Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 395018, 11 pages
doi:10.1155/2009/395018

Research Article

A Taxonomy of Reconfigurable Single-/Multiprocessor Systems-on-Chip

Diana Göhringer,[1] Thomas Perschke,[1] Michael Hübner,[2] and Jürgen Becker[2]

[1] *FGAN-FOM, Research Institute for Optronics and Pattern Recognition, 76275 Ettlingen, Germany*
[2] *Fakultät für Elektrotechnik und Informationstechnik, Universität Karlsruhe, Institut für Technik der Informationsverarbeitung (ITIV), 76131 Karlsruhe, Germany*

Correspondence should be addressed to Diana Göhringer, dgoehringer@fom.fgan.de

Received 19 December 2008; Accepted 12 May 2009

Recommended by Gilles Sassatelli

Runtime adaptivity of hardware in processor architectures is a novel trend, which is under investigation in a variety of research labs all over the world. The runtime exchange of modules, implemented on a reconfigurable hardware, affects the instruction flow (e.g., in reconfigurable instruction set processors) or the data flow, which has a strong impact on the performance of an application. Furthermore, the choice of a certain processor architecture related to the class of target applications is a crucial point in application development. A simple example is the domain of high-performance computing applications found in meteorology or high-energy physics, where vector processors are the optimal choice. A classification scheme for computer systems was provided in 1966 by Flynn where single/multiple data and instruction streams were combined to four types of architectures. This classification is now used as a foundation for an extended classification scheme including runtime adaptivity as further degree of freedom for processor architecture design. The developed scheme is validated by a multiprocessor system implemented on reconfigurable hardware as well as by a classification of existing static and reconfigurable processor systems.

1. Introduction

The usage of Multiprocessor System-on-Chip (MPSoC) for accelerating performance intensive applications is an upcoming trend in current chip technology. The approach of distributing tasks of an application to several processing units is a well-established method for increasing computational performance in computer science. In 1958, the IBM researchers Cocke and Slotnick discussed the usage of parallelism in numerical calculations for the first time. Four years later the first approach with four discrete parallel working processors is documented in [1]. In this first multiprocessor system, discrete chips were connected on a printed circuit board in a certain topology. Nowadays the System-on-Chip approach allows for integration of multiple devices on one chip die. Many difficulties in hardware technology were resolved. However, achieving a well-balanced workload using an optimized partitioning of the application tasks to the different processor cores remains an unsolved challenge. The partitioning algorithms and methodologies are able to detect inherent parallelism in a dataflow graph (DFG), and a mapping tool is able to distribute the tasks to a certain processing unit. The gap in this procedure is that the tasks can only be optimized to a certain extent to a given (multi)processor architecture. A certain granularity or requirement of a task cannot be further optimized. Exactly this is the problem whereof static multiprocessor hardware architecture suffers from. Changing applications and requirements do not reach an optimized work balance of the processors and the communication infrastructure. A promising approach for bridging this gap is to exploit runtime reconfigurable hardware. The requirements of an application's task can be fulfilled by a flexible hardware, which adapts in terms of computational performance and communication bandwidth.

Therefore, several academic labs all over the world are investigating the use of reconfigurable hardware in single- and multiprocessor systems-on-chip (MPSoC) (e.g., [2–4]). The hardware adaptivity offers a greater flexibility, which is not only beneficial in the global view of an MPSoC, but

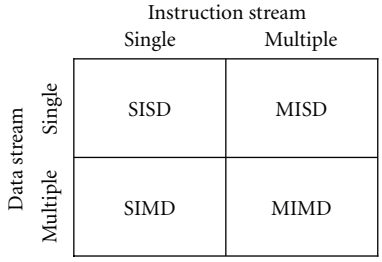

Figure 1: Flynn's taxonomy (1966) [7].

also in the special case of a single processor. This way, a static single processor optimized for a special application (Application Specific Instruction Set Processor (ASIP)) can be transformed into a RISP (Reconfigurable Instruction Set Processor) allowing for optimizations at runtime for a range of applications. Many of the novel reconfigurable MPSoCs consist of several RISPs (e.g., [5, 6]).

The most used classification scheme until today for single-/multiprocessor systems is Flynn's taxonomy [7], which uses four classes separating them in respect to single-/multiple data and instruction streams. Based on this classification scheme a new taxonomy is proposed, where runtime reconfiguration is included to further separate between static and dynamic processor systems, but also to discriminate between the different kinds of runtime reconfigurability (data and instructions).

This paper describes the new taxonomy and validates it by classifying several existing static and dynamic single- and multiprocessor systems. Furthermore, it is validated by a runtime adaptive multiprocessor system-on-chip (RAMPSoC) [6].

The paper is organized as follows: the state of the art will be presented in Section 2. Section 3 describes the new taxonomy for reconfigurable single-/multiprocessor systems-on-chip. Examples for each class of the new taxonomy are given in Section 4. Section 5 describes the RAMPSoC architecture, which forms the superclass of the new taxonomy. Finally the paper is closed by presenting the conclusions and future work in Section 6.

2. State of the Art

Until today, the most known classification scheme for single- and multiprocessor systems is the taxonomy of Flynn introduced in 1966, which is shown in Figure 1.

This taxonomy groups processor systems into four classes depending on their support for single/multiple data and/or instructions streams. At the time, when Flynn developed his taxonomy, only static processor systems existed. Due to their static architecture, they force the user to follow a top-down designflow, in which the application integration has to follow the given hardware architecture. Especially for multiprocessor systems this is a major drawback, because a given application can only be optimized to a certain extend to the static hardware architecture. As the processors and their communication infrastructure are fixed only few applications, for which the architecture was optimized, achieve a well balanced workload. This drawback exists not only at design-time, but also at runtime, as the hardware cannot be adapted on-demand in respect to the application requirements.

Looking at state-of-the-art processor architectures a novel trend toward using runtime reconfigurable hardware for single- and multiprocessor systems can be observed [8]. The systems using runtime reconfigurability can overcome the major drawback of static processor systems, because they can be adapted to the requirements of the application. As these adaptive systems differ in respect to their degree of reconfigurability, they have to be classified. The separation within Flynn's taxonomy is very coarse and does not account for the new degree of flexibility and adaptivity of these novel hardware systems.

There exist previous approaches to classify reconfigurable systems. Sanchez et al. [9] proposed a very coarse classification by dividing the systems into static and dynamic configurable systems. Following their terminology, static configurable systems are configured by an external processor once before executing a task. Dynamic configurable systems can configure themselves to adapt to a changing environment. This classification approach is too coarse for classifying state-of-the-art single- and multiprocessor systems. Also only FPGA-based architectures are taken into account.

Page [10] proposes possible categories, which can be used to develop a taxonomy for reconfigurable processor architectures. These reconfigurable processor architectures are nowadays called RISPs and consist of a single processor and a reconfigurable array. Therefore, his classification is very useful for such architectures. It is also very fine grained, as it classifies these systems depending on their interconnection, on the used memory architecture and memory operation and also on the models used for programming the reconfigurable array. A major drawback is that it does not take into account reconfigurable MPSoC systems or static single- and multiprocessor systems.

Also, Hartenstein [11] describes a classification for RISPs processors, but he only takes into account coarse-grained reconfigurable architectures. He classifies these architectures according to their architecture (mesh-, linear array-, or crossbar-based). Each architecture class is further divided depending on the used granularity, fabrics, mapping, and intended target applications. This results in the same major drawback than described above for [10].

Additionally, Sima et al. [12] describe a taxonomy to classify RISPs, also named Field-Programmable Custom Computing Machines (FCCMs). In contrary to [10, 11], they classify these systems only on architectural criteria and not on implementation details. This results in a coarser taxonomy consisting of four classes. Also here the major drawback is that this taxonomy does not take into account reconfigurable MPSoC systems or static single- and multiprocessor systems.

Radunovic and Milutinovic [13] propose the "Olymp" classification for reconfigurable systems. In a first step, they divide these systems depending on their aim of reconfiguration (increasing performance or fault tolerance). Further categories are the granularity of the reconfigurable

device (fine-, medium, or coarse-grained), the integration (dynamic, static closely coupled, static loosely coupled), and, finally, if the external network is static or reconfigurable. For the integration category, they use the same separation as proposed in [9]. This means, dynamic systems are stand-alone systems, and static systems consist of a host processor that manages the configuration of the reconfigurable array. Out of this, 15 classes are generated. Each of them is named after a Greek god. For six of these classes, examples of existing systems are given. These systems are classified according to their dominating characteristic. The advantage of this classification is that it supports a fine-grained classification for reconfigurable computing systems, which use reconfigurability to achieve a better performance. A disadvantage is that all reconfigurable systems designed for fault-tolerance reasons are grouped together and are not finer classified. An additional drawback is that there is no separation made between static and reconfigurable systems, and also there is no separation between single and multiple data and/or instruction streams and therefore no separation between single- and multiprocessor systems is done by this classification. Finally, using names of Greek gods in such a classification makes it also difficult to remember, which Greek god resembles which class of systems. The naming convention used by the taxonomy of Flynn is straighter forward.

In summary, the major drawback of all these classification approaches is that none of them classify static and reconfigurable single- and multiprocessor systems. Therefore, a new taxonomy, supporting a finer classification of static and dynamic single- and multiprocessor systems, is required and will be described in detail in the next section.

3. The Taxonomy Extension to Flynn

Nowadays, while more and more different implementations of reconfigurable processors and reconfigurable MPSoCs are introduced, Flynn's taxonomy is not sufficient anymore, as it does not differentiate between reconfigurable and static processor systems. Also, as mentioned before, a finer separation with respect to the type of reconfigurability is required. Therefore a new taxonomy for the classification of reconfigurable Single-/Multiprocessor Systems-on-Chip is necessary. These reconfigurable systems differ from each other not only with respect to single or multiple data and instruction streams, but also with respect to the grade of reconfigurability, which they support. This results in the new taxonomy shown in Figure 2.

The form of the classification scheme is based on a Karnaugh diagram with four variables. First, the systems are divided into single (SI) and multiple instruction (MI) stream systems. Second, they are divided further into single (SD) and multiple data (MD) stream systems. These divisions were adopted from Flynn. Furthermore, the systems are separated in static and reconfigurable instruction (RI) stream systems. Reconfigurable instruction stream systems can exchange either their instruction memory or their instruction set, parts of which can also be implemented in

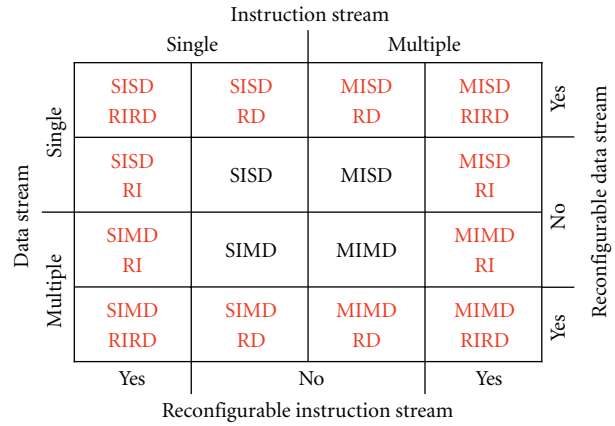

Figure 2: The new taxonomy.

reconfigurable hardware accelerators. Finally, the systems are divided with respect to static and reconfigurable data (RD) streams. Processor systems with reconfigurable data streams can either exchange their data memory or modify the data paths or both. As can be seen in Figure 2, the four classes in the middle define the static processor systems known from Flynn's taxonomy. The reconfigurable classes are grouped around them.

For this taxonomy the following terminology is used: a system with an additional degree of freedom means that this system offers one additional degree of parallelism and therefore has a greater flexibility. The parallelism of a system is divided into spatial and temporal parallelism. Spatial parallelism is used to distinguish between systems with single or multiple streams. Here temporal parallelism is used to differentiate static and reconfigurable systems. Reconfigurable systems have one or two degrees of temporal parallelism, depending, if only one or both, instruction and data streams are reconfigurable. Static systems, therefore, do not have a temporal parallelism following to this terminology. Of course, from the view of a computer architect, processor system with pipelined architectures, like vector processors, also have a temporal parallelism. In this terminology this kind of temporal parallelism is low level compared to the temporal parallelism offered by reconfigurability. Therefore, it will not be considered.

Using this terminology a hierarchy between the classes exists depending on their degree of parallelism. This way, some classes are a subclass of others. This hierarchy can be seen in Figure 3. Classes on the same level offer the same degree of freedom. The difference between each of the 5 levels is one degree of freedom.

This means that SISD is the lowest class by offering the lowest flexibility. In other words, each of the other classes can be reduced down to SISD by removing its spatial and/or temporal parallelism. By adding one degree of parallelism to the SISD architecture, either in respect to supporting multiple data or instruction streams, or by adding the reconfigurability of either the instructions or the data, the next level is reached. Therefore, this level consists of the following classes: SIMD, SISD_RI, SISD_RD, and MISD. By

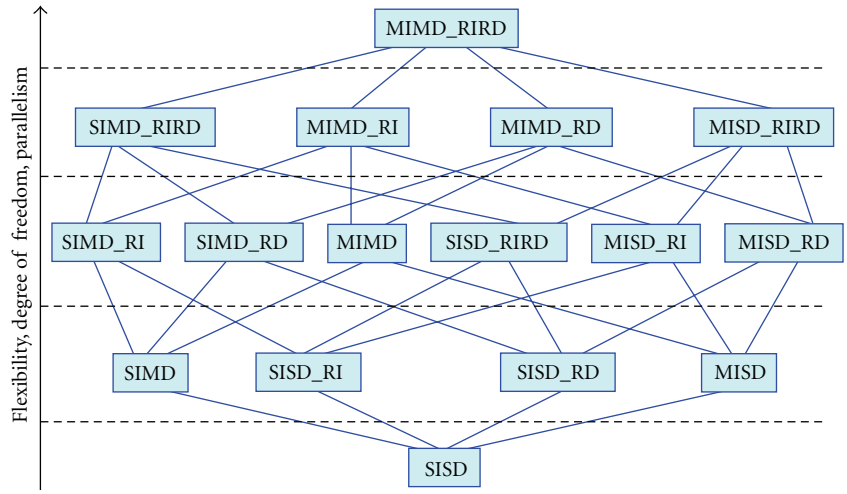

Figure 3: Hierarchy of the classes depending on their degree of freedom, their flexibility, and parallelism. The neighborhood relations are shown by the connections between the classes.

adding more flexibility the next level is reached and so on until finally the highest level is reached. In this level only the MIMD_RIRD class is present. This class supports the highest degree of freedom and therefore is the superclass of the whole taxonomy. MIMD_RIRD resembles a reconfigurable multiple instruction and reconfigurable multiple data stream architecture. To the best of our knowledge, so far only one system combines this flexibility. This system is a runtime adaptive MPSoC called RAMPSoC, which will be presented in Section 5.

4. Examples for the Different Classes

In the following subsections each class will be described. Several representative systems will be referenced for each class. Of course, for most classes more systems exist than can be mentioned here.

4.1. SISD. SISD processors are usually based on von Neumann or Harvard architectures. Examples are processors based on older x86 architectures, like the Intel 80486 processors. Most microcontrollers used in embedded systems fall into this category. Examples are the PowerPC405 cores [14] used on Xilinx Virtex-II Pro and Virtex-4 FPGAs and soft-core processors like Xilinx MicroBlaze [15] or Leon [16].

4.2. MISD. Systems with this architecture are very rare. In [7] a pipelined structure is presented as a MISD architecture. There, a number of separate execution units with their own instruction code act on the data processed by the preceding execution unit. Examples for such a scheme are streaming architectures for image processing used on FPGAs. It is also possible to include redundant computer systems like the one used in the space shuttles [17] in this category. Additionally, MISD architectures are used for pattern matching [18], because they support to simultaneously search for multiple patterns within a single data stream.

4.3. SIMD. This class can be further divided in array and vector processor-based classes. Generally speaking, array processors use parallelization in space while vector processors use low level parallelization in time to gain processing power.

Array processors consist of an array of regularly connected identical execution units. At each cycle all units execute the same instruction on different data. This architecture was used in older systems, like the Connection machine CM-2. Systolic arrays can be included in this category. Array processors are also used in embedded systems to increase the performance compared with SISD architectures. Especially, in the image processing domain (e.g., [19–21]) they are used to increase the overall performance, for example, for matrix multiplications.

Vector processors, for example the NEC SX-6 [22] processor, use vector pipelines on vectors of floating point numbers. The execution unit is deeply pipelined. Vector pipelines with the same or different functionality can be chained to form vector units. Vector processors also include scalar units.

4.4. MIMD. The MIMD class is usually divided in different subclasses, as too many different architectures fall into this category.

Memory coupled processor systems, like multicore CPUs, can be grouped depending on the type of memory access in UMA (uniform memory access), NUMA (non-uniform memory access), and COMA (cache only memory access) systems.

Processor systems based on message passing can be grouped in MPP (massive parallel processing) systems, like the Intel ASCI-Red [23] and COW (cluster of workstations).

Superscalar processors, like the recent x86 architectures, VLIW (Very Large Instruction Word) architectures mostly used in DSP processors and the similar EPIC (Explicitly parallel instruction computing) architectures used in the Intel Itanium [24] processors can be included in the MIMD class.

An additional example for the MIMD category is the Cell processor [25], which consists of one power processor element (PPE) and eight synergetic processor elements (SPEs). While each single PPE or SPE has an SIMD architecture, the overall system belongs to the MIMD architecture.

4.5. SISD_RI. SISD_RI is similar to SISD with a parallelism in the time domain. This means, that the instruction stream can be reconfigured at runtime. The best known systems that fall into this class are Reconfigurable Instruction Set Processors (RISPs) [26]. These processors can adapt their instruction set on-demand. They consist of a processor with an attached reconfigurable block. Today, lots of research are done in this field, and there exist several different types of RISPs (such as [2, 3, 27]). They differ from each other with respect to their processor architecture, the reconfigurable block (coarse, fine grained), how the reconfigurable block is connected to the processor (coprocessor, tightly-coupled or loosely coupled on-processor), the reconfiguration method (partial reconfiguration, context switch ...), and so forth.

4.6. SISD_RD. These systems have a parallelism in the time domain. The difference to SISD_RI systems is that here the instruction stream remains static, but the data stream is reconfigurable. Such systems are SISD processors which can exchange their data memory or the data path to connect to a different data source or both. The data memory can be exchanged by using dynamic and partial reconfiguration as it is done in [28, 29]. There also exist reconfigurable communication infrastructures [30], which can be used to adapt the data path at runtime.

4.7. SISD_RIRD. This is a combination of the SISD_RI and the SISD_RD systems described before. Here, two degrees of flexibility are added, as now the instruction stream as well as the data stream can be reconfigured at runtime. A representative system for this class is one single processor in RAMPSoC [6], because the exchange of both instruction and data memory is supported, and a reconfigurable communication structure is used between the processing elements as well as with the environment.

4.8. SIMD_RI. These systems are basically SIMD systems with an extended parallelism in the time domain. These systems can reconfigure their instruction stream at runtime, by reconfiguring, for example, their processing elements/ hardware accelerators or their instruction memory. A representative system is the Montium Tile Processor from Recore Systems [31], which has a reconfigurable instruction set. Also, systems consisting of a processor with a loosely coupled reconfigurable hardware accelerating the instructions of the processor belong to the SIMD_RI architecture, if both can access independently the data memory and if the data path outside the reconfigurable hardware remains static. In this case the reconfigurable hardware is an accelerator and cannot work without the processor. Therefore, from the abstract system view, only a processor with an attached accelerator is seen. Reconfigurable data paths or processing elements within the accelerator itself are not taken into account when classifying these systems. Such a system is, for example, MorphoSys [32]. Also, the MORPHEUS platform [33] consisting of one ARM9 processor connected to three heterogeneous reconfigurable accelerators belongs to this class. In [34] three additional systems falling into this category are described: DReAM (a coarse-grain Dynamically Reconfigurable Architecture for Mobile communication systems), Triscend A7, and XPP (eXtreme Processing Platform).

4.9. SIMD_RD. These systems are SIMD which can exchange their data memory or reconfigure their communication infrastructure [35].

4.10. SIMD_RIRD. SIMD_RIRD systems combine the two different kinds of parallelisms in the time domain of the SIMD_RI and SIMD_RD systems. Therefore, these systems can configure both their data stream and their instruction stream. An example of such a system would be a single processor in RAMPSoC with several reconfigurable accelerators that compute the same instruction on different data streams. Additional to the accelerators also the data and instruction memory as well as the communication infrastructure support runtime reconfiguration. Furthermore, the Maia architecture [34], consisting of an ARM processor connected over a reconfigurable network to an FPGA accelerator as well as several embedded memories, MACs and ALUs, belongs to this class. In this system the network (data stream) and the FPGA accelerator (instruction stream) are reconfigurable. A similar system is the Pleiades architecture [36] consisting of a microprocessor, which is connected over a reconfigurable interconnect to several reconfigurable accelerators called satellites.

4.11. MISD_RI. These systems extend the static MISD architecture by having multiple reconfigurable instruction streams. These could be redundant systems consisting of several processors that can reconfigure their instruction memories or their processing elements/hardware accelerators.

An additional example would be a group of SISD_RI processors, which compute multitarget image processing algorithms, where each processor is searching for a different object in the same data. An additional example would be a streaming architecture for image processing with reconfigurable instruction streams.

4.12. MISD_RD. These systems are MISD systems with an additional parallelism in the time domain. As opposed to the MISD_RI systems here the instruction stream is static, but the data stream is reconfigurable. These systems support the runtime exchange of the data memory or the communication infrastructure. An example for a MISD_RD system is a group of SISD_RD processors, which are connected to the same data source.

4.13. MISD_RIRD. These are the most flexible of the MISD systems because they have reconfigurable instruction and data streams.

4.14. MIMD_RI. MIMD_RI systems are MPSoCs with reconfigurable instruction streams. Several research labs investigate this kind of systems. For example, Paulsson et al. [37] presented a system, which supports the reconfiguration of the instruction memories. Furthermore, Claus et al. [5] and Bobda et al. [4] developed MPSoCs with reconfigurable accelerators. Also, the XiRisc reconfigurable processor [38] is a representative system for this class. It consists of a VLIW RISC core with a runtime reconfigurable data path, called PiCoGA (Pipelined Configurable Gate-Array). By reconfiguring the PiCoGA, which is within the data path of the processor; the instruction set of the processor and therefore the instruction stream are reconfigurable. An additional example for such a VLIW processor with a runtime reconfigurable data path is the ADRES architecture [39].

4.15. MIMD_RD. These systems are MPSoCs with reconfigurable data streams. This means, they have either reconfigurable data memories or a reconfigurable communication infrastructure or both by using similar functionalities as described in the SISD_RD class.

4.16. MIMD_RIRD. RAMPSoC [6] includes all the above classes. To the best of our knowledge, this is the only architecture supporting such flexibility. Section 5 describes the advantages over static MPSoCs and the new designflow methodology used for RAMPSoC. In addition, the hardware system architecture is described in detail. Finally, different hierarchy layers are defined for the RAMPSoC hardware architecture, the software toolchain, and also the runtime reconfigurability.

5. RAMPSoC—the Superclass of the New Taxonomy

RAMPSoC represents the MIMD_RIRD class and is therefore the superclass of all classes within the new taxonomy. This means, it supports the highest degree of flexibility. So, by reducing this flexibility it can be reduced down to the other classes. For example, by removing the reconfigurability, RAMPSoC can be seen as an MIMD architecture. By further reducing the instruction stream to a single stream, it resembles a SIMD structure. Finally, by reducing the data stream from multiple to single, we get a SISD architecture, which resembles one single processor in RAMPSoC without the reconfiguration capabilities.

RAMPSoC was developed to overcome the deficiencies of static MIMD approaches. The drawbacks of static MIMD architectures are that they are not optimized for the tasks they have to execute. Therefore, they are not efficient with respect to performance and power consumption, as for most applications the workload between the processors is not well balanced. Furthermore, due to their high-power consumption, they are often not well suited for embedded systems.

Certainly, there also exist static MIMD architectures, which are optimized for a limited field of applications within an application domain. For these applications they achieve a very good performance in combination with low-power consumption. Their disadvantage is that they are only efficient for the applications, for which they are optimized. They cannot efficiently be reused for a different application domain, nor is their workload efficiently balanced for all applications of their application domain.

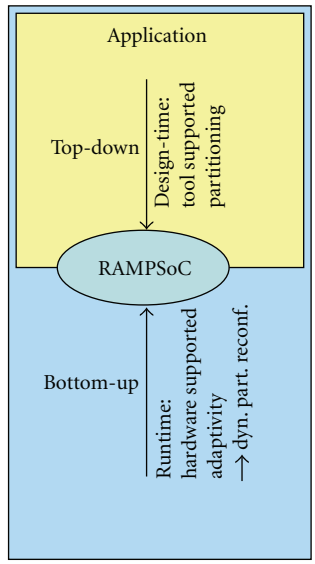

Figure 4: "Meet in the middle" Designflow approach for RAMPSoC.

This leads to the main drawback of all MIMD architectures, which is their top-down design flow. This means, the application integration has to follow the given hardware architecture of the MPSoC. Therefore, a given application can only be optimized to a certain extend to a static MPSoC architecture. This drawback exists not only at design time but also at runtime as the hardware cannot be adapted on-demand in respect to the application requirements.

Therefore, runtime reconfigurable hardware is used to overcome these deficiencies of static MIMD systems. The exploitation of runtime reconfiguration offers a new degree of freedom, because the hardware architecture can now be adapted to the application requirements. This adaptation is possible at design-time as well as at runtime allowing an optimized distribution of computing tasks. Therefore, constraints regarding the performance, the power consumption, or the used area can be achieved more efficiently. By combining the standard top-down designflow from static MIMD systems with the new bottom-up designflow the novel "meet in the middle" designflow for RAMPSoC is created. This designflow is illustrated in Figure 4. From the application perspective it offers the standard top-down designflow in terms of application partitioning and mapping. At the same time it supports from the hardware perspective the new bottom-up approach, enabling hardware adaptation at design-time as well as at runtime using dynamic and partial reconfiguration. At design-time the bottom-up approach is used to generate an initial RAMPSoC hardware architecture optimized for the actual application requirements. Additionally, the bottom-up approach supports the on-demand adaptation of the RAMPSoC system to the time variant application requirements.

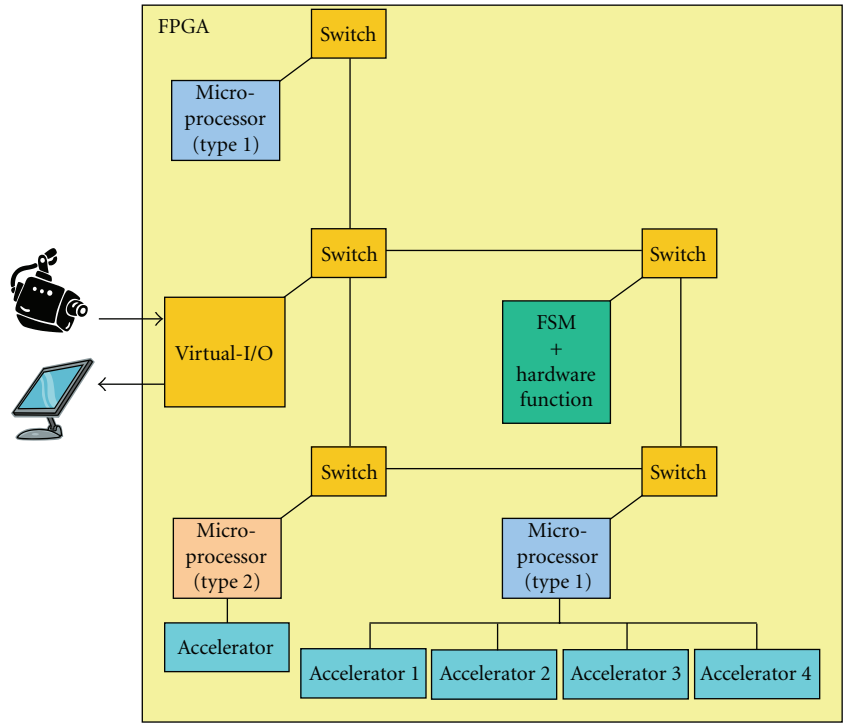

Figure 5: RAMPSoC system at one point in time.

Exactly this feature improves furthermore the functional density as discussed in [40, 41]. In contrast to the traditional and well-established methodologies of the Hardware/Software-Codesign (see [42]), the "meet in the middle" approach includes the option of a static or dynamic realization of hardware system parts. It therefore contains the "static/dynamic"-Codesign as a subpartition option of the hardware distribution. This option supports the development process at design phase, where a flexible hardware realization, based on elements from a hardware library, helps to improve the efficiency of the technology mapping process. At runtime the "meet in the middle" approach is used to provide the hardware elements on-demand. This is physically realized through dynamic and partial reconfiguration. In general, the "meet in the middle" approach targets to improve furthermore the benefits of the Hardware/Software-Codesign by the introduction of new design-time and runtime flexibility through the exploitation of reconfigurable hardware.

High performance within RAMPSoC can be achieved by exploiting the following three kinds of concurrency, which are inherently included in most applications. First, data parallelism is supported by distributing the data onto different processors, which execute the same (SIMD) or different (MIMD) instructions on their subset of data. Especially image processing algorithms lend themselves very well to data parallelism as an input image can be divided into independent tiles. Second, task parallelism is supported, as different processors or processor groups can execute different algorithms on the same (MISD) or different (MIMD) set of data. For example, this can be used in Multitarget Tracking algorithms, where each processor searches for a different target. And finally, instruction level parallelism is exploited by using processors with an instruction pipeline and by outsourcing complex instructions into one or several hardware accelerators connected to the processor. This provides an additional speedup. Here, also the benefit of the runtime reconfiguration is exploited since not all complex instructions have to be present at the same time. Instead, the hardware accelerator can be reconfigured with the required instruction by its corresponding processor on-demand at runtime.

5.1. Hardware System Architecture. Figure 5 shows the hardware system architecture of RAMPSoC at one point in time. As can be seen, RAMPSoC represents a modular, heterogeneous runtime adaptive MPSoC.

The processing elements in RAMPSoC can be different types of processors with an arbitrary number of accelerators. The processor types differ from each other for example, with respect to their bitwidth or their performance. Instead of a processor, it is possible to use a Finite State Machine (FSM) closely coupled with a hardware function.

Additionally, different types of communication infrastructures such as Network-on-Chip (NoC), Bus, or Point-to-Point connections, or a mixture of those are supported for interprocessor communication. In the example in Figure 5 a switch-based NoC with a mesh topology is used.

For the communication between RAMPSoC and its environment, such as a camera or a monitor, the Virtual-I/O component is used. The function of this component is to split the input data and forward it to one or several different processors. Additionally, it is responsible for collecting the results coming from the processors and sending them out

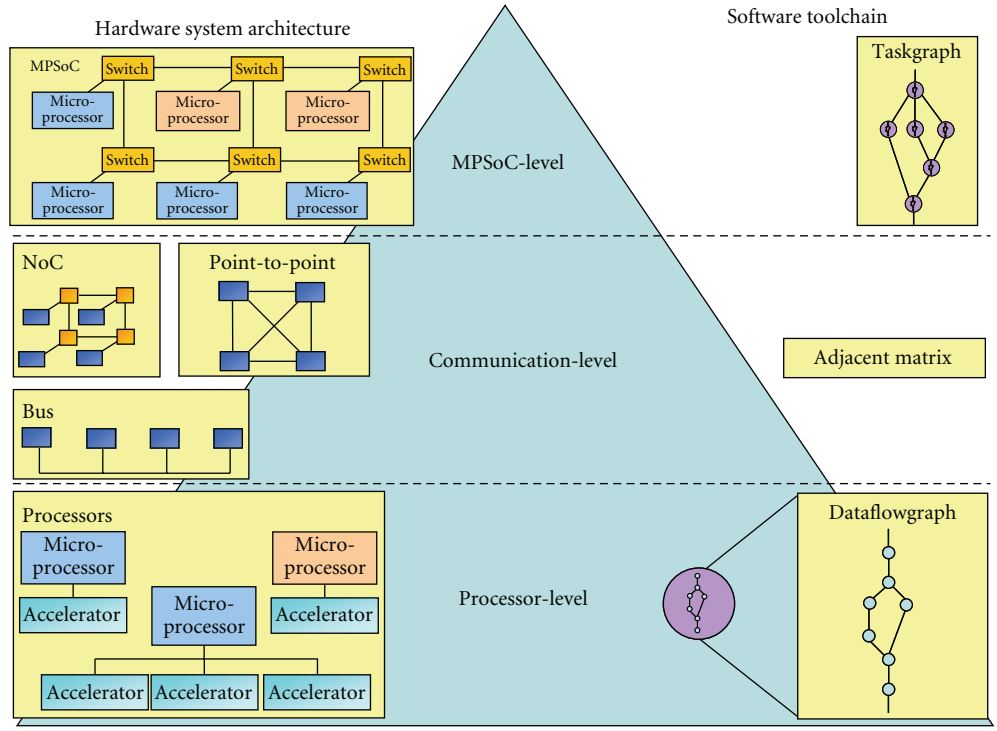

Figure 6: Three levels of hierarchy of RAMPSoC.

to the environment. The values to control the splitting of the input data as well as for the merging of the results are stored in several registers and can be modified at design-time as well as at runtime. The modification of these values can be done by overriding them either directly or by using dynamic and partial reconfiguration. Additional parts of the Virtual-I/O are two FSMs and several input and output buffers. If the number of processors in the system changes, the whole Virtual-I/O component can be modified, using dynamic partial reconfiguration. In this case, the loss of data is prevented by the data buffers.

Implementation results for three different RAMPSoC systems can be found in [43]. The systems differ from each other in respect to the number of used processors (1, 2, and 4). All use the Xilinx MicroBlaze processor, which can run at a maximum frequency of around 150 MHz. These systems were implemented to test the achievable speedup for the calculation of an image processing algorithm. Reconfiguration was not required by this algorithm, but can be easily extrapolated. As described in [43], these RAMPSoC systems were implemented on a Xilinx Virtex-4 FPGA. This FPGA family uses a 32-bit Hardware ICAP (Internal Configuration Access Port), which can run at 100 MHz. This results in a theoretical throughput of 400 Mbytes/s. Therefore, the reconfiguration of a Xilinx MicroBlaze Processor would take approximately 216 microseconds. The reconfiguration of the Virtual-IO component would require approximately 189 μs.

5.2. The Three Hierarchy-Levels of RAMPSoC. Within RAMPSoC, three virtualization layers exist forming the hierarchy-levels of RAMPSoC. These virtualization layers apply to the hardware system architecture as well as to the software toolchain as can be seen in Figure 6. Also, the runtime adaptation capabilities of the hardware using dynamic and partial reconfiguration are applied to the different layers.

The highest abstraction layer is the MPSoC-Level. Here, the exchange of whole processors is supported to fulfil quality of service parameters, such as achieving a higher performance or reducing the power consumption. Also, in this level a coarse partitioning of the application into several tasks is done. At the same time, a preliminary decision about the number and types of processors is made. Furthermore, as a result of the application partitioning the required communication infrastructure is defined.

The second abstraction layer is the Communication-Level. Here, the communication infrastructure, defined in the MPSoC-Level, is physically established. Using runtime reconfiguration the communication infrastructure can be modified by adapting the connections between the processors, for example, to change the bitwidth or the topology. Additionally, the routing algorithms can be adapted. To make this adaptation feasible and flexible the communication infrastructure is transferred to a processor-based abstraction level as shown in Figure 7.

The idea here is to see the communication infrastructure as a processor, which is distributed over the reconfigurable area. The picture shows exemplarily one possible scenario with a mixture of a NoC and a bus-based topology connecting several IP cores. The heterogeneous communication system makes sense, if, for example, a bus-based interconnect between cores is required due to a streaming data application,

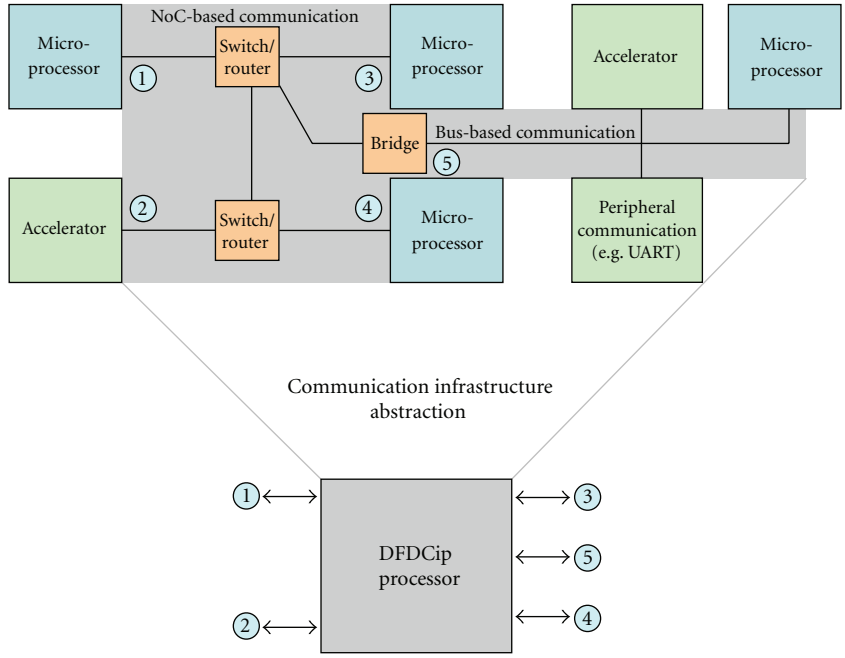

Figure 7: Communication infrastructure abstraction into a data flow dependent communication infrastructure processor (DFDCip).

which would block a NoC. Furthermore, a star or mesh topology enabled by a NoC has advantages for some applications in terms of performance and power consumption as described in [44, 45]. The idea, which is depicted in Figure 7, transforms the communication infrastructure into a module, which only has the required input and output channels connected to the respective IP cores. The bridge between the NoC and the bus system is also integrated in the module, which is called Data Flow Dependent Communication Infrastructure Processor (DFDCip). The DFDCip is a model for the communication infrastructure of the RAMPSoC approach, which can be handled as a processor providing communication interfaces on-demand. It is obvious, that this module has to be adaptive to a changing requirement of the dataflow of an application. Transferring the communication infrastructure of the MPSoC approach to a processor-based model has the advantage that a well-established programming paradigm can be used to adapt the communication hardware in relation to the dataflow of tasks running on the several IP cores. The example shows that also the DFDCip enables a new degree of freedom in terms for adaptivity by parameterization. Additionally, in case of a request for restructuring of the communication infrastructure, the components within the DFDCip block can be adapted using dynamic reconfiguration.

The third and lowest abstraction layer is the Processor-Level. Using dynamic and partial reconfiguration the instruction set of the processor can be modified by reconfiguring the accelerator. Additionally, the instruction as well as the data memory can be modified at runtime. Therefore, each processor within RAMPSoC represents a RISP and belongs to the SISD_RIRD class. Normally, RISPs only fulfil the requirements for the SISD_RI class, but the processors within RAMPSoC support the exchange of the data memory and therefore have a reconfigurable data stream, which classifies them for SISD_RIRD.

6. Conclusions and Outlook

This paper presents a new taxonomy for static and reconfigurable Single- and Multiprocessor Systems-on-Chip. This taxonomy extends the well-known taxonomy for very high-speed computers, which was proposed by Flynn in 1966. The new classification scheme is validated by referencing several representative systems for each class and extendable for novel architectures of the future. Finally, RAMPSoC is described as the representative systems for the superclass of the new taxonomy. This superclass supports the highest degree of freedom by using multiple reconfigurable instructions and data streams. The novel "meet in the middle" approach, which exploits the hardware adaptivity at design and runtime, enables to overcome the restrictions of traditional MPSoC architectures.

References

[1] A. Gary, "The Power of Parallelism," IEEE Computer World, November 2001, http://www.computerworld.com.

[2] T. Vogt and N. Wehn, "A reconfigurable application specific instruction set processor for viterbi and log-MAP decoding," in *Proceedings of IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS '06)*, pp. 142–147, Banff, Canada, October 2006.

[3] L. Bauer, M. Shaflque, and J. Henkel, "A computation- and communication-infrastructure for modular special instructions in a dynamically reconfigurable processor," in *Proceedings of the International Conference on Field Programmable*

Logic and Applications (FPL '08), pp. 203–208, Heidelberg, Germany, September 2008.

[4] C. Bobda, T. Haller, F. Mühlbauer, D. Rech, and S. Jung, "Design of adaptive multiprocessor on chip systems," in *Proceedings of the 20th Symposium on Integrated Circuits and System Design (SBCCI '07)*, pp. 177–183, Rio de Janeiro, Brazil, September 2007.

[5] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision—a run-time reconfigurable MPSoC architecture for future driver assistance systems," *Information Technology Journal*, vol. 49, no. 3, pp. 181–187, 2007.

[6] D. Göhringer, M. Hübner, V. Schatz, and J. Becker, "Runtime adaptive multi-processor system-on-chip: RAMPSoC," in *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS '08)*, Miami, Fla, USA, April 2008.

[7] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[8] "Cray XD1 Datasheet," http://www.cray.com.

[9] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, and A. Perez-Uribe, "Static and dynamic configurable systems," *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 556–564, 1999.

[10] I. Page, "Reconfigurable processor architectures," *Microprocessors & Microsystems*, vol. 20, pp. 185–196, 1996.

[11] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proceedings of Design, Automation and Test in Europe (DATE '01)*, pp. 642–649, Munich, Germany, March 2001.

[12] M. Sima, S. Vassiliadis, S. D. Cotofana, J. T. J. van Eijndhoven, and K. A. Vissers, "Field-programmable custom computing machines—a taxonomy," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '02)*, pp. 79–88, Montpellier, France, September 2002.

[13] B. Radunovic and V. Milutinovic, "A survey of reconfigurable computing architectures," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '98)*, pp. 376–385, Tallin, Estonia, 1998.

[14] "Xilinx PowerPC Processor Reference Guide," http://www.xilinx.com.

[15] "Xilinx MicroBlaze Reference Guide," http://www.xilinx.com.

[16] J. Gaisler, "The LEON Processor User's Manual," http://www.gaisler.com.

[17] A. Spector and D. Gifford, "The space shuttle primary computer system," *Communications of the ACM*, vol. 27, no. 9, pp. 872–900, 1984.

[18] A. Halaas, B. Svingen, M. Nedland, P. Saetrom, O. Snove Jr., and O. R. Birkeland, "A recursive MISD architecture for pattern matching," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 7, pp. 727–734, 2004.

[19] R. L. Rosas, A. de Luca, and F. B. Santillan, "SIMD architecture for image segmentation using sobel operators implemented in FPGA technology," in *Proceedings of the 2nd International Conference on Electrical and Electronics Engineering ((ICEEE '05)*, pp. 77–80, September 2005.

[20] P. Bonnot, F. Lemonnier, G. Edelin, G. Gaillat, O. Ruch, and P. Gauget, "Definition and SIMD implementation of a multi-processing architecture approach on FPGA," in *Proceedings of Design, Automation and Test in Europe (DATE '08)*, pp. 610–615, Munich, Germany, March 2008.

[21] NEC Electronics Europe, "IMAPCAR—the solution to automotive embedded image processing," http://www.eu.necel.com.

[22] "NEC SX-6 Series," http://www.nec.co.jp/press/en/0110/0301.html.

[23] "ASCI-Red," http://www.sandia.gov/ASCI/Red.

[24] "Intel Itanium Processor 9100 Series Product Brief," http://www.intel.com.

[25] D. Pham, S. Asano, M. Bolliger, et al., "The design and implementation of a first-generation CELL processor," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '05)*, vol. 48, pp. 184–592, San Francisco, Calif, USA, February 2005.

[26] F. Barat, R. Lauwereins, and G. Deconinck, "Reconfigurable instruction set processors from a hardware/software perspective," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 847–862, 2002.

[27] A. Lodi, L. Ciccarelli, C. Mucci, R. Giansante, and A. Cappelli, "An embedded reconfigurable datapath for SoC," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 303–304, Napa Valley, Calif, USA, April 2005.

[28] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: using FPGA configuration circuitry to emulate a network-on-chip," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 257–262, Heidelberg, Germany, September 2008.

[29] O. Sander, L. Braun, and J. Becker, "An exploitation of data reallocation by performing internal FPGA self-reconfiguration mechanisms," in *Proceedings of the Reconfigurable Computing: Architectures, Tools and Applications (ARC '08)*, pp. 312–317, London, UK, March 2008.

[30] L. Braun, D. Göhringer, T. Perschke, V. Schatz, M. Hübner, and J. Becker, "Adaptive real time image processing exploiting two dimensional reconfigurable architecture," *Journal of Real-Time Image Processing, Springer*, vol. 4, no. 2, pp. 109–125, 2009.

[31] L. T. Smit, G. K. Rauwerda, A. Molderink, P. T. Wolkotte, and G. J. M. Smit, "Implementation of a 2-D 8×8 IDCT on the reconfigurable Montium core," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 562–566, Amsterdam, The Netherlands, August 2007.

[32] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[33] A. Deledda, C. Mucci, A. Vitkovski, et al., "Design of a HW/SW communication infrastructure for a heterogeneous reconfigurable processor," in *Proceedings of Design, Automation and Test in Europe (DATE '08)*, pp. 1352–1357, Munich, Germany, March 2008.

[34] J. Becker, "Configurable systems-on-chip: commercial and academic approaches," in *Proceedings of the IEEE 9th International Conference on Electronics, Circuits and Systems (ICECS '02)*, vol. 2, pp. 809–812, September 2002, Dubrovnik, Croatia.

[35] H. Fatemi, B. Mesman, H. Corporaal, T. Basten, and R. Kleihorst, "RC-SIMD: reconfigurable communication SIMD architecture for image processing applications," *Journal of Embedded Computing*, vol. 2, no. 2, pp. 167–179, 2006.

[36] M. Wan, H. Zhang, V. George, et al., "Design methodology of a low-energy reconfigurable single-chip DSP system," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 28, no. 1-2, pp. 47–61, 2001.

[37] K. Paulsson, M. Hübner, H. Zou, and J. Becker, "Realization of real-time control flow oriented automotive applications

on a soft-core multiprocessor system based on Xilinx Virtex-II FPGAs," in *Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC '05)*, pp. 103–110, Algarve, Portugal, February 2005.

[38] A. Cappelli, A. Lodi, C. Mucci, M. Toma, and F. Campi, "A dataflow control unit for C-to-configurable pipelines compilation flow," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 332–333, Napa Valley, Calif, USA, April 2004.

[39] M. Berekovic, A. Kanstein, and B. Mei, "Mapping MPEG video decoders on the ADRES reconfigurable array processor for next generation multi-mode mobile terminals," in *Proceedings of the Global Signal Processing Conferences & Expos for the Industry: TV to Mobile (GSPX '06)*, Amsterdam, The Netherlands, March 2006.

[40] J. M. Wirthlin and B. L. Hutchings, "Improving functional density using runtime circuit reconfiguration," *IEEE Transactions on VLSI*, vol. 6, no. 2, pp. 247–256, 1998.

[41] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, and G. Cambon, "Metrics for digital signal processing architectures characterization: remanence and scalability," vol. 3133 of *Lecture Notes in Computer Science*, pp. 128–137, 2004.

[42] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.

[43] D. Göhringer, M. Hübner, T. Perschke, and J. Becker, "New dimensions for multiprocessor architectures: on demand heterogeneity, infrastructure and performance through reconfigurability: the RAMPSoC approach," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 495–498, Heidelberg, Germany, September 2008.

[44] L. Benini and G. de Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[45] P. T. Wolkotte, G. J. M. Smit, N. Kavaldjiev, J. E. Becker, and J. Becker, "Energy model of networks-on-chip and a bus," in *Proceedings of the International Symposium on System-on-Chip (SoC '05)*, pp. 82–85, Tampere, Finland, November 2005.