A Concept for Waveform Description based SDR Implementation

T. Kempf, E. M. Witte, O. Schliebusch, G. Ascheid
Institute for Integrated Signal Processing Systems,
RWTH Aachen University, Germany
kempf@iss.rwth-aachen.de

M. Adrat, M. Antweiler
Research Establishment for Applied Science (FGAN),
Dept. FKIE/KOM, Wachtberg, Germany,
adrat@fgan.de

***Abstract*— Waveform specifications are typically given in the form of written documents. In certain cases these textual specifications reveal a lack of unambiguousness, i.e., there exist several interpretations how to implement the waveform onto hardware platforms. Sometimes the solution selected for the one hardware platform might not run properly on another one. Finding a unique waveform description which can be ported to any arbitrary hardware platform becomes a challenging task in particular in view of future *Software Defined Radios* (SDR).**

In this paper, we propose a novel concept for a seamless waveform design. This concept consists of four key parts: 1) a unique waveform description, 2) a general hardware platform description, 3) a temporal and spatial mapping of waveform functionality on the hardware elements, and 4) the automated generation of (SCA-compliant) software/hardware code.

I. Introduction

Software Defined Radios (SDRs) are attracting more and more interest in military, civil and commercial communications. Three key properties of SDRs are the *portability* of waveform functionality from one hardware platform to another, the *interoperability* between SDRs of different vendors as well as with legacy radios, and the *loadability* of new waveform functionality over-the-air. The basis for these features has been set within the JTRS (*Joint Tactical Radio System*) program of the US DoD by defining the *Software Communications Architecture* (SCA) [1]. The SCA is an open, distributed, object-oriented architecture which achieves a separation of the waveform application from the underlying hardware platform. For the smooth cooperation of both parts a set of interfaces has been defined. However, JTRS assumes that the waveform functionality is provided in a way which can directly be applied to any SDR hardware platform. Constraints have neither been defined for the description of waveforms nor for the generation of Soft- and Hardware-code applicable to a specific SDR hardware platform.

In order to provide solutions for these problems research activities have been started in several countries. One of the most famous examples is the international military research program called *Advanced Transmission Language and Allocation of New Technology for International Communications and Proliferation of Allied Waveforms* (ATLANTIC PAW) [2]. In 2000, ATLANTIC PAW was launched as common program of the United States, the United Kingdom, France, and Germany. The aim of this program was to agree on a unique waveform specification standard which allows each partner to port the waveform application to their own programmable hardware platform using a nation specific interpreter. The development of the platform independent *Waveform Description Language* (WDL) has mainly been carried out within the UK's *Programmable Digital Radio* (PDR) program. The UK's PDR program was a two-phased program. In the first phase, an initial prototype for a WDL was developed [3], [4] which is known to be implementation-neutral and which allows to describe the externally observable behavior of a system unambiguously. The automated software/hardware code generation is based on Java and Ptolemy. The combination of the WDL and of the tools automatically generating appropriate software/hardware code is referred to as *Waveform Development Environment* (WDE).

In the second phase of the UK's PDR program [5], a (more or less) completely new concept was derived which is based on Matlab. In the waveform development community, Matlab is more popular than Java or Ptolemy since many libraries and toolsets exist comprising the relevant waveform functionality. In addition, tools are already available for the automated generation of software/hardware code.

Anyhow, both solutions of the ATLANTIC PAW program are linked to specific tools like Matlab or Ptolemy. The close relation to a commercial tool is beneficial [6], [7], because waveform developers are familiar with such tools, but otherwise, there are also a couple of limitations. For instance, neither of these (or of similar) tools is able to automatically generate software/hardware code which is compliant to the *Software Communications Architecture* (SCA) [1].

The major contribution of this paper is the concept of a new seamless design approach for SDRs from a waveform description down to the implementation of the waveform onto an arbitrary SDR hardware platform. The proposed concept is so generic that it can be used with or without commercial tools like, e.g. Matlab. In addition, in view of the *multi-mode* capability of SDRs, several modes of a waveform specification or even several complete standards are summarized in an efficient manner. This allows a fast switching between different modes. The Soft- and Hardware-code can be generated such that they are SCA-compliant.

This paper is structured as follows. In Section II we start with a brief overview of the four key aspects of the basic concept. Each aspect will separately be discussed in more detail in one of the following Sections III to VI. In some cases an representative example will illustrate the proposed concept. Finally, we will close with some conclusions.

II. Basic Concept Overview

Figure 1 depicts the basic principle of the proposed design flow. On the basis of waveform specifications, typically

This research project was performed under contract with the *Technical Center for Information Technology and Electronics* (WTD-81), Germany.

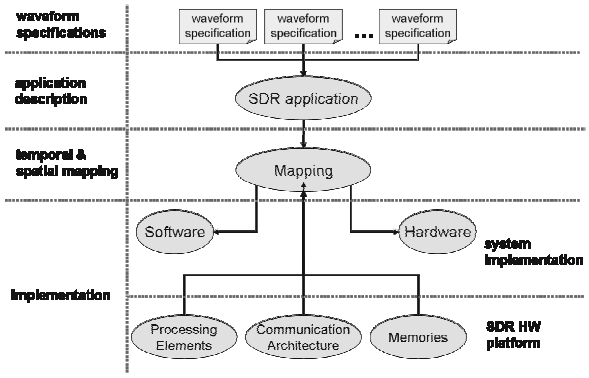

Fig. 1. Design flow from abstract system description to implementation

given in a written document, first a system description is being developed. For an intuitive and efficient description of arbitrary waveforms we introduce a concept which is based on general programming language structures. Advantage of such an approach is a programming language and/or tool independent description, which will be demonstrated exemplarily with the MIL-STD-188-110B waveform. This conceptional approach of a waveform description is far more general than earlier proposed *Waveform Description Languages* (WDLs). In contrary to most of these earlier introduced WDLs, our concept does not focus solely on the description of a waveform. The proposed concept describes a full design approach from specification to implementation.

A second key aspect besides the waveform description is a high-level description of an arbitrary SDR hardware platform in a general and straight way. This description classifies the elements of a SDR hardware platform into three groups: the Processing Elements (PE), the Communication Architectures (CA) and the Memories (Mem). The class of Processing Elements comprises all elements used for executing functionality on the underlying platform. Communication Architectures describe the data exchange capabilities of the PEs, and Memories summarize the storage elements. With the proposed modeling approaches of both, the waveform and the SDR hardware platform, mapping of the application onto the platform becomes feasible.

The mapping of an application, comprising one or multiple waveforms, onto an arbitrary platform can be summarized by a temporal and spatial mapping of the different functionalities onto the Processing Elements. Spatial mapping has to be understood in this context as the mapping of functionalities onto the different PEs in the system, whereas temporal mapping comprises the temporal execution of different functionalities on one PE. The mapping together with the abstract platform model is the basis of the generation of the hardware-configuration and software from the SDR application. Our proposed concept illustrates how to generate automatically these soft- and hardware implementations.

In the following sections these four aspects will be discussed in more detail.

III. SDR Application Description

The two key properties, *interoperability* and *portability*, of tomorrow's SDRs can only be sufficiently provided by a seamless design flow starting from the waveform's specification down to the implementation. Advantage herein is the common basis of the SDR application description, which automatically guarantees interoperability and portability of the derived implementations. In the following we will propose a concept for a SDR application description.

Common characteristic of all waveforms and wireless communication systems is the feature to represent a waveform by a block diagram or equally as a directed graph. Within such descriptions blocks/nodes[1] represent sub-functions of the complete waveform. Edges between the nodes characterize data exchange and finally the overall graph's topology combines the sub-functions to a complete system. Summarizing this, these three key aspects have to be defined within a proper system description:

- *Functionality:* Nodes represent the sub-functions of the complete system. It is essential to provide a sophisticated approach to model such functionalities that will be later introduced.
- *Data Types:* The data types of the graph's edges characterize the data exchange between the nodes.
- *Topology:* The topology graph summarizes the order of sub-function execution and combines them to a complete SDR application.

Herein it is essential to provide a universal and comprehensible modeling approach for such highly complex SDR applications, comprising several waveform implementations. In the following subsections we will introduce a programming-language and tool independent system description dealing with these aspects.

A. Functionality

It is key, to enable an intuitive and effective description of the system's functionality. Effective is meant to decrease the modeling effort, e.g. by maximizing reusability of already developed functionalities. Intuition has to be understood in this context that developers can easily make use of the proposed description language, as it is e.g. easy to learn. Therefore the proposed description-language bases on the very basic structures of common programming languages, like C/C++. Central element in the proposed is the principle of procedures and sequences. These definitions are as follows:

- A *procedure* consists out of several internal variables and a list of sequential expressions, which combine, when executed in sequential order, the functionality of the complete procedure. Exemplarily a code segment looks like the following:

```
Procedure Name {
    procedure variables;
    ...
    expressions;
    ...
}
```

- A *sequence* is characterized by internal variables and a list of sequentially executed procedures. The variables are only accessible from the sequence itself and the procedures contained within, enabling data exchange between these procedures. A sequence code looks as follows:

```
Sequence Name {
    sequence variables;
    Procedure1();
    Procedure2();
    ...
}
```

[1]The terms block and node are equal within this context. In the following of this paper only the term *node* will be used.

Automatically the validity of variables is defined by this concept, which is similar to the *namespacing* principle in C/C++ [9], [10].

Key advantage of this principle is an environment of high modeling efficiency and high reuse. Developed procedures can be arbitrarily reused in different sequences and therefore development effort can be minimized. By a small example of the MIL-STD-188-110B [13] waveform this shall be illustrated:

One of the first blocks of the transmitter part of the MIL-STD-188-110B waveform is the Forward-Error-Correction (FEC)-Encoder. Investigation of the waveform's 13 modes reveals that four different implementations of the FEC-Encoder exist within this single waveform. Despite modeling each single mode, the proposed concept enables development of all sequences on basis of three procedures: *convolutional code-*, *puncturing-* and *repeating- procedure*. By combining these procedures in different ways, modeling of all occurring FEC encoder implementations can be achieved as illustrated in Table I.

TABLE I
EXAMPLE FEC ENCODER

FEC encoder implementation	sequential order of procedures within the sequence
no coding	empty sequence
$2/3$ punctured convolutional code	puncturing convolutional code
$1/2$ convolutional code	convolutional code
$1/2$ convolutional code with repeating	convolutional code repeating

Efficient and intuitive modeling of functionality becomes feasible by this, but real platform implementations demand additionally *initialization-*, *destruction-* and *control-sequences*. To cope with this, we lift the proposed procedure/sequence concept to higher level by defining a *node* element, combining everything required for implementation:

- *Configuration-Sequence:* Performs the switching between different modes. Exists for each node only once.
- *Control-Sequence:* Controls data receiving and sending. Exists for each node only once.
- *Initialization-Sequences:* Used for mode specific initializations. Exists for each implemented mode once, i.e., there might exist several versions for each node, e.g., in case of a multi-mode waveform.
- *Functionality-Sequences:* Defines the modes functionality and therefore exists per implemented mode once.
- *Destruction-Sequences:* Calling of the preempted modes destruction-sequence is done while changing between different modes. For each mode one destruction-sequence exists.

On behalf of this definition the configuration process and data exchange of nodes can be modeled as illustrated in Figure 2.

Configuration starts with an activation signal at the control sequence, which then activates the configuration-sequence for mode switching. First of all the destruction-sequence of the currently active mode is called and afterwards the initialization-sequence of the next active mode is called. Additionally the new mode's control- and functionality-sequences are selected for further data processing.

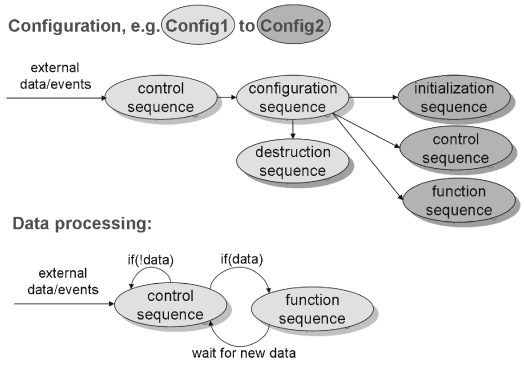

Fig. 2. Node: configuration and data exchange

At the arrival of new data the control-sequence checks, whether data is available to process the functionality, otherwise activation is delayed until sufficient data is available. In case of sufficient data, it is processed by calling the functionality-sequence. Later on control is given back to the control-sequence.

With the proposed node element, highly efficient modeling of every sub-function within an arbitrary waveform becomes feasible. But still no data exchange or the complete SDR application can be modeled with the already proposed. In the next section the issue of data exchange will be addressed.

B. Data Types

Key element for effective modeling is the independence of the exchanged data and the functionality itself. For this purpose in this section we introduce a concept of a highly generic interface, which separates functionality from data exchange. As waveforms can be described by directed graphs, it is sufficient to introduce only unidirectional interfaces. Nevertheless, if users are in the need of bidirectional edges, emulation of these can be done by combining two contrary directed unidirectional interfaces.

The *Master* defines the node, which is sending data over the edge to a *Slave*, which is receiving the data, as illustrated in Figure 3. Key element of this from SystemC [11] derived concept is the independence of Master and Slave that is provided by the so called *edge* element. For instance, let us assume the case that the Master generates a bitstream, but the Slave requires a 32-bit or 16-bit long data word. Conversion of the bitstream to the required data type will be done by the edge element, which makes modeling of the Master and Slave node totally independent. From that follows that the Slave's required bits can arbitrarily change without effecting the Master's bitstream.

To provide reliable data exchange a handshaking is introduced between the Master and Slave. First the Master sends data to the edge element. The edge converts the data from the Master's data type to the requested data type of the Slave

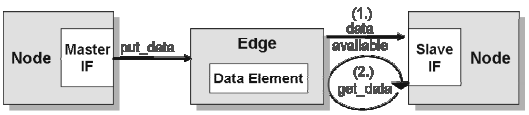

Fig. 3. Data exchange of edge element

and notifies data availability. By accessing the edge element the Slave can request the data.

In the next section the last required issue for modeling complete waveforms will be discussed, namely the topology description that combines the single node elements to a complete system.

C. System Topology

The topology description has to define the constitution of all nodes into a complete system. As it was already shown, even a single waveform can comprise several modes and therefore a complete SDR application, consisting out of several waveforms, can combine hundreds or even thousands of different modes. To cope with the challenge to model such a highly complex system, an efficient modeling for the topology description of a SDR applications has to be invented.

As discussed it is sufficient regarding the topology description to define the connection of nodes and edges of a complete system. As depicted in Figure 4, first of all a topology graph for each mode is being developed. The combination of all of these graphs into one results in the complete waveform's graph. Finally, to gain the complete SDR application topology graph these graphs are summarized in one single graph, holding all modes of all implemented waveforms.

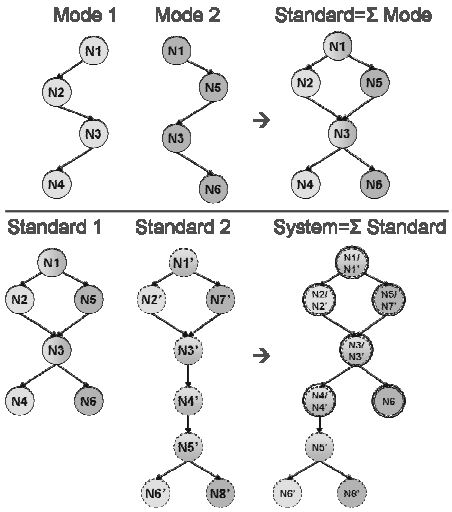

Fig. 4. Topology description

It should be noted, that only waveforms and modes are implemented in such graphs, which have to be available at run-time on the SDR platform. Thus the graph description is kept as small as possible.

A formal definition of the above is like the following:

- $\mathcal{T}$ is the complete SDR application's topology graph
- $\mathcal{S}$ is a waveform's standard topology graph
- $\mathcal{M}$ is a waveform's mode topology graph

Therefore it follows:

$$\mathcal{M} \subseteq \mathcal{S} \subseteq \mathcal{T} \tag{1}$$

Finally with the proposed it is feasible to describe arbitrary SDR applications in a sophisticated way with an efficient modeling approach. Nevertheless it is only the first step to implement a SDR application onto a platform. As in the basic concept mentioned, mapping of the SDR application is only feasible with the knowledge of the underlying hardware platform. Thus in the next section we will focus on how to model an arbitrary hardware platform.

IV. SDR Hardware Platform

To map an SDR application onto a hardware platform, a very coarse-grained model/description of the underlying hardware platform is usually sufficient. Therefore a high-level description of such platforms will be highlighted. Later on we will prove the usefulness and easiness of the proposed in an example of the PRO-3100 board, which is part of the SDR-3000 [12] platform.

Common elements of all SDR hardware platforms can be subsumed by the three groups *Processing Elements (PEs)*, *Communication Architecture (CA)* and *Memories (Mem)*. The class of Processing Elements comprises all elements used for executing functionality on the underlying platform. Communication Architectures describe the data exchange capabilities of the PEs and Memories summarize the storage elements.

The *Communication Architectures* of the hardware platform form the basis for data exchange between the platform's elements. Within the proposed high-level description of such SDR platforms two key parameters characterize the CAs sufficiently:

1) The *type* of the Communication Architecture, which typically is one of the following: wires, point-to-point channels, buses or complex Network-on-Chips (NoCs)
2) The connected *interfaces*, over which the Processing Elements and Memories can exchange data.

Subdivision of the *Processing Elements* can be done into three subgroups, namely the *programmable*, *reconfigurable* and *configurable* ones. Programmable PEs comprise all processors from General Purpose Processors (GPPs) over Digital Signal Processors (DSPs) down to Application Specific ones (ASIPs). Reconfigurable PEs mainly consists out of Field Programmable Gate Arrays (FPGAs), which provide the capability of dynamical reconfiguration to provide more flexibility, compared to the last group of configurable PEs. These consists out of high efficient Application Specific Integrated Circuits (ASICs) with high performance, but a minimum of provided flexibility. Together with this type definition and the existing interfaces of the PEs, they can be completely described at this very coarse-grained description level.

Memories are necessary for data exchange and their temporal storage capabilities. They set constraints for the system-to-architecture mapping, but do not effect the mapping directly. According to their type and interfaces they can be described sufficiently for such a high-level description.

On basis of these three elements (PE, CA, and Mem) arbitrary SDR hardware platforms can be easily described, like a block diagram. Two major parts characterize within this high-level description every platform. The first part consists out of the instances of all elements belonging to the platform. The second part defines the connection of elements, respectively their interfaces. Figure 5 illustrates a model of the PRO-3100 board, which is part of the SDR-3000 platform [12].

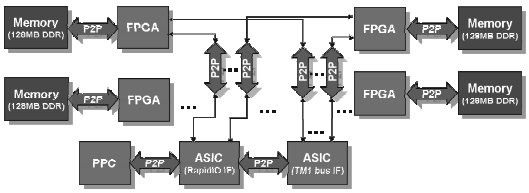

Fig. 5. PRO-3100 board model [12]

The description of this PRO-3100 board would look like the following:

1) Instance section:
   - PEs:
     - Four FPGAs
     - One PowerPC
     - Two ASICs
   - CA: Several Point-to-Point (P2P) connections
   - Mem: Four 128MB DDR Memories
2) Connection of the model's elements:
   - Mem_1.IF → P2P_1
   - FPGA_1.IF → P2P_1
   - ...

With this very coarse-grained platform description the basis for mapping the SDR application onto an arbitrary platform is given, which will be discussed in the next section of this paper.

V. Mapping

Common implementations of systems onto any platform follow always the principle of *temporal and spatial task mapping*. First the system is analyzed and tasks/sub-functions are extracted. Finally these are mapped onto the platform's PEs that are capable to implement these tasks/functions. Despite the crucial effect of such mappings to the systems performance, this issue is not in the scope of this paper. What is in the scope of this paper, is how to generate from any arbitrary mapping an optimized software implementation or hardware configuration for the SDR hardware platform.

Therefore in this section all possibly occurring mappings, as in Figure 6 illustrated, will be discussed and constraints for each implementation case will be highlighted. These investigations will form the basis for the later on discussed automatic code generation. The following mapping cases might occur:

- *m:1-mapping:* m different nodes are mapped onto one PE.
- *1:n-mapping:* One single node is mapped n times on different PEs.
- *m:n-mapping:* m different nodes are mapped onto n different PEs.

A. m:1-mapping

Key element of the m:1-mapping is the temporal mapping of the different nodes to a single PE. The issue of spatial mapping can be neglected, as all nodes and respectively their functionalities are being executed onto this single PE. Regarding the later addressed code generation, whether the nodes are implemented as a software solution on a programmable device or as a hardware configuration on a reconfigurable device, the m:1-mapping requires a sophisticated

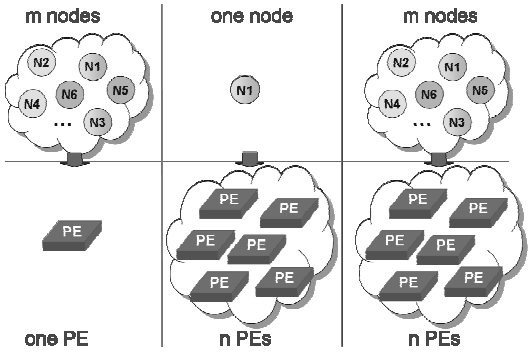

Fig. 6. m:1-, 1:n- and m:n-mapping

temporal scheduling of the functional execution. In the field of software solutions this key issue is quite common and already addressed by Operating Systems (OS) or Real-Time Operating Systems (RTOS). In special cases, where a fixed scheduling can be calculated before run-time more primitive schedulers can also be utilized. Hardware solutions typically implement Controllers to perform such scheduling.

Contrary to the m:1-mapping, the next introduced 1:n-mapping, is in the need of spatial scheduling rather than temporal scheduling.

B. 1:n-mapping

Within the 1:n-mapping one unique node is being mapped onto multiple different or equal PEs. Obviously temporal scheduling is rather simple as one node is being mapped onto each PE and thus the resource can be captured exclusively by the instance of the node. Spatial scheduling becomes key as it is not anymore fixed, which of the nodes instances shall be addressed. Thus a spatial scheduler, implemented inside the previous adjacent node(s), has to determine the addressed instance. Such common scheduling algorithms are e.g. the Round-Robin (RR), Last-Recently-Used (LRU) or Most-Recently-Used (MRU)- algorithms.

It should be noted that a 1:n-mapping is similar to parallel processing and typically has to fulfill several constraints, whereas probably the most important one is the independence of data.

C. m:n-mapping

The occurring mapping case, where m nodes are mapped to n PEs, shall be highlighted by a differentiation into the three following subcases, which are illustrated in Figure 7.

1) *Each of the m nodes is mapped only once:* From sight of the PEs, one or more nodes are mapped

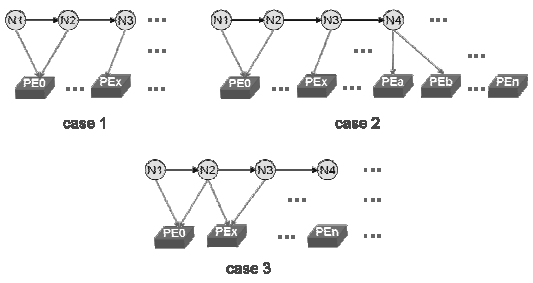

Fig. 7. m:n-mapping cases

to each of the PEs. Thus this mapping can be reduced to several m:1-mappings.

2) *Nodes are mapped multiple times, but nodes mapped more than once are exclusively mapped onto the PE:* This case combines strictly divided m:1- and 1:n-mappings. Classification of the PEs is as follows: PEs with an exclusive mapped node are part of 1:n-mappings, whereas PEs with one or several mapped nodes, belong to m:1-mappings.
3) *Nodes can be mapped arbitrarily and multiple times:* This case does not set any constraint for the number of nodes and PEs. In addition, no restrictions on the mapping occur. Unfortunately no direct reduction to earlier cases can be performed. Anyhow, earlier considerations can be taken up: PEs implementing several nodes require a temporal scheduling (see the m:1-mapping), whereas previous adjacent nodes of nodes, which are mapped multiple times, require a spatial scheduler (see the 1:n-mapping).

The described principles have illustrated all possible system-to-architecture mappings and have highlighted the requests for additional features. In pseudo-code this looks like the following:

```
if (each node is mapped once) {
    A x  [m_{A}:1- mapping]
}
else if( (nodes are mapped more than once) &&
    (these nodes are mapped exclusively to a PE) ) {
        A x  [m:1- mapping] +
        B x  [1:n- mapping]
}
else {
    PEs with multiple tasks require temporal
    scheduling (see m:1- mapping)

    parallel execution of nodes requires
    scheduler in directly previous nodes
    (see 1:n- mapping)
}
```

Based on the mapping considerations automatic code generation, whether software code or hardware configurations, becomes feasible.

VI. SW/HW Code Generation

Goal of the proposed seamless design flow is the implementation of an arbitrary SDR application onto an hardware platform. Therefore it is key to generate automatically, on basis of the discussed mapping, optimized and efficient soft- and hardware implementations for the underlying platform. On a first look soft- and hardware implementations are quite different, whereas software typically is developed in a high-level programming language like C/C++ and hardware is mainly developed in some Hardware Description Language, like e.g. Verilog or VHDL. Nevertheless the basic principle of how to generate a software or hardware solution is the same. On basis of the mapped nodes a combined implementation for the PE has to be generated. This code comprises on the one hand the initialization and mode-change and on the other hand the functional implementation.

To provide a highest possible degree of interoperability and portability it is mandatory to fulfill the Software Communications Architecture (SCA) [1] defined within the JTRS. As depicted in Figure 8 external data is exchanged according to the interfaces specified by the SCA (see Section VII). Here it should be noted that the specified interfaces are expected to produce a significant overhead. Wherever possible it is beneficial to replace such complex interfaces by simpler ones. Our proposed concept with the automatic code generation includes such code optimization right from the start.

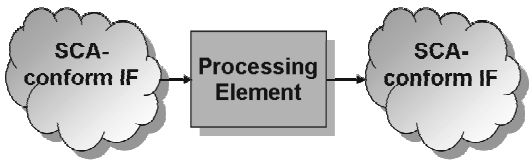

Fig. 8. SCA-conform Processing Elements

A. Initialization & Mode-change

Initializations and mode-changes follow a call principle independently from a system-to-architecture mapping. The following pseudo-code example illustrates such a basic principle:

```
(1) activates control sequence;
(2) calls configure sequence;
(3) if(mode change) calls destructor sequence
    of all mapped nodes;
(4) calls initialization sequences of new
    configuration of all mapped nodes;
(5) selects function sequences of new configuration;
(6) selects control sequences of new configuration;
```

This calling principle guarantees sufficient data saving of the preempted mode as well as initialization of the activated mode. Together with the selection of the active function- and control-sequences mode initialization is finished.

B. Functionality

Dependent on the mapping of the system onto the underlying platform functional processing can be optimized with respect to the used interfaces and implementation. Hereby all available information regarding the system, platform and mapping will be used for optimizations during code generation.

Regarding the earlier discussed mappings a concept for soft- and hardware code generation will be introduced.

m:1-mapping:

Key element within the m:1-mapping is the temporal scheduling, whereas spatial scheduling can be neglected as only one PE is used for implementation. On the one hand software solutions request Operating Systems (OS) or Real-Time-OS and on the other hand hardware solutions typically demand Controller structures for such temporal scheduling. Although code optimizations on basis of the available information can increase the performance significantly. For optimizations two different cases have to be distinguished:

1) Adjacent nodes, which are directly connected over an edge, can make use of simpler interfaces as the ones provided by the SCA. These interfaces within software solutions range from data exchange over complex data structures till data exchange over local variables. For hardware these interfaces appear as wires, registers, buffers or even more complex structures.
2) Unfortunately for non adjacent nodes such optimizations are not available.

1:n-mapping:
As in Section V discussed this mapping requires spatial scheduling in the implementation of the directly previous adjacent nodes of the n times mapped one. This implementation works on the basis of common scheduler algorithms, whereas the Round-Robin algorithm is probably the most common one. In pseudo-code an automatically generated code of the previous node would look like the following:

```
...
function();
if (data send multiple times mapped node) {
    Instance = schedule(node instance);
    send data to Instance;
}
...
```

m:n-mapping:
The earlier discussion of the m:n-mapping has revealed the special cases of such a mapping, which can now be used for optimized code generation.

1) Regarding the first case of once mapped nodes the reduction of the mapping to several m:1-mappings is feasible. As above illustrated code optimizations with respect to this case are available.
2) Within the second case the m:n-mapping can be traced back to strictly divided m:1-and 1:n-mappings. As in the previous case code generation of these mappings has been already introduced.
3) Finally, arbitrary and multiple mapping inside the m:n-mapping requests a mixed implementation code based on the basic m:1- and 1:n-mapping.

VII. Relations to the SCA

In addition to the physical hardware elements (PE, CA, and Mem) each SDR hardware platform will exhibit an *operating environment*. In case of an SCA-compliant platform the operating environment consists of a real-time operating system, a CORBA middleware (for the distributed and platform independent data processing) and a core framework [1]. The core framework defines a set of interfaces which can be grouped into the three subsets. The *base application interfaces* are needed e.g. for communication between the software components, the *framework control interfaces* are used for e.g. the control and management of these components, and the *file service interfaces* provide file access services. Of course, the automatic SW/HW code generation has to be compliant to all of these interfaces.

However, there exist a couple of additional items where the proposed concept bears some conjunctions with the SCA. Considering these conjunctions can improve the efficiency of the proposed concept and it can ease the design flow from waveform specification to implementation.

For instance, another key part of the SCA specification is the core framework's *domain profile* which is a set of descriptors. One of these is the *software assembly descriptor* (SAD) which describes the *assembly* of software components and hardware devices. Such an *assembly* resembles the system topology as described in Section III-C. Thus, when generating the SW/HW code with SCA compliant interfaces also the generation of *domain profile* files like SAD is beneficial. These files cover important information for the SCA *ApplicationFactory* as this manages the mapping of software code to hardware devices as well as the proper linkage of the software components. Notice, in this respect the proposed node element can be considered as an SCA software *component* and the *edge* element is similar to the *port* interface of an SCA *resource*. Moreover, the PEs & CAs of Section IV are represented as *HWModules* within the SCA specification, and finally, both types of devices, the *loadable* and the *executable* ones, can be seen as an implementation similarly to the proposed SW/HW code implementation.

VIII. Conclusions

In this paper, we propose a concept for a seamless design flow for Software Defined Radio applications. The main contribution lies in a concept starting from the waveform's specifications and ranging down to the implementation onto an arbitrary SDR hardware platform. Key element within this design approach is a common description of an SDR application, which ensures interoperability and portability automatically. By a programming-language and tool independent approach the proposed concept is not limited to any proprietary tool and it makes SCA compliant code generation available. Together with a highly abstract SDR hardware platform model, mapping of the application onto the platform becomes feasible. This temporal and spatial application-to-architecture mapping has been broken down to all possibly occurring cases, whereas code generation of SCA-conform software implementations and hardware configurations have been discussed within this paper.

In future we will focus on the development of a tool implementing the proposed concept. Finally, we aim to prove the proposed concept on basis of a waveform, like e.g. the MIL-STD-188-110B. Additionally, our future work will concentrate on the generation of simulation models for early application and platform development and exploration.

IX. Acknowledgement

The authors would like to thank Mr. T. Gempel, M. Plessa, and C. Hatzig of the *Technical Center for Information Technology and Electronics* (WTD-81) for inspiring discussions.

References

[1] Software Communications Architecture (SCA) Specifications V2.2, *http://jtrs.army.mil/*.
[2] ATLANTIC PAW Webpage, *http://mccoy.ucsf.edu/emondi/Public/APaw/APaw_Writeup.htm*.
[3] E.D. Willink, WDL Webpage, *http://www.ee.surrey.ac.uk/Personal/E.Willink/wdl/wdl.html*.
[4] E.D.Willink, *Waveform Description Language : Moving from Implementation to Specification*, IEEE Military Communications Conference (MILCOM 2001), pp. 208–212, vol.1, Washington, D.C., Oct. 2001
[5] R.S. Prill, *E2E Reconfigurability Using OSI Layered WDL Canonical Radio Model*. E2R Workshop on Reconfigurable Mobile Systems and Networks Beyond 3G, Barcelona (Spain), Sept. 2004 .
[6] SDR-Forum WDE Workshop, Rome (NY), Nov. 2000, *http://www.sdrforum.org/MTGS/wde_wkshp_11_00/wdl_ws_agenda_final.htm*
[7] M.S. Gudaitis and R.D. Hinman, *Practical Considerations for a Waveform Development Environment*. IEEE Military Communications Conference (MILCOM 2001), pp. 190–194, Washington, D.C., Oct. 2001.
[8] Common Object Request Broker Architecture (CORBA), *http://www.corba.org*.
[9] ISO 9899 C-programming language.
[10] ISO/IEC 14882 C++-programming language.
[11] System Design with SystemC, T. Grötker, S. Liao, G. Martin, S. Swan, Kluwer Academic Publishers, 2002.
[12] SDR-3000 platform by Spectrum Signal Processing Inc., *http://www.spectrumsignal.com*.
[13] MIL-STD-188-110B Departement of Defense Interface Standard, April 2000.