Communication Patterns: a Novel Modeling Approach for Software Defined Radio Systems

Andrea ENRICI, Ludovic APVRILLE and Renaud PACALET
Institut Mines-Telecom
Telecom ParisTech, CNRS/LTCI
Biot, France
Email: {andrea.enrici, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

***Abstract*—Efficiently programming Software Defined Radio applications still remains a pending challenge. While most of the efforts are focused on the processing part of a design, communications are a great source of performance and portability issues that is often neglected. Within the frame of a Model Driven Engineering methodology for the design of dataflow processing applications, this paper proposes a novel approach to model complex communication interactions. This approach relies on communication patterns to capture communication protocols and standards at system-level, independently of computations. A case study for cognitive radio shows how communication patterns are efficiently used to generate cross-platform models that can be ported and refined to specific applications and architectures.**

***Keywords*—Software defined radio; hardware/software codesign; model-driven engineering**

I. INTRODUCTION

In the last decade, the demand for more flexible and reconfigurable solutions able to support multiple communication standards has lead to a shift from radios where the full range of capabilities was supplied by hardware elements to Software Defined Radios (SDRs) [1] where some or all of the physical layer functions are software defined [2]. This technological shift takes advantage of common Digital Signal Processing (DSP) algorithms, shared by modern air-services protocols and standards (e.g., Global Positioning System). A SDR system is composed of a *platform*, intended as the set of hardware elements (e.g., DSPs) and software layers (e.g., Operating System), on top of which a software *waveform* is executed. The latter is defined as the software application coordinating and configuring the platform in order to transform the information contained in the signals to be transmitted and received. Implementing a waveform in software adds greater flexibility to radio systems as their functionality can be changed by a simple software update without the need to change equipment. This added flexibility yields numerous advantages including increased service life time of equipments, cross-platform portability of the software, reduced costs and better ease in terms of system implementation, upgrade and maintenance. In order to adequately exploit the benefits of software waveforms, SDR platforms provide a way to implement computations and communications in a very generic and flexible way. However, the price to pay for this flexibility and genericness is an increase in the complexity of programming SDR platforms [3]. Currently, the latter are manually programmed by system experts in languages like C/C++. Nevertheless, such languages do not shield programmers from the system complexity, leading to higher development costs, longer time-to-market of new products, as well as reduced cross-platform portability of waveforms.

Thus, a current hot topic for the industrial and scientific community is to find an efficient way to automatically program such complex systems [4]. Currently, the most promising strategies rely on Model Driven Engineering (MDE) combining Domain Specific Modeling Languages (DSMLs) with transformation engines and generators to synthesize source code or alternative model representations [5]. Methodologies for the design of SDR systems employing the above strategies are discussed in Section II. These methodologies, however, either informally prioritize computations over communications or propose non-portable solutions that are specific for a given platform. A consequence of such approaches is that the limited expressive power of models restricts the systems that can be described and does not allow to efficiently exploit all the platform capabilities in terms of addressing and data transfers. The work presented in this paper enriches DiplodocusDF [6], a methodology for design and code generation of heterogeneous processing applications of type dataflow, based on MDE and UML. The main contributions introduced here are (1) a novel feature to capture complex communication schemes, early at system-level, independently of computations and (2) a novel approach for modeling SDR systems in a portable way.

This paper is organized as follows: the related work is described in Section II followed by the context of our works, in Section III. Section IV describes our solution to model communications, its integration into DiplodocusDF and how the latter paves the way to a novel modeling approach. To show the benefits of our contributions, Section V applies them to a practical case study for cognitive radio. Conclusion along with the state and directions of our works are given in Section VI.

II. RELATED WORK

SDR design is a hardware/software co-design topic that deals with most issues of design space exploration for embedded real-time systems. The complexity of SDR systems and their need to dynamically adapt to the environment (cognitive radio) make current methodologies for embedded real-time systems unsuitable to properly cope with all the requirements of SDR design (e.g., flexibility, heterogeneous processing, hardware abstraction). UML methodologies that reflect this statement, are those for the development of embedded applications that are based on the MARTE profile [7]. UML-MARTE in fact, lacks the notations to describe dataflow applications such as SDR in a pure abstract way and its

 ISBN: 978-1-61208-323-0

procedural approach leads to models with only one centralized controller, as mentioned in [6]. UML-MARTE has been successfully proposed for platform design, instead: the MOPCOM [8] project applies this profile to describe real time properties and to modeling in order to generate code for implementation and verification. On the other hand, closer to the aims of DiplodocusDF, the A3S project [9] provides a tool and a methodology to design, model and verify SDR systems in UML, targeting non-functional characteristics of both hardware and software components. A3S defines a UML profile, specific to SDR, but lacks the generation of validation and implementation code, as opposed to DiplodocusDF. Nevertheless, UML is not the only solution for abstract system modeling of SDR systems: other existing approaches are based on dataflow graph (DFG) representations. [10] is a MDE approach that proposes a lightweight programming model for describing SDR applications, by means of dataflow Models of Computation. The authors propose a minimally intrusive design flow that, however, needs preexisting tools, libraries and processes to breathe life into models. The paper [11] presents an interesting solution that, to our knowledge, shares the most with DiplodocusDF in terms of gathering within a unique framework several concepts and tools for heterogeneous design of SDR systems. The authors propose a methodology based on the Syndex tool [12], that allows executable code generation from high-level models through a series of graph transformations. The proposed approach is oriented to ultra-fast prototyping for heterogeneous platforms so as to deal with realistic implementations instead of simulation results. Applications are described as extended DFGs where nodes represent processing operations and edges represent data transfers. Similarly in the architecture graph, vertexes model hardware components and hyper-edges represent communication media. The key strength of the proposed methodology lays in its support for heterogeneous processing platforms (e.g., Field-Programmable Gate Arrays), the automatic code generation and the associated scheduling. With respect to communication modeling, [11] does not extend the way communications are handled in Syndex, so the critics related to graph-based approaches in Section III-C can be addressed to [11] also.

III. CONTEXT

In order to illustrate SDR platforms, this section provides an outline of the hardware architecture of Embb [13], the platform on which the implementation code generated by DiplodocusDF has been successfully executed. A more detailed description of DiplodocusDF follows.

A. The hardware platform Embb

The authors in [13], propose a new generic baseband architecture for SDR applications. An instance of such a platform is depicted in Fig. 1. It is composed of (1) DSP units, (2) a system interconnect and (3) a main CPU. The DSP units are in charge of executing the processing operations (e.g., FFT). They are equipped with a hardware accelerator as computational unit (Processing SubSystem, PSS), a DMA to transfer data, an internal memory mapped on the main processor memory and a microncontroller (μC) that allows to reduce interventions of the main CPU. The latter is linked to DSPs via the system interconnect that allows data-blocks and control information to be exchanged among units. The main processor executes the waveform control operations: it manages data-transfer operations, the computational units and the interface with the external environment (Fig. 1, yellow area).

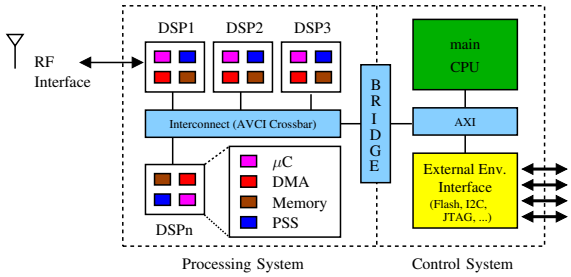

Fig. 1. The architecture of an Embb instance

Executing parallel applications on SDR platforms is not trivial because of both memory and computational resource pooling. This implies dense flows of data and control information being transferred among hardware units. Since these flows access shared resources (e.g., the bridge in Embb between the control and the processing systems), their impact over the system's performance cannot be neglected and urges for modeling techniques to properly take them into account when designing applications.

B. The DiplodocusDF methodology

DiplodocusDF [6] is a UML MDE methodology for the design of heterogeneous dataflow applications for real time embedded systems, in particular radio (such as Software Defined Radio), Fig. 2. It stems from DIPLODOCUS, [14], a UML Model Driven Engineering methodology for hw/sw partitioning of Systems on Chip at high abstraction level, currently implemented by the free software TTool [15]. The core strength of DIPLODOCUS is the automatic transformation of models for simulation and formal verification [16]. However, the DIPLODOCUS approach is too abstract to permit automatic code generation for SDR systems as models lack the necessary expressiveness to face the complexity of platforms and waveforms. DiplodocusDF is a first attempt to fill the aforementioned gaps; it enriches DIPLODOCUS with the following extensions:

- **A dataflow semantics:** an application is modeled as a dataflow graph, where nodes represent tasks (processing, routing, addressing operations) and edges are used to carry data-blocks and the related control parameters (e.g., r/w memory addresses).
- **A specialization of the architecture language:** heterogeneous platforms are represented as a network of computation nodes (e.g., DSPs), storage nodes (e.g., memories) and data-transfer nodes (e.g., bus) interconnected by communication edges.
- **An environment for automatic generation of executable code:** the description of a waveform mapped over a platform is translated in C-language code via an Intermediate Representation completed by the platform Application Programming Interface (API) and by a Run Time Environment for scheduling computations.

 ISBN: 978-1-61208-323-0

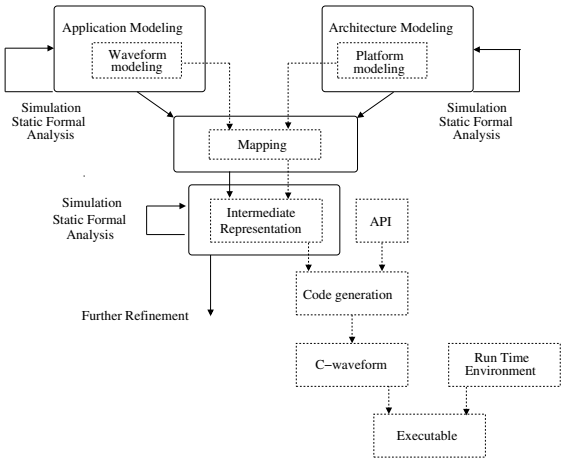

Fig. 2. The DIPLODOCUS (solid lines) and DiplodocusDF (dotted lines) methodologies

C. Communication modeling in DiplodocusDF

Since DiplodocusDF is a graph-based methodology, it models communication exchanges among processing operations by means of edges in the waveform DFG. At mapping stage, edges are projected over *buffers*, defined as memory regions of the platform storage elements, in a one-to-one fashion. Consequently, this forces the buffer of the producer operation to reside in the same memory as the one of the consumer operation. At mapping stage, the application graph is adapted to the platform addressing capabilities in order to take into account data transfers. This typically implies the injection of additional nodes in the application, e.g., to map multiple edges to the same buffer, to model data transfers that employ a DMA. This adaptation may also lead to alter the natural sequence of processing operations. As for the graph-based approaches of Section II, in DiplodocusDF application models, data transfers from a producer node to a consumer node can only be mapped on simple P2P paths in the platform. In other words, no complex data-transfer scheme with more than one DMA and with intermediate memories between the source and destination memories can be described. The source node and the destination node of a transfer must be able to directly access a storage element via either a bus or one single DMA. These characteristics restrict the design space in terms of waveforms and platforms that the methodology can describe.

IV. Communication Patterns

In addition to computations, the processing of radio signals includes communication exchanges between architecture nodes (e.g., DSP, bus). For instance, these communications represent exchanges for operations running on different units, or items (i.e., data, instructions) fetched from memory as part of one processing operation. These communications may be implemented by means of different standards and protocols (e.g., AMBA, PCI Express) within the same platform, thus making each transfer different in terms of performances and interactions among nodes. However, communications share common patterns independently of the mechanisms being used. Thus, our objective is not to model communication standards in details, but rather to model their underlying patterns at a high level of abstraction and adapt them to the transfer capabilities of a specific platform. This permits to describe the influence of communication interactions on the system's performance and to provide models with the expressive power to generate code for communications in a portable way. To reach this target, the initial concepts of *communication pattern* [17] are extended and the resulting contribution is integrated into an enriched DiplodocusDF methodology.

A. Communication Patterns in a Nutshell

A Communication Pattern (CP) describes a transfer between architecture actors. It is associated to a *communication flow*, between a source S and a destination D nodes in the application that are continuously connected by a set of edges. In other words, edges must build a path in the application graph that links S to D without interruptions. More formally, a communication pattern is defined as a tuple:

$$\mathcal{CP} = (\mathcal{N}_{arch}, \mathcal{E}_{arch}, \prec_{\mathcal{N}}, \mathcal{N}_{app}, \mathcal{E}_{app}, \mathcal{A}, \prec_{\mathcal{A}})$$

- $\mathcal{N}_{arch}$ is the set of architecture actors involved in the transfer;
- $\mathcal{E}_{arch}$ is the set of architecture edges connecting the architecture actors of $\mathcal{N}_{arch}$;
- $\prec_{\mathcal{N}} \subseteq \mathcal{N}_{arch} \times \mathcal{N}_{arch}$ is a total order relation among architecture actors;
- $\mathcal{N}_{app}$ represents the source S and destination D nodes from the application;
- $\mathcal{E}_{app}$ is the set of edges from the application graph that make up the communication flow from S to D;
- $\mathcal{A}$ is a set of actions performed by $\mathcal{N}_{arch}$ to accomplish the transfer;
- $\prec_{\mathcal{A}} \subseteq \mathcal{A} \times \mathcal{A}$ is a partial order relation among actions;

A CP is represented independently with respect to the waveform and platform models as an extended UML Sequence Diagram. Fig. 3 shows an abstract communication pattern for the communication flow of edge ed1 in Fig. 6. In Fig. 3, a generic data transfer is requested by a Controller actor and executed by a Transfer actor between a source and a destination storage actors. The actors (N_{arch}) in Fig. 3, are *architecture supernodes* representing generic architecture elements that are purely functional. Supernodes can be classified in three

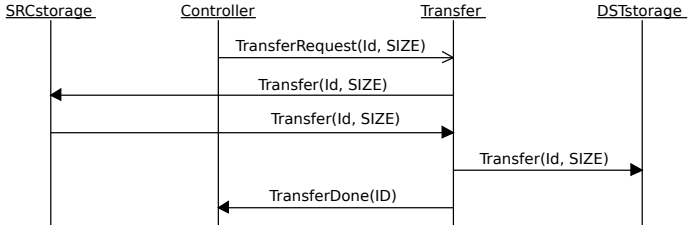

Fig. 3. An abstract communication pattern

classes: *storage* (e.g., memory) for item storing, *controller* (e.g., CPU, DSP) for control and configuration, *transfer* (e.g., bus, bridge) for routing and dispatching. Fig. 3 also shows some of the actions A among the architecture actors. An action is associated to an actor and provided with a type representing

its functionality and a set of parameters to specify the latter. Actions are classified in three types: *Message Actions*, *Execution Specification Actions* and *Conditional Actions*.

Message Actions represent an abstraction of the interactions used by communication protocols and standards. A message action can either be blocking or non-blocking: the former blocks the sender until completion while the latter allows the sender to switch to another task (e.g., *TransferRequest()* in Fig. 3).

Execution Specification Actions model the processing time of an ongoing communication. The semantics of these actions depend on the underlying architecture node and can be observed after mapping CPs over the architecture. Two execution specification operators are defined: Exec, which represents the cost of actively executing a communication operation (e.g., the number of cycles taken by a bus to execute a transfer) and Delay, which is a generic time interval used when a node is not actively involved in a communication operation, e.g., to model the time taken to complete a DMA transfer from the CPU perspective.

Conditional Actions add conditional behavior to communication patterns to capture, for instance, precedence constraints in complex transfers. Conditional actions are of two types: the loop construct (Fig. 5, 10) and the if-else construct, respectively for iterative and conditional communication exchanges. A conditional action includes other actions and may be nested according to the scenario being described.

B. The Proposed Methodology

Following the above description of what a communication pattern is and looks like, it is now illustrated how describing communications with CPs results in a novel modeling approach. The added value of communication patterns is the separation of concerns between communications and computations that allows to elegantly capture the functionality of an application description. Since both communications and computations involve different sets of architecture elements, this separation of concerns translates into a separate mapping and refinement on the architecture graph. Fig. 4 illustrates such a novel approach that intends to replace the classic Y-Chart [18] scheme adopted by DiplodocusDF in the modeling phase of Fig. 2. Instead of projecting the application on the architecture in one single stage, followed by an adaptation stage, computations in the waveform DFG are mapped first and communications (CPs) are mapped next. In our approach, computations are atomic operations whose execution cannot be split over several processing nodes. Thus, mapping of computations is done in a single step by associating the application nodes onto the architecture processing nodes (level L2, Fig. 4). Conversely, as complex communication paths are expressed by multiple kinds of elements, a separate mapping step is required. In Fig. 4, the box corresponding to level L3 regroups this mapping in a single stage.

The top-most level of the methodology, L0 in Fig. 4, is represented by three sets of repositories containing the building bricks of the application, architecture and communication patterns. The application repository contains nodes for signal processing operations and edges for inter-node communication that are used to build the waveform DFG. For the architecture graphs and the communication pattern models, two separate repositories are provided, as supernodes in CPs have a higher

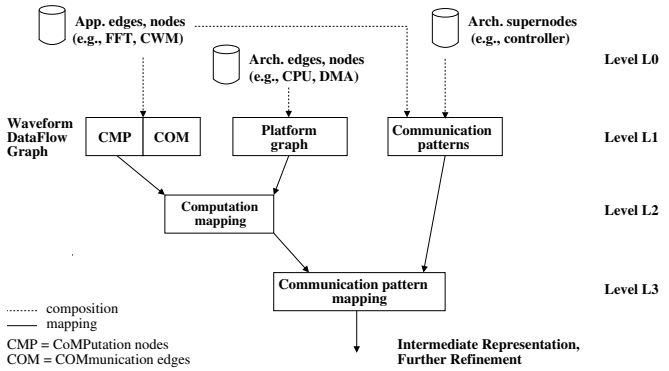

Fig. 4. The proposed modeling approach

level of abstraction with respect to nodes in the platform.
Thus, at level L1, models consist in a waveform DFG (Fig. 6), a platform graph (Fig. 8) and stand-alone communication patterns (Fig. 3 and 9). At this level of abstraction, the purpose of such high-level models for communications (CPs) and computations (waveform DFG) is to express the application in a functional and portable way, that transcends from the actual platform implementation since the methodology aims at designing portable SDR applications.
Once level-L1 models are available, at level L2 computations are mapped over the architecture processing units according to their computational power (Fig. 8). Due to the communication capabilities of processing units, this mapping step constraints the mapping of communication patterns at level L3. Here, the abstract supernodes are mapped over the platform nodes: storage supernodes are associated to memories, controller supernodes to CPUs and transfer nodes to a network of DMAs, buses and bridges (Fig. 5, 10). It is important to state that mapping of storage and controller supernodes takes place in a one-to-one fashion. In fact, none of the two types of supernodes can be split over multiple platform units. Allowing such a one-to-many mapping would in fact imply additional transfers, within the set of mapped units, that were not described in the initial communication pattern. On the other hand, transfer supernodes nodes are associated to a network of transfer units in a one-to-many fashion to permit modeling of complex transfer paths requiring intermediate elements in between the source and the destination storage.

Fig. 5 shows the abstract communication pattern of Fig. 3,

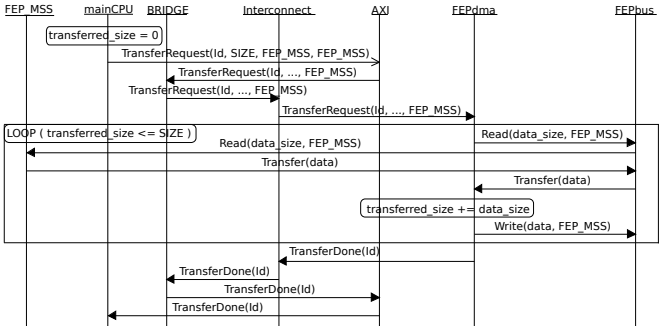

Fig. 5. The Communication Pattern of Fig. 3, after mapping level L3

mapped (level L3) on the architecture of Fig. 8. At level L2, SRC and FFT computations have been mapped to mainCPU

and FEP (a DSP unit), respectively. Fig. 5 describes how data between SRC and FFT is moved within memory FEP_MSS (where SRCstorage and DSTstorage supernodes have been mapped) via the DMA FEPdma. mainCPU requests the DMA transfer via a network of transfer nodes made up of AXI bus, BRIDGE and Interconnect. Such a transfer is then executed by iteratively reading data from FEP_MSS, storing them into the intermediate FIFO of FEPdma and writing to FEP_MSS. The latter actions are governed by a loop conditional action that iterates according to the value of transferred_size. A notification is eventually sent back from FEPdma to mainCPU via Interconnect, BRIDGE and AXI.

V. Case Study: Welch Periodogram Detector

In this section, an implementation of the Welch Periodogram Detector (WPD) algorithm is taken from [6] as a scenario for a case study. WPD is a energy detection algorithm used for sensing the spectrum and detecting when a given frequency band can be opportunistically used. Fig. 6 shows the dataflow graph of level L1 for WPD, where only edges and processing operations are modeled, together with a FORK node to broadcast data. Here, a source (SRC) produces the input vectors whose frequency representation (FFT) is processed by the component-wise square of modulus (CWM). Next, two CWM output vectors are component-wise added (CWA) and the elements of the resulting vector are summed (SUM) and collected (SINK). On the other hand, Fig. 7 illustrates how the DFG of Fig. 6 is adapted, in DiplodocusDF, to Embb by injecting routing and addressing nodes (e.g., OVLP, DMA). Fig. 8 shows the architecture graph of level L1 for the portion of Embb (Fig. 1) relevant to the case study. The latter figure also displays mapping of computation operations (level L2) and the mapping of storage supernodes (level L3) for the CPs relevant to the case study. The Front End Processor (FEP) DSP [13], offers all the computational power required to process the WPD waveform. It is modeled as an interconnected subsystem with a CPU (FEP) for executing computations, a memory (FEP_MSS) for storing data and a DMA (FEPdma) for transfers to/from the memory via an internal bus (FEPbus). The remaining elements in Fig. 8 model the interconnect (Interconnect, TAVCI, BRIDGE) and the control part (mainCPU, mainBus, mainMemory, mainDMA) of Embb.

With respect to [6], the original case study is re-visited by adding the following requirement: let us suppose that sensing the spectrum with WPD is part of a larger scenario where the frequency representation of the input signal must be stored apart for later processing. Modeling such a requirement in DiplodocusDF or in one of the graph-based methodologies of Section II, would require a waveform re-design, e.g., to add a node collecting the output of FFT plus the related addressing and routing operations to perform the transfer. Instead, communication patterns can capture this exigency handily by adding one simple edge, ed8 in Fig. 6, to the waveform DFG of level L1. The communication pattern of level L1 for the communication flow corresponding to edges ed1 and ed8 is showed in Fig. 9. Here, data are transferred between Storage1 and Storage2 (edge ed1), then copied from the latter to Storage3 (edge ed8). Computations at level L2 are mapped to FEP, while at level L3 the transfer path can now be described with much greater flexibility: the designer can chose to copy data to FEP_MSS, mainMemory or any combination of the two. Similarly, the communications can be executed with or without DMA, via FEPdma, mainDMA, Interconnect, FEPbus or any combination of these elements. In this case study, it has been chosen to copy the FFT output to mainMemory by means of mainDMA.

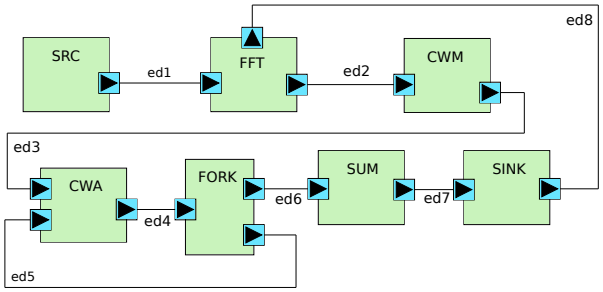

Fig. 6. WPD waveform DataFlow Graph of level L1

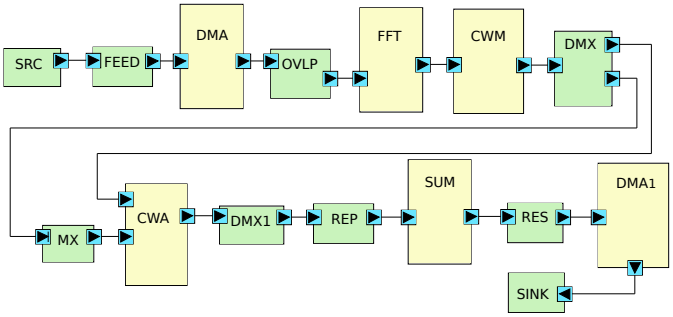

Fig. 7. WPD waveform DataFlow Graph of level L1 adapted to Embb

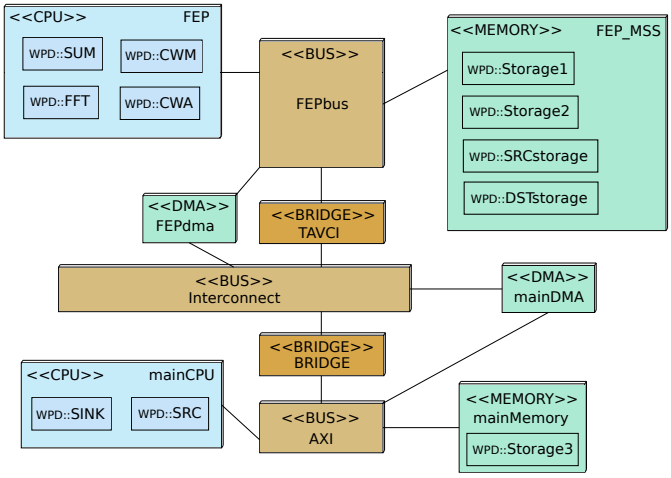

Fig. 8. The graph of the instance of Embb used for the WPD case study

For the sake of simplicity, Fig. 10 only illustrates the CP of level L3 for edge ed8. The complete CP of level L3 for both ed1 and ed8 can be composed by merging those in Fig. 5 and Fig. 10. In Fig. 10, the scenario is similar to that of Fig. 5, but more actors are involved and data are copied instead of being transferred. So, mainCPU programs a CopyRequest

 ISBN: 978-1-61208-323-0

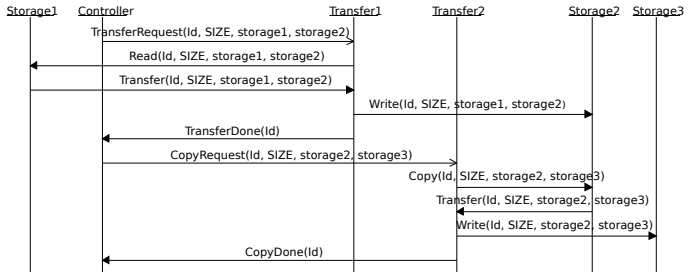

Fig. 9. Communication Pattern of level L1 for edges ed1, ed8

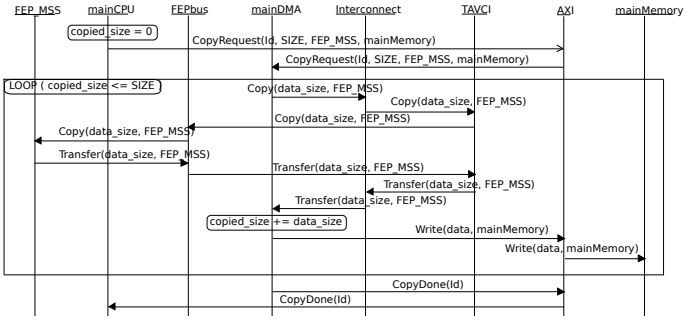

Fig. 10. Communication Pattern of level L3 for edge ed8

to mainDMA via the bus AXI. mainDMA then reads data from the source memory FEP_MSS via the transport network made up of Interconnect, TAVCI and FEPbus. The read data are stored in a FIFO in mainDMA and then written to the destination mainMemory via the bus AXI. The latter actions are iteratively executed until all data have been transferred according to the loop conditional action and an acknowledgment is sent to mainCPU by FEPdma via AXI bus.

VI. Conclusion and Future Work

This paper described Communication Patterns, a novel feature for modeling the behavior of complex communication schemes at system-level. While CPs are presented here for SDR systems, they also represent a solution for other dataflow processing domains, e.g., image processing. As illustrated by the case study, our contributions provide the expressive power to describe complex multi-point transfer schemes that cannot be captured by traditional graph-based approaches that normally call for a re-design of the application. Moreover, CPs make application models portable by eliminating the need to adapt the latter to the addressing capabilities of a specific platform, thus leading to finer and faster designs. This paves the way to a novel modeling approach where waveform and platform graphs are disjoint and information contained in the models is separately mapped on the architecture in a extended Y-Chart fashion. The main gain of Communication Patterns results in the developer having complete control over communications, independently with respect to the rest of the application and portably with respect to different platforms. Our current works are dedicated to integrate CPs and the proposed methodology in TTool. At this stage, the advantage of using CPs is given in terms of modeling. However, once completely integrated into TTool, CPs will allow the user to separately investigate and extract information about the performance of data/control transfers, either via simulation or via code generation, without changing the rest of the system's models. Our future works will provide a more complete description of mapping and refinement rules for CPs. For instance, level L3 will be extended into intermediate mappings where each class of supernodes will be addressed separately, in order to target buffer addressing and indexing. These aspects are currently handled in DiplodocusDF waveforms by decorating edges with addressing parameters (e.g., read/write memory offsets) and instantiating dedicated nodes (e.g., OVLP in Fig. 7) that define how data are stored in memories. Our aim is to embed these functionalities in communication patterns and processing operations in order to achieve portable waveforms made up of pure dataflow representations like the one in Fig. 6.

References

[1] J. Mitola III, *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*, Ph.D dissertation, Royal Institute of Technology (KTH), May 2000.

[2] The SDR forum, *http://www.wirelessinnovation.org*, [retrieved: December, 2013].

[3] B. Trask, *Two Thirds of SDR is SD*, in SDR WInnComm, 2010, pp. 197-198.

[4] O. Anjum et al., *State of the art baseband DSP platforms for Software Defined Radio: A survey*, in EURASIP Journal on Wireless Communications and Networking, vol. 2011, no. 1, 2011.

[5] D. C. Schmidt, *Model-Driven Engineering*, in IEEE Computer, vol. 39, no. 2, 2006, pp. 25-31.

[6] J. M. Gonzalez Pina, *Application Modeling and Software Architectures for the Software Defined Radio*, Ph.D dissertation, Telecom ParisTech, May 2013.

[7] The UML profile for MARTE, *http://www.omgmarte.org*, [retrieved: December, 2013]

[8] S. Lecomte, S. Guillouard, C. Moy, P. Leray and P. Soulard, *A co-design methodology based on model driven architecture for real time embedded systems*, in Mathematical and Computer Modelling, vol. 53, no. 34, 2011, pp. 471-484.

[9] S. Rouxel et al., *UML Framework for PIM and PSM Verification of SDR Systems*, in SDR Forum Technical Conference, 2005.

[10] C.C. Shen, W. Plishker, H.H. Wu and S.S. Bhattacharyya, *A Lightweight Dataflow Approach for Design and Implementation of SDR Systems*, in SDR-WInComm, 2010.

[11] C. Moy and M. Raulet, *High-Level Design for Ultra-Fast Software Defined Radio Prototyping on Multi-Processors Heterogeneous Platforms*, in Advances in Electronics and Telecommunications, vol. 1, no. 1, 2010, pp. 67-85.

[12] The AAA Methodology and Syndex, *http://www.syndex.org/*, [retrieved: December, 2013].

[13] N. -ul. -I. Muhammad, R. Rasheed, R. Pacalet, R. Knopp and K. Khalfallah, *Flexible Baseband Architectures for Future Wireless Systems*, in EUROMICRO DSD, 2008, pp. 39-46.

[14] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert and R. Pacalet, *A UML-based Environment for System Design Space Exploration*, in IEEE ICECS, 2006, pp. 1272-1275.

[15] TTool, *http://ttool.telecom-paristech.fr/*, [retrieved: December, 2013].

[16] D. Knorreck, L. Apvrille and R. Pacalet, *Formal system-level design space exploration*, in Concurrency and Computation: Practice and Experience, vol. 25, no. 2, 2013, pp. 250-264.

[17] C. Jaber, *High-Level SoC Modeling and Performance Estimation Applied to Multi-Core Implementation of LTE eNodeB Physical Layer*, Ph.D dissertation, Telecom ParisTech, September 2011.

[18] B. Kienhuis, E.F. Deprettere, P. Van der Wolf and K.A. Vissers, *A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach*, in IEEE SAMOS, 2002, pp. 18-37.

 ISBN: 978-1-61208-323-0