A SOFTWARE DEVELOPMENT AND VALIDATION FRAMEWORK FOR SDR PLATFORMS

Jeroen Declerck (IMEC, Leuven, Belgium; jeroen.declerck@imec.be); Erik Umans (IMEC, Leuven, Belgium; erik.umans@imec.be); Antoine Dejonghe (IMEC, Leuven, Belgium; antoine.dejonghe@imec.be); Martin Trautmann (IMEC, Leuven, Belgium; martin.trautmann@imec.be); Miguel Glassee (IMEC, Leuven, Belgium; miguel.glassee@imec.be); Liesbet Van der Perre (IMEC, Leuven, Belgium; liesbet.vanderperre@imec.be)

ABSTRACT

The trend towards reconfigurable radio has moved the software design for radio systems to a new level of complexity. Not only have some of the hardware components been replaced by software kernels on general purpose or application specific processors, but also the system level software has moved down the latter from MAC level towards the PHY level. This paper describes the software framework used for the PHY and MAC software development, simulation and validation on the IMEC SDR platform and illustrates it with several use cases.

1. INTRODUCTION

Anything, anywhere, anytime: the query is not new. Still, offering ubiquitous wireless connectivity and seamless access to multimedia services is not yet a reality. From a terminal perspective, a major enabler of this vision is software defined radio (SDR): a reconfigurable and multi-purpose radio implementation, which should offer support for a large variety of wireless standards in a flexible and cost-effective way. As a matter of fact, the combination of the increasing need for functional flexibility in communication systems (the number of wireless standards to be supported is large - Fig. 1 - and can be expected to grow) and the exploding cost of system-on-chip design will make implementation of wireless standards on such reconfigurable radios the only viable option in the coming years [1], [2].

Due to the evolution from hardware to software processing and control, the software design of radio systems has moved to a new level of complexity. On the one hand, the hardware radio has been replaced by several general purpose or application specific processors running optimized kernels that fulfill one task in the data processing chain and can be changed at runtime to support different wireless standards. On the other hand, the system software that used to delegate the data between the host system and the hardware radio adding Medium Access Control (MAC) has moved to a lower level. On a SDR this system software is now also responsible of routing the data through the different hardware cores towards the antenna interface. As a result, the physical layer (PHY) is now partly implemented in the system software instead of solely in hardware.

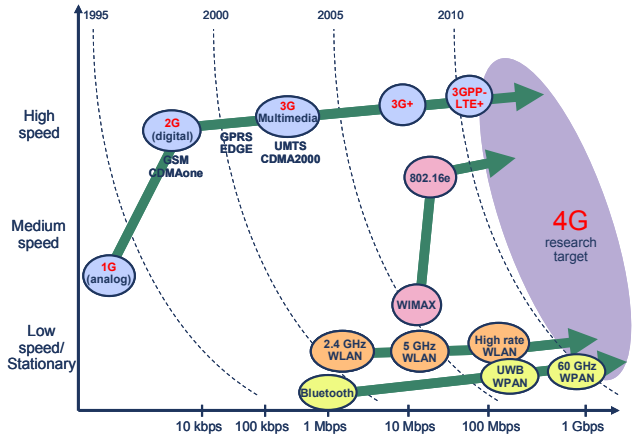

Fig. 1: The variety of wireless standards

This paper describes a software framework used for the development, implementation and validation of the system level software of an SDR platform. In section 2, the IMEC SDR platform is introduced in order to provide a concrete set of assumptions (without loss of generality). The proposed software framework is then introduced in section 3. Section 3.1 first briefly covers the component-level software design and the overall platform design/validation (as this is not the main focus of the present paper; the interested reader is referred to [3] for an extensive presentation). Section 3.2 then presents our approach for system-level software design in great details. Section 4 exemplifies the proposed framework in several use cases, illustrating its effectiveness and flexibility. This covers 802.11n PHY (section 4.1), 802.11n MAC (section 4.2), new architecture exploration (section 4.3) and energy profiling (section 4.4). Section 5 finally concludes the paper.

2. THE IMEC SDR PLATFORM

In the present section, we briefly introduce the IMEC SDR implementation, covering both a reconfigurable analog front-end and an SDR digital baseband platform.

2.1. Reconfigurable Analog Front-end

For the reconfigurable analog front-end, architectures and circuits should be designed to offer flexibility in carrier frequency, channel bandwidth and noise performance with minimal penalty in power consumption. Such a reconfigurable zero-IF analog front-end was presented in [12], covering frequencies from 174 MHz to 6 GHz, and the RF specifications of the following standards: 802.11a/b/g/j/n, 802.15.1,4, 802.16e, UMTS-TDD/FDD, HSDPA, 3GPP-LTE, DAB/ DMB/DVB-H.

2.2. Digital Baseband platform

A heterogeneous multi-processor system-on-chip (MPSOC) SDR digital baseband platform was conceived, achieving both functional flexibility and energy efficiency through opportunistic partitioning (see Fig. 2) [3], [1], [4]. This platform is referred to as BEAR: Baseband Engine for Adaptive Radio. It features the following elements:

- A 32-bit segmented Advanced Microcontroller Bus Architecture (AMBA) connecting the different cores, and two Direct Memory Access (DMA) controllers for transferring date between the different cores.
- Three Digital Front-End (DFE) cores responsible for the interface with the analog front-end, synchronization and automatic gain control upon reception of packets. This DFE includes a synchronization processor, which is a dedicated ASIP programmable in assembly. [5]
- Two baseband processors ('BB engine') for data processing, which are optimized instantiations of IMEC's ADRES core, i.e., a 2-dimensional VLIW with 4x4 functional units featuring SIMD instructions offering a high degree of parallel processing [6].
- Four Outer Modems (OMD) cores (two TX and two RX) taking care of Forward Error Correction (FEC), scrambling/descrambling and CRC calculations.
- One ARM926 processor for event-based control flow, inter-core data transfer, and MAC. This ARM processor can be considered as the main platform controller. Based on incoming interrupts, it can start/stop the other components and organize/program data transfers.

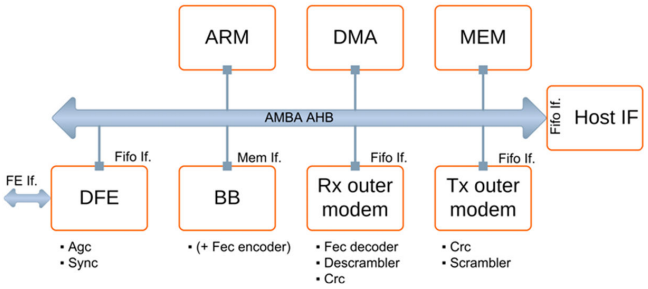

Fig. 2: IMEC SDR digital baseband platform

3. DEVELOPMENT AND VALIDATION TOOLS

We first briefly introduce our approach for component-level software design and overall platform design/validation (section 3.1). We then introduce the proposed framework for the development, simulation and validation of the system level software of an SDR platform (section 3.2).

3.1. Component-level software design and overall platform-level design/validation

Software is developed for different components on the platform (baseband processor, DFE). For the baseband processor software in particular, short deployment time yet excellent performance/power is achieved, thanks to the innovative C-compiler associated with the ADRES processor [7]. Thanks to a systematic Matlab-to-C design flow, C code running on the ADRES architecture can be obtained starting from Matlab code. We refer the interested reader to [8] for a detailed description of the associated methodology.

For the overall platform design and validation, the CoWare ConvergenSC toolset [10] is used. This toolset allows the co-simulation of hardware and software components. The different cores can be described at different levels of abstraction using different description languages (e.g., SystemC, VHDL). This enables executable models with parts at different level of abstraction and, consequently, early architecture exploration, hardware/ software verification and progressive refinement from programmer view down to hardware verification view (full VHDL model). The interested reader is once again referred to [9] for further details.

For the purpose of testing the components and the overall platform integration (e.g., verification of critical timing requirements in the data path), a hardware API was developed on the ARM processor. It is referred to as BEAGLE ("BEAr proGramming LayEr") and enables running tests on both the TLM and VHDL models.

3.2. System Software Design and Validation framework

Under system software design, we understand the design of the software that runs on the ARM processor (C programming language) and that is responsible for the implementation of the data path on the SDR for the considered wireless standards. Considering the TX path, this means providing the necessary headers to the data, programming the different components (OMD, BB, DFE), transferring all data to this components over the AMBA bus up to the DFE. In terms of OSI layering, this comes down to implementing a part of the physical layer (PHY) and the time critical-part of the MAC in software on the ARM processor.

3.2.1 Hardware Abstraction Layer

In order to enable efficient PHY/MAC development, a proper abstraction of platform specific issues is needed (e.g., register bitmaps, bus and memory addresses, interrupt handling ...). For that purpose we defined a Hardware Abstraction Layer (HAL) API, providing an interface between the platform components and all the higher level system software on the platform. The system software code only uses the HAL API to access the platform components. The HAL is implemented on the platform by using the BEAGLE API (see the platform-centric simulation in Fig. 3).

The HAL API is defined with two programming concepts in mind: Object-Oriented Programming and Data Hiding. Every platform component is treated as an object with an associated state stored in hardware, software or both. As the code is written in C, the software state is stored in a C structure. Each object is represented in the system code by its handle, a pointer to the structure storing the software state. Several methods, implemented as C functions taking the handle as argument, instruct the hardware and change its state. Using methods, the implementation details of the structure can be hidden to the system software, only exposing a pointer.

The collection of methods embodies the object HAL. The method declarations and handle data type embodies the objects HAL API. In practice it means that software components like the PHY or MAC will never access hardware blocks directly (by writing to registers for example), but will call specific methods defined in the HAL API. To call those methods, only functional information about the hardware is needed. For example, the PHY system software does not have to program all the registers of the DMA controller. Instead, the MAC will open a DMA object and call a "Dma_start()" function with size, source- and destination address as an argument.

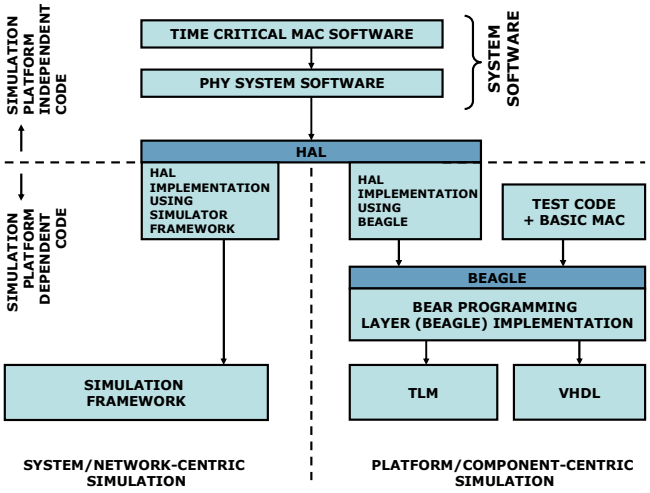

Fig. 3: System software layers (ARM processor) and their relation with the different simulation levels.

3.2.2 A System-Level Platform Simulator

Simulating the platform with the CoWare ConvergenSC design toolset is very effective to check the platform in an instruction accurate or even gate level accurate way. All data processing algorithms can be verified up to bit level. However, there are several drawbacks when moving to the design and simulation of system software. First, the simulation focuses on one single SDR terminal. So there are no means to simulate interaction between several network nodes. Second, the simulation speed is far to slow to simulate MAC and PHY system software. This mainly comes from the fact that the data processing algorithms are completely simulated, i.e. data is really modulated, demodulated, interleaved, etc... From a platform simulation point of view, this is needed; however this is an unnecessary overhead for simulating the system software.

These two drawbacks lead us to the development of a new platform simulator explained and illustrated in more detail in the remainder of the present paper. The level of abstraction for this new simulator is illustrated on the left side of Fig. 3. The idea is to mimic the behavior of the HAL API towards the system software, meaning that from the system software point of view, the HAL should behave the same as it does when simulating the system software on the TLM or VHDL model. The proposed simulator is network transparent. It runs on a host as a service and accepts socket based connections. Several simulated systems can then connect to this simulator (referred to as the Simulator Server) and use the three services it provides: a time model service, an ether bandwidth service and a logging service (see Fig. 4).

Time service: The first service is the time model that the simulation server provides. The server offers a uniform way to register events that take up a specific amount of time, called the time model (Fig. 4). When implementing the HAL for the DMA controller for example, it is known that a transfer of X amount of bytes will take ΔT amount of time. So when implementing the "Dma_start()" function that is part of the DMA HAL API as mentioned before, at some time an event is registered to the system simulation server that takes ΔT amount of time. When registering this event, the model also provides a callback function that will be called when the ΔT time has elapsed. After ΔT time, this callback function is called by the system simulator server and the DMA model signals to the higher software layer (the PHY) that the transfer was done. This way the DMA HAL API has the same behavior to the higher software as on the real platform, i.e. it takes the same amount of time to do a DMA transfer. For all components (BB, DFE, OMD…) the corresponding HAL implementation makes use of this event registering to model their behavior towards the system software layers.

The timings used when registering events are obtained from the platform simulations in ConvergenSC (see section 3.1). This way, all the timings on the system simulated platform closely matches the timings of the "real platform". Note finally that this timing API of the simulation server is used by all the simulated terminal/node instances to register their events. That way several terminals/nodes can be simulated in a synchronized way.

Ether Bandwidth Service: with the time model of the simulation server it is possible to synchronize several simulated terminals/nodes. However, as long as they do not share a medium and exchange date, the PHY/MAC system software cannot be properly simulated. This medium connection is needed in the HAL implementation of the DFE where data is normally send to the analogue front-end and send over the air. To simulate this, the simulation server offers the 'ether bandwidth' service. When the HAL DFE model sends data to this service, the other simulated nodes/terminals will receive this data. The service also includes features for carrier sensing, instantiating several antennas', transmitting on several channels and collisions. Furthermore, the ether bandwidth model can be extended by adding a channel model in the form of a dynamic loadable library. This way, the system software can be simulated under different channel conditions.

Logging Service: when running system-level software on the platform with the CoWare ConvergenSC tools, the main visualization and debugging tool is the ARM debugger console. To facilitate development/validation, the simulation server provides a logging interface (see Fig. 4). Log messages from all simulated nodes/terminals are collected in the log reflector of the simulation server.

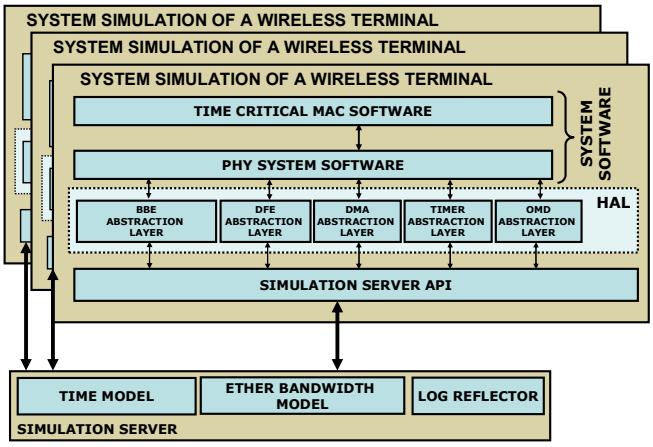

Fig. 4: Simulating several terminals using calls to the system simulation servers API

Logging applications can then connect to the server, collect all the log messages and process them for display. The simplest logging application used is a small routine that dumps the log messages to the screen or in a file. The result is comparable with the fprintf statement in C, with the difference that the output of all terminals is centralized. For more advanced logging purposes, an application was written in the JAVA programming language, using an open source charting library [11]. This application can interpret some predefined log message formats and display them on a graph in a window.

4. USE CASES

The proposed system simulation server and its API were used to implement the HAL and model all the platform components in a behaviorally-correct way. The server, the simulated nodes and the JAVA application can run on different WINDOWS/LINUX/UNIX hosts in a network, but can also run on one single (laptop) PC. This allows making a trade off between simulation speed or demo purposes. The present section exemplifies the proposed framework in several use cases, illustrating its effectiveness and flexibility.

4.1. 802.11n PHY

Based on the proposed framework, the first layer of system level software (the part of the PHY that is implemented in software) was developed, considering the 802.11n standard as an example. This software controls the reception/transmission of a data packet from the antenna interface to the MAC layer and vice versa. This data path is shown in Fig. 5 for the transmission of a packet. The PHY performs several tasks. It takes the payload data from the MAC layer and constructs the signal field and preamble. Then it programs all the platform components and delegates

the data up to the DFE over the AMBA bus using the DMA controllers.

Fig. 6 shows the simulation results obtained from the JAVA application for the transmission of a packet. The above subplot (DFE.T2) shows the state of the second DFE tile (upper bar) and the state of the analog front-end (lower bar). The state information of the analog front-end is shown through de DFE, as the DFE is the interfacing block for the analog front-end. The second subplot (DMA2.1) gives information about channel 1 of DMA controller number 2. High indicates when it is transferring data over the bus. As this is a simulation result for MIMO, there are two data streams going to two antennas. The third subplot (DFE.T1) gives the DFE and antenna information for the other DFE tile and the front-end antenna it is connected to. The fourth subplot (BB) shows the status information of the BB. The step function indicates when it actually processes data. The upper bar shows whether it is modulating of demodulating and the lower bar shows the modulation order (BPSK, QPSK). The first step shows the modulation of the signal field in BPSK and the next steps show the modulation of the payload data in QPSK. The fifth subplot shows the hardware timer information. When the stop function is high, the timer is counting, the arrows pointing downwards is when the timer elapses. The PHY software uses this timer to trigger the start of the transmission as the arrow coincides in time with the analogue front-end starting to transmit data.

4.2. 802.11 MAC (DCF)

After the development of an 802.11a and 802.11n PHY system software layer, the focus was on the next OSI layer, i.e., the 802.11 MAC layer, but only the time critical part. The Distributed Coordination Function (DCF) was developed only using calls to the underlying layer, i.e., the 802.11a PHY software layer (see section 4.1).

Fig. 7 shows some simulation results for one terminal. The lower subplot in the picture shows the status information of the PHY layer, or in other words the 802.11a/n PHY. The 'W' states that it is in a 'Wait' state, i.e. not receiving a packet, nor transmitting one. First the terminal receives two packets, indicated by the 'Rx' and then it gains access to the channel and transmits a packet, indicated by the 'Tx'. Shortly after the transmission it receives a very small packet which is the acknowledgment that was send by the addressed receiver of the packet within the Short InterFrame Space (SIFS). Thereafter, it again transmits a packet, but after the reception of the acknowledgment, the terminal goes into the 'Cs' state which stands for 'Carrier Sensing'.

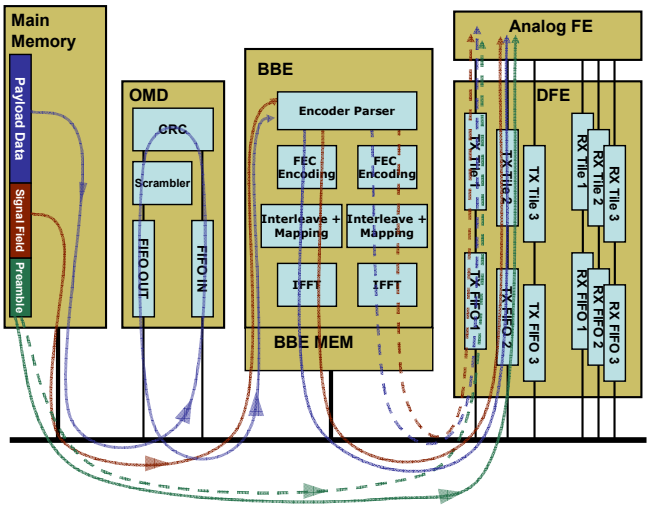

Fig. 5: The data path through the main platform components for the transmission of an 802.11n data packet.

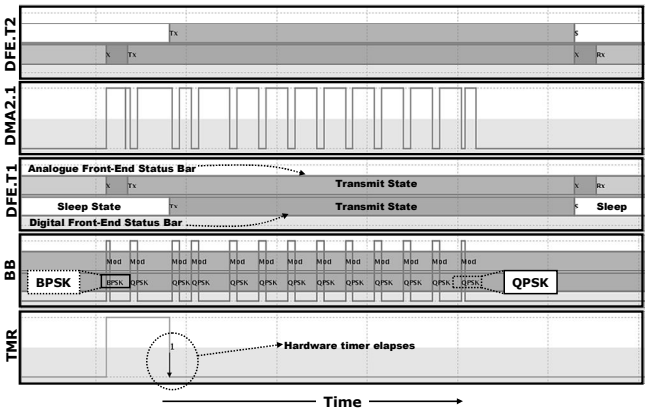

Fig. 6: Simulation results for the transmission of a data packet in MIMO mode.

This means that the PHY detected an error during reception and lost synchronization with the channel. It then starts sensing the channel (the 'Cs' state) until it senses that the channel is not occupied by another terminal any longer. When the channel is freed, it backs off from the channel by a time indicated by the MAC layer. The upper subplot indicates requests from the MAC layer to the PHY to back off from the medium following the 802.11 standard.

4.3. Platform Exploration

The system simulator can also be used to perform a platform exploration for future SDR platform candidates. This means replacing the current IMEC SDR platform template by future/generic platform candidates.

As an example the HAL layer and PHY layer software were replaced by two C++ modules that interface with the above described 802.11 MAC as a higher layer and the system simulator API as a lower layer (see Fig. 8). One C++ module performs the function simulation of the PHY software on packet granularity. The other module simulates timing by modeling the packet processing steps as task graphs and by executing them on a target platform.

Since the platform exploration uses the same 802.11 DCF MAC software than the other use cases, it can share existing test benches for traffic generation. Interfacing the system simulator API enables the reuse of the ether bandwidth model as well as the existing JAVA application for visualizing the log messages. The use case of using the Simulator Framework for platform exploration clearly shows that the framework is flexible enough to allow replacement of components with ones that are implemented in a different style and which are modeled on a different granularity.

4.5. Energy Profiling

In the implementation of the HAL on the system simulator, the code sends log messages to JAVA application about the energy it has consumed. The graphical logger then uses this energy information to generate plots about the power consumption of the system and its components.

Fig. 9 illustrates this for the BB and a DFE tile. As the power consumption of the IMEC SDR platform is very dynamic, depending of the settings of the different components, this allows viewing the power for different scenarios, i.e. receiving terminal, a transmitting terminal, a busy terminal, an idle terminal etc…

5. CONCLUSION

Software design is one of the key challenges in implementing wireless access schemes on SDR's. This paper described the different levels of software design, simulation and validation: component level, platform level and system level. It introduced a simulation framework that allows the design, simulation and validation of system level software (PHY/MAC) interacting with connected nodes (network centric simulation) while the impact on the platform components can be monitored. For the future, this framework will also be used to profile the energy consumption of SDR architectures for different wireless scenarios. It is also being used for future SDR platform exploration.

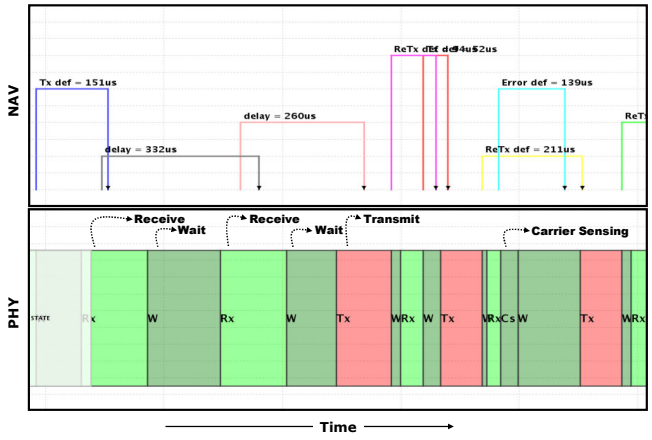

Fig. 7: Simulation results for a DCF 802.11 MAC terminal

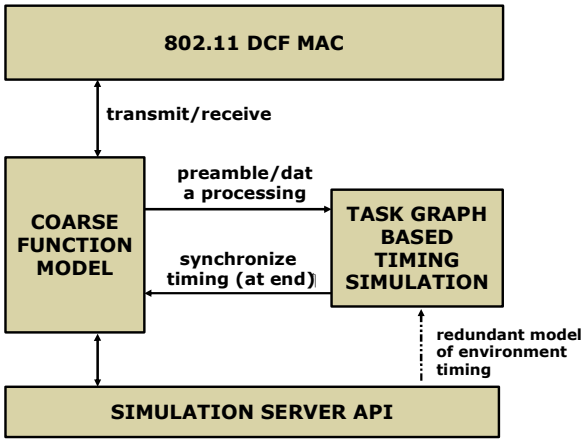

Fig. 8: Layout of a generic platform architecture simulated with the system simulation server and reusing the 802.11 DCF MAC software layer

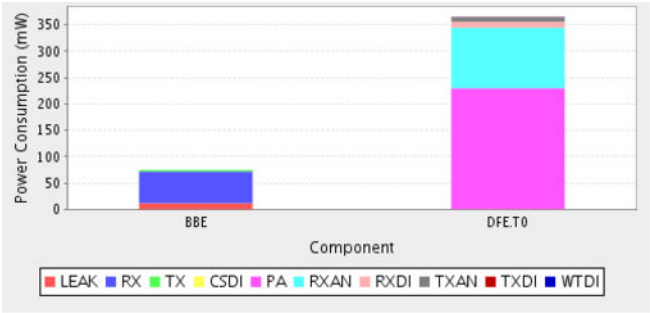

Fig. 9: Using the system simulation framework for energy profiling

5. REFERENCES

[1] A. Dejonghe, B. Bougard, S. Pollin, L. Van der Perre, F. Catthoor, " Green Reconfigurable Radio Systems: Creating and Managing Flexibility to Overcome Battery and Spectrum Scarcity," *IEEE Signal Processing Magazine, special issue on Resource-Constrained Signal Processing,Communications, and Networking*

[2] G. Desoli and E. Filippi, "An Outlook on the Evolution of Mobile Terminals," *CAS Magazine*, second quarter 2006.

[3] L. Van der Perre et Al., 'Architectures and Circuits for Software Defined Radios: scaling and scalability for low cost and low energy', ISSCC 2007, Febr. 2007, San Francisco, USA

[4] B. Bougard, D. Novo, F. Naessens, L. Hollevoet, T. Schuster, M. Glassee, A. Dejonghe, L. Van der Perre, " A scalable programmable baseband platform for energy-efficient reactive software-defined radio", *Crowncom*, June 2006.

[5] B. Bougard, L. Hollevoet, F. Naessens, A. Ng, T. Schuster, L. Van der Perre, "A low power signal detection and pre-synchronization engine for energy-aware software defined radio", *Software Defined Radio Forum Technical Conference*, Nov. 2006.

[6] D. Novo, W. Moffat, V. Derudder, L. Van der Perre "Mapping a multiple antenna SDM-OFDM receiver on the ADRES coarse-grained reconfigurable processor", *IEEE Workshop on Signal Processing Systems*, Nov. 2005.

[7] B. Mei et al., "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *IEEE Proc. on Computers and Digital Techniques,* Vol. 150, pp. 255-261, 2003.

[8] L. Van der perre, M. Glassee, V. Ramon, B. Bougard, H. Cappelle, S. De Rore, A. Dewilde, A. Folens, R. Vandebriel, T. Van der Aa, "Effecient SW desing and SW design efficiency: Fuel for Software Defined Radios", *International Symposium on Spread Spectrum Techniques and Applications*, invited paper, Bologna, Italy, 2008.

[9] A. Ng, J. Weijers, M. Glassee, T. Schuster, B. Bougard, L. Van der Perre, "ESL design and HW/SW co-verification of SDR platforms", International Conference on Hardware/Software Codesign and System Synthesis - CODES-ISSS, Salzburg, Austria, Oct. 2007.

[10] www.coware.com

[11] www.jfree.org/jfreechart

[12] J. Craninckx1, M. Liu1, D. Hauspie1, V. Giannini1, T. Kim2, J. Lee2, M. Libois1, B. Debaillie1, C. Soens1, M. Ingels1, A. Baschirotto3, J. Van Driessche1, L. Van der Perre1, P. Vanbekbergen1 'A Fully Reconfigurable Software-Defined Radio Transceiver in 0.13µm CMOS', *ISSCC 2007*, San Francisco, USA, Feb. 2007