOs by *firing* to *consume* input tokens and *produce* output tokens. A set of firing rules determine, for each actor, how and when it fires. A simple DPN is shown in Figure 1. For synthesis and optimisation of an embedded implementation of a simple dataflow graph (DFG) $G = (V, E)$, where V is a set of vertices or actors, and E a set of FIFO edges, a methodology such as that in Figure 2 is common [2, 1].

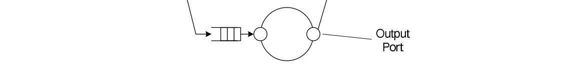


Figure 1: Simple DPN Structure

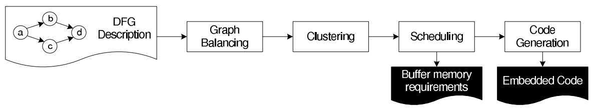


Figure 2: Rapid Software Synthesis from DFGs

Algorithm level manipulations change the structure of the algorithm topology itself by manipulating the topological arrangement of actors and edges. The subsequent *graph balancing* manipulates the numbers of firings of each actor to ensure, if possible, bounded memory execution or to manipulate the number of firings for efficient implementation and minimisation of run-time overheads (e.g. the blocking schedules of [3]).

Having determined the number of actor firings in an iteration of the schedule, these are *clustered* into groups of one of more firings for scheduling. This balance of the clustering operation can influence the run-time efficiency of the implementation and the memory requirements [14]. Optimisations at the *Scheduling* and *Code Generation* levels order the execution of clusters of actors, and have the ability to negate the effectiveness of optimisations at the higher levels. For instance, in the case where low data memory requirements are paramount, if actors are gathered into a small cluster at the previous level, it is a requirement of this step to interleave the schedule of the implementation such that source and destination cluster instantiations are interleaved, such that the intermediate FIFO buffer memories may be reused. Alternatively, if small instruction memory is of paramount importance, the clustering step must create large clusters of actor firings such that the scheduler must then created looped schedules, which may be efficiently implementation by looped code produced by the code generator [2].

Such an approach is difficult to extend to FPGA systems, where structure as much as behavior must be manipulated. For multiprocessor implementation it is the firing of an actor, rather than the presence of an actor on the DFG itself which is relevant. This may not be the case for FPGA design, where manipulating the graph topology itself via various transformations may provide more designer control of the implementation structure [10, 9, 8, 12]. Furthermore, in the literal context in which DFG structures are translated to coarse grained dedicated hardware on FPGA, although some graph manipulation techniques such as loop skewing in Compaan [10] do allow schedule manipulation of the implementation, it is difficult to judge how effective this may be in Globally Asynchronous, Locally Synchronous (GALS) architectures. As such the 'standard' FPGA dataflow design approach, if it exists, is quite different from that above, placing an emphasis on structure as much as behaviour.

Consider also the place of dataflow languages in FPGA system design processes such as Function-Architecture Codesign (FAC), Figure 3 [15]. In such a design process there is an iterative loop beginning with algorithm specification, followed by mutual refinement of the algorithm and architecture from a set of components until a satisfactory functional/real-time performance point is met. Whilst dataflow is an excellent choice for algorithm specification, there are three major problems with its use in such a design strategy:

- Modern dataflow languages and manipulation techniques are increasingly becoming application specific and ad-hoc. there is no way to integrate such random manipulation into a toolset for general use.
- Whilst the environment in which an algorithm operates pervades every facet of the algorithm, from filter lengths and numerical representation, there is no mechanism in modern dataflow languages to support parameterisation of the algorithm in terms of environment characteristics (e.g. Signal to Noise Ratio (SNR))
- Whilst the manipulation of the algorithm is dependent on the capabilities of the target platform, there is no way to incorporate these attributes into the dataflow specification during refinement

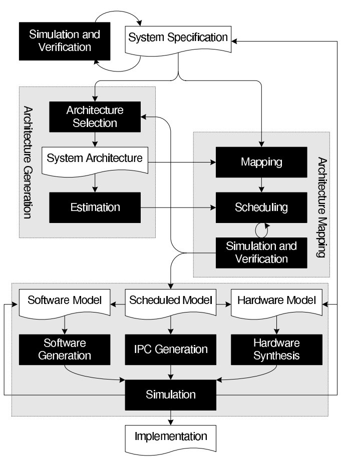


Figure 3: Function-Architecture Codesign

In short, whilst dataflow is good for specifying algorithm behaviour, it cannot be the backbone of an entire design process. Whilst they are all dataflow design processes, tools such as Compaan/Laura [10], ESPAM [11], Owen [8] and Labview FPGA all apply seemingly ad-hoc optimisation techniques to their algorithm inputs. Section 3 attempts to establishcommon traits in FPGA application synthesis approaches by examining four FPGA dataflow application synthesis examples.

3. DATAFLOW-BASED SYSTEM DESIGN

3.1 Case Studies

To illustrate the varying nature of the transformations applied, four case studies are used.

- a Normalised Lattice Filter (NLF) Bank [9]
- a 128-channel Fixed Beamformer (FxBF) [8]
- a Motion Estimation (ME) operation as part of an image processing application [12]
- a multi-channel variable length FFT [13]

The NLF and FxBF applications are modelled using the MADF computational domain [9], with the DFGs shown in Figures 4 and 5 respectively. A block diagram of the ME is shown in Figure 6.

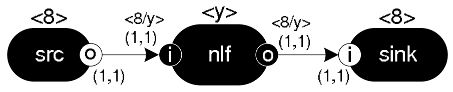


Figure 4: NLF MASDF Graph

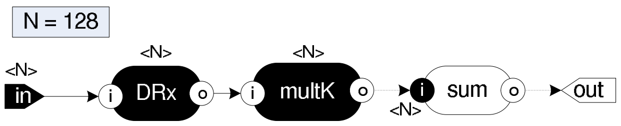


Figure 5: FxBF MASDF Graph

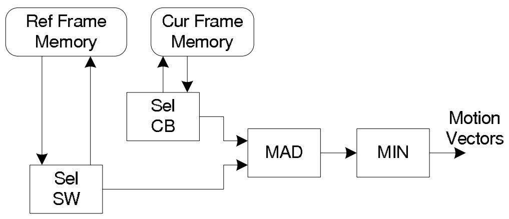


Figure 6: ME Block Diagram

In each subsequent section, the application of a different class of transformation to one or more of the examples, and the resulting influence on certain aspects of implementation efficiency, are outlined.

3.2 Vertex Topological Manipulation

Vertex Topological Manipulation describes a class of transforms employed to explicitly alter the number of vertices and edges in the DFG. In current dataflow-based FPGA design methodologies such as Compaan/Laura [10] or Mor [8], this variation in the topology of a graph is frequently used to control the number of components in the FPGA implementation, and as such has a strong influence on resource requirements and throughput of the implementation.

Consider the FxBF application; for an N-channel beamformer MADF graph as given in Figure 5, N *DRx* and *multK* actors are required. However, to implement the DRx and multK actors may require only M cores, where $1 \leq M \leq N$, with each DRx and multK core processing $\frac{N}{M}$ channels of data. Accordingly, the MADF graph for the FxBF in Figure 5 which has as many DRx and multK actors as channels processed currently, can be adapted to express this variability by introducing the factor M into the graph, as shown in Figure 7. Given this control, the DFG topology can then be synthesised toward an FPGA implementation. The results of such a synthesis operation, for varying values of M are shown in Table 1 for $N = 128$.

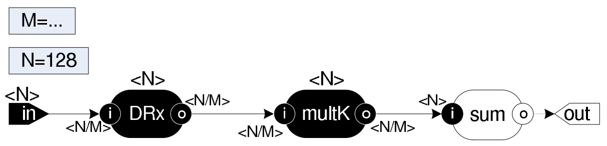


Figure 7: Modified FxBF MADF Graph

Likewise, for the NLF filter bank in Figure 4, for 8 channels of data $1 \leq y \leq 8$ cores may be utilised, each of which processes $\frac{8}{y}$ channels of data. Again, the parameter y has been specifically integrated into the MADF graph to control number of cores, allowing the graph to be synthesised to an FPGA implementation. The results of this synthesis, for varying values of y are shown in Table 2.

M	**LUTs**	**Multipliers**	**Throughput** (MSamples/S)
1	28069	99	1.45
2	29392	198	3.18
4	33130	396	6.19

Table 1: FxBF Synthesis Results

y	**LUTs**	**Multipliers**	**Throughput** (MSamples/S)
8	1472	312	397.4
2	368	78	377.9
1	393	39	208.6

Table 2: NLF Synthesis Results

In both these cases the value of such optimisation to controlling implementation resource and throughput is obvious. For the FxBF application manipulating the number of cores has enabled firstly a complex 128-channel fixed beamformer to be implemented on a resource constrained device (a Virtex II Pro 100 - which is full for option $M = 4$) and provides regular and predictable variation in resource and throughput by varying M. In the NLF designthe ability to manipulate the number of cores has enabled the exposition of a scenario whereby the 2 core version (y=2) can achieve simililar throughput to the 8 core version, with only a fraction of the hardware resource. Given the specific computational redundancy in the NLF, this kind of topological manipulation has significantly increased the operational efficiency of these components (by a factor of 3.9). Furthermore, manipulation of the parameter y has enabled the required host device complexity to drop reduce by an order of magnitude - from a Virtex II Pro 70 to a Virtex II Pro 7).

It is clear that the actualy values of M chosen is highly dependent on the environment in which the algorithm operates (to define the required throughput) and the target device toward which the implementation is to be targetted (which defines the available resource).

3.3 Data Topological Manipulation

Dataflow decrees that DFG edges are traversed by data tokens, which are atomic units composed of multiple data words. Data Topological Manipulation describes a class of transformations which explore the design space trading-off token and data-word characteristics.

In the study of a multi-channel, variable point-size FFT operator in [13], a new type of dataflow optimisation has arisen. The FFT, which processes two streams of input data, which has a dynamic range as shown in Figure 8. The size of each data word is determined by the worst case dynamic range, however as Figure 8 shows, the majority of the data is within a dynamic range of less than 50% of the maximum. As such, in the majority of cases, the data words processed by the core are oversized and do not contain enough valid information to justify the defined wordsize for the majority of samples. Given the fixed-point arithmetic exploited in this design, it is possible then to collapse multiple of the input data streams into a single data word (Figure 9), enabling a new kind of sub-token design space where the dimensions of the tokens are traded-off with the dataword capacity.

The result of this optimisation in the context of this FFT system design are stark, and given in Table 3. As this shows, despite modest resource savings (around 12.5% in

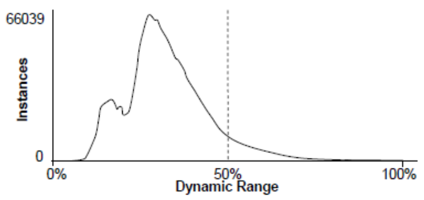


Figure 8: FFT Input Data Dynamic Range

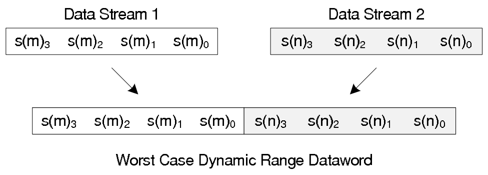

Data Stream 1
s(m)3 s(m)2 s(m)1 s(m)0
Data Stream 2
s(n)3 s(n)2 s(n)1 s(n)0
s(m)3 s(m)2 s(m)1 s(m)0 s(n)3 s(n)2 s(n)1 s(n)0
Worst Case Dynamic Range Dataword

Figure 9: FFT Data Word Packing

programmable logic cost and 50% in multiplier cost) between the dual channel original version (2FFT) and the compressed new core (FFT2), the real gains are to be found in power consumption reduction, with highly impressive 55.6% and 50.6% reduction in power consumption for two sample radar datasets (PC1 and PC2 respectively).

	LUTs	**Mults**	PC1 (mW)	PC2 (mW)	**Throughput** (MSamples/S)
2FFT	2780	18	916	873	1600
FFT2	2421	9	407	431	1600

Table 3: FFT Power Consumption Synthesis Results

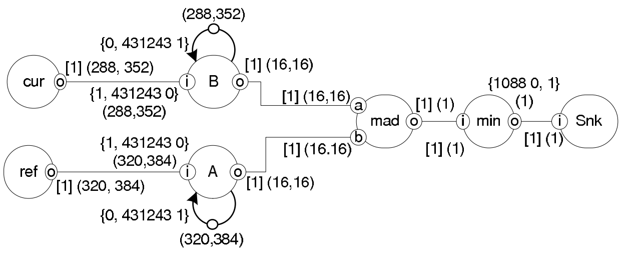
The value of this kind of exploration for power consumption optimisation is, as such, obvious. However, again the applicability of this transformation and it's effect is highly dependent on the particular nature of the environment in which the FFT operator resides - from the viewpoints of the fact that the underutilisation of the dynamic range of the dataword and the fact that the dynamic range is sufficiently small to enable efficient fixed-point arithmetic to be used in the first place.

3.4 Semantic Manipulation

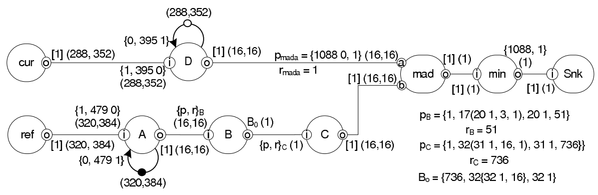
Dataflow modelling domains such as CSDF [6] and CSDF-SC [7] are specialities of the simpler dataflow domains since they manipulate the firing semantics of an actor and the token production/consumption characteristics applicable to those semantics. For instance, CSDF generalises the single firing rule semantics and constant integer thresholds of SDF actors to cyclically variable firing rules and thresholds. This enables cyclically changing behaviour to be included in DFGs without sacrificing the advantageous compile-time properties of SDF [6]. The Semantic Manipulation class describes a class of transforms which manipulate these characteristics for FPGA implementation optimisation and these are widely exploited in approaches such as Compaan and Mor.This semantic manipulation may be applied, for instance, to reduce communications bandwidth by promoting data reuse or manipulate the nature of the communications bandwidth (e.g. bulk or small data transfers) [14].

As an illustration of the application of this technique to FPGA architectures, consider the design of a ME actor for image processing appications (see Section 3). The relatively large token dimensions in this situation, encapsulating entire video frames, means that despite the relatively low token processing rate (30 frames/s) that high communications bandwidth is required in tandem with relatively large storage capacity.

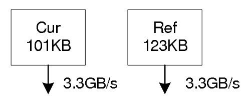
From the starting point of the ME block diagram, a CSDF specification can be constructed, as in Figure 10(a). The major problem with implementing this model is that the combination of high communications bandwidth and large size of the storage required for the current and reference frames (Figure 10(c)) to be supplied to actors *A* and *B* in Figure 10(a). This results in the situation where implementation as-is is impossible since the use of large off-chip storage to meet the memory size demands cannot meet the high bandwidth requirements [12]. By refinements of this model into the CSDF-SC domain [7], to enable data reuse and consumption without removal of the token from the incoming FIFO, this DFG may be refined to that of Figure 10(b). This has an accompanying memory hierarchy as given in Figure 10(d), which is implementable on a standard FPGA platform.



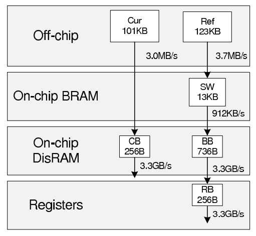
(a) ME CSDF Specification



(b) ME CSDF-SC Specification



(c) ME CSDF Memory Requirements



(d) ME CSDF-SC Memory Requirements

Figure 10: ME Memory Requirements

This kind of optimisation has a profound impact on the communications mechanisms in the implementation, reducing off-chip bandwdith requirements by almost three orders of magnitude, from 3.3.GB/s to less than 4 MB/s, whilst keeping smaller, higher bandwdith portions of the algorithm closer to the datapath, making a previously infeasible implementation possible. Similarly the interleaved/block processing variability implemented by Mor in the NLF and FxBF examples ([9] and [8] respectively) fall into this domain of transformation.

Again, however, such manipulation is only required since the off-chip memory in this case cannot support the required bandwidth - had such a facility been available such manipulation would not be required to the same extent, if at all. The ability to integrate this kind of target-specific information, in a parameterised manner, into a dataflow model is an important step would be an important enabling step toward generating the implementation. Again, however, such a facility in the algorithm modelling domain of interest, is not available.

3.5 3D Constrained Algorithm Manipulation for FPGA Synthesis

Sections 3.2-3.4 have shown there are general similarities between different specific transforms applied in very different applications, to achieve very different performance gains on synthesis to FPGA implementation. However, the specific nature of the transform are driven by the operatinf context of the DFG in every case, in terms of both the operating environment in which the algorithm finds itself, and the platform to which it is to be targetted. There is no standard semantic or syntax by which the kinds of operating environment and target platform information required in these design processes can be included in, or complementary to the dataflow models describing the algorithm. The emergence of such models is critical if dataflow languages are to become the backbone of FPGA DSP system design tools. This process thus requires a three dimensional design process manipulating the vertex and data toplogy of the algorithm, as well as it's semantics, driven by the operating environment and target platform.

The requirement to consider the operating environment and target platform, as well as the actual algorithm behaviour becomes even more important in the context of systems operating in dynamic operating environements. For instance, Multiple-Input Multiple-Output (MIMO) receivers operating on dynamic communications channels, with varying levels of signal distortion, must be able to adapt the processing of the system on the fly based on characteristics of the channel. Filters must vary in length, decoders must search larger received signal spaces etc. As such, in this case not only the operational efficiency of the system ,but the effective operation itself, is dependent on the ability to integrate environmental information into the algorithm definition and adapt the algorithm according to this information.

4. SUMMARY

Dataflow design languages have proven highly effective and modelling DSP and image processing systems with a view to rapid and efficient implementation. However, work on extending this work to FPGA architectures is immature and adhoc.

In this paper, via analysis of four case-study designs, it has been shown that current seemingly ad-hoc algorithm optimisation approaches all fall broadly into three classes: *vertex topological*, *data topological* or *semantic*. Further, these are all motivated by two key aspects not considered in the algorithm description: the operating environment, and the target platform.

This discrepancy becomes even more apparent in the modern generation of applications, such as DSP for mobile communications, whose behaviour can fluctuate wildly with their dynamic operating environments. This discrepancy must be addressed if dataflow languages are to form the backbone of design tools for dynamic FPGA systems.

REFERENCES

[1] R. Woods and J. McAllister and G. Lightbody and Y. Yi, *FPGA-Based Implementation of Signal Processing Systems*. Address: Wiley, 2008.

[2] S. S. Bhattacharyya and P.K. Murthy and E.A.Lee, *Software Synthesis from Dataflow Graphs*. Address: Kluwer, 1996.

[3] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow,” *Proc. IEEE*, vol. 75, pp. 1235–1245, Sept. 1987.

[4] E. A. Lee and T. M. Parks, “Dataflow Process Networks,” *Proc. IEEE*, vol. 83, no. 5, pp. 773–799, May 1995.

[5] P. K. Murthy and E. A. Lee, “Multidimensional Synchronous Dataflow,” *Proc. IEEE*, vol. 50, no. 8, pp. 2064–2079, August 2002.

[6] G. Bilsen and M. Engels and R. Lauwereins and J. Peperstraete, “Cyclo-Static Dataflow,” *IEEE. Trans Signal Processing*, vol. 44, no. 2, pp. 397–408, Feb. 1996.

[7] K. Denolf and M. Bekooij and J. Cockx and D. Verkest and H. Corporaal, “Exploiting the Expressiveness of Cyclo-Static Dataflow to Model Multimedia Implementations,” *EURASIP Journal on Advances in Signal Processing*, vol. 2007, 2007.

[8] J. McAllister and R. Woods and S. Fischaber and E. Malins, “Rapid Implementation and Optimisation of DSP Systems on FPGA-centric Heterogeneous Platforms,” *Journal of Systems Architecture*, vol. 53, no. 8, pp. 511–523, 207.

[9] J. McAllister and R. Woods and R. Walke and D. Reilly “Multidimensional DSP Core Synthesis for FPGA,” *Journal of VLSI Signal Processing Systems*, vol. 43, no. 2-3 pp. 207–221, June 2006.

[10] T. Stefanov and C. Zissulescu and A. Turjan and B. Kienhuis and E.Deprettere, “System Design Using Kahn Process Networks: The Compaan/Laura Approach,” in *Proc. Design, Automation and Test Europe (DATE)*, February 2004.

[11] H. Nikolov and T. Stefanov and E.Deprettere, “Multiprocessor System Design with ESPAM,” in *Proc. 4th International Conference on Hardware/software Codesign and System Synthesis*, 2004, pp. 211–216.

[12] S. Fischaber and R. Woods and J.McAllister, “SOC Memory Hierarchy Derivation from Dataflow Graphs,” in *Proc. 2007 IEEE Workshop on Signal Processing Systems*, Oct. 2007, pp. 469–474.

[13] S. McKeown and R. Woods and J.McAllister, “Power efficient dynamic-range utilisation for DSP on FPGA,” in *Proc. 2008 IEEE Workshop on Signal Processing Systems*, Oct. 2008, pp. 233–238.

[14] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. Address: CRC Press, 2009.

[15] P. Pop and P. Eles and Z. Peng *Analysis and Synthesis of Distributed Real-Time Embedded Systems*. Address: Springer, 2004.