FP7 Information & Communication Technologies (ICT)

CONSERN

Deliverable D2.2

Terminal Level Energy Optimisations Solutions

Document Number:	INFSO-ICT-257542/CONSERN /WP2/D2.2/31.05.2011
Contractual Date of Delivery:	31/05/2011
Authors:	Miguel Glassée, Jeroen Declerck... (see contributors)
Work package:	WP2
Distribution / Type:	PU
Version:	1.0 (release version)
Total Number of Pages:	71
File:	CONSERN_D2.2_v1.0.docx

Abstract: This deliverable covers the integration of energy awareness in terminal design tools at three design levels: system design, platform design and component design. The respective selected tools for this are Petri-net modelling, the SystemC and Transaction Level Modelling (TLM) based Cobra virtual Platform and the SuperESCalar Simulator (SESC). Three mechanisms for adding energy awareness to these tools are proposed.

Executive Summary

This deliverable D2.2 is the first WP2 deliverable of the CONSERN project [1] (along with deliverable D2.1 that is released at the same time (M12)). It focuses on terminal specific energy awareness. The word "terminal" is interpreted in the broadest way as a static or mobile networking node ranging from sensors over portable communication devices to access points.

First the deliverable is situated within the context of WP2 (which is on Optimisations for Energy Efficiency) and Task 2.2 (which is on and Architectural Level Energy Optimisation). This shows that this deliverable covers several of the WP2 objectives:

- Modelling tools for algorithmic solutions,
- Development of tools for modelling and management of platform energy consumption in a 'real-link' context,
- Design tools targeted towards energy efficiency.

Then it is shown that for the design of terminals, different levels of abstraction can be used for simulating terminals:

1. System simulation and design tools where there is no notion of hardware, software or components,
2. Platform simulation and design tools where the distinction between hardware and software is made and where all the components of the system/platform are visible and simulated,
3. Component simulation and design tools where a very detailed simulation and analysis of a single component is done comprising a sub-part of the functionality of the platform.

Three simulation tools are then selected, one at each level of simulation or abstraction:

1. A system level framework based on the Petri-net formalism [40],
2. The Cobra virtual platform which is a specific platform using a SystemC and TLM environment for hardware and software co-design,
3. A cycle accurate, microprocessor architectural simulation framework based on the SuperESCalar Simulator (SESC).

These three solutions however lack the necessary energy awareness necessary for the implementation of energy efficient terminals. This work describes then the technical implementation of energy awareness mechanisms into these design tools. With all these enhancements simulations results on energy efficiency can be obtained from all simulation tools adding energy awareness to the terminal design flow.

Document Revision History

Date	Version	Author	Summary of changes
22.03.2011	0.01	Miguel Glassee	Initial ToC
22.03.2011	0.02	Jeroen Declerck	Elaboration on ToC
29.03.2011	0.1	Jeroen Declerck	Updated ToC with NKUA comments
02.05.2011	0.2	Evangelos Rekkas	Added NKUA technical contribution
06.05.2011	0.2.1	Miguel Glassee	Added draft imec technical contribution
10.05.2011	0.2.2	Tim Lewis	Added TREL technical contribution
10.02.2011	0.3	Jeroen Declerck	Assigned editorships for the different sections
16.05.2011	0.3.1	Miguel Glassee	Added NKUA updates
16.05.2011	0.3.2	Jeroen Declerck	Added section 1.1 and 1.2
17.05.2011	0.3.3	Miguel Glassee	Updated schedule and expected contributions for PhC
18.05.2011	0.3.4	Miguel Glassee	Updated with TREL technical contribution and added comments for final partner's review
23.05.2011	0.3.5	Jeroen Declerck	Merged partner's review, added summary and conclusion
24.05.2011	0.3.6	Jeroen Declerck	Merged and added comments from PhC
25.05.2011	0.3.7	Jeroen Declerck	Merged reviews from NKUA and TREL
25.05.2011	0.4	Jeroen Declerck	Document ready for independent review
31.05.2011	0.4.1	Jeroen Declerck	Merged comments from Fraunhofer and Huawei Sweden
01.06.2011	0.4.2	Jeroen Declerck	Addressed comments from independent reviewers
02.06.2011	0.4.3	Jeroen Declerck	Included updates from TREL
02.06.2011	1.0	Jeroen Declerck	Release version
03.06.2011	1.1	Jeroen Declerck	Fixed broken references
07.06.2011	1.2	Jeroen Declerck	Fixed even more broken references

Contributors

First Name	Last Name	Company	Email
Miguel	Glassée	imec	miguel.glassee@imec.be
Jeroen	Declerck	Imec	Jeroen.declerck@imec.be
Evangelos	Rekkas	NKUA	erekkas@di.uoa.gr
Konstantinos	Chatzikokolakis	NKUA	kchatzi@di.uoa.gr
Tim	Lewis	TREL	tim.lewis@toshiba-trel.com

Acronyms

Acronym	Meaning
3GPP	The 3rd Generation Partnership Project [71]
ADRES	Architecture for Dynamically Reconfigurable Embedded Systems
AFE	Analog Front End
AGC	Automatic Gain Control
AGRAC	AGC and Resource Activity Controller
AHB	Advanced High-performance Bus [69]
AMBA	Advanced Microcontroller Bus Architecture [69]
ARM	Advanced RISC Machine [70]
ASIP	Application Specific Instruction-set Processor
BBE	BaseBand Engine
BiFI	Bit Fiddler Input
BiFO	Bit Fiddler Output
BPSK	Binary Phase Shift Keying
CFO	Carrier Frequency Offset
CGRA	Coarse GRain Array
CMP	Chip Multi-Processors
CR	Cognitive Radio
CRC	Cyclic Redundancy Check
DIFFS	Digital Front End For Sensing and Synchronization
DMA	Direct Memory Access
DoW	Description of Work
DRESC	Dynamically-Reconfigurable Embedded System's Compiler
DSE	Design Space Exploration
DVB	Digital Video Broadcasting [73]
DVB-T	DVB Terrestrial [73]
EDP	Energy-Delay Product
FFT	Fast Fourier Transform
FlexFEC	Flexible Forward Error Correction engine
FU	Function Unit
GSPN	Generalised Stochastic Petri-nets
HFL	Hierarchical Fuzzy Logic
ILP	Instruction Level Parallelism
IMD	Inner MoDem
IRQ	Interrupt ReQuest
LADON	Latency Aware Data Oriented Network
LDPC	Low Density Parity Check
LTE	Long Term Evolution [71]
LTE-A	LTE Advanced
MIMO	Multiple Input Multiple Outpus
MPSOC	Multi-Processor System On Chip

Acronym	Meaning
OFDM	Orthogonal Frequency-Division Multiplexing
OMD	Outer MoDem
PDP	Power Delay Product
PEP	Power-Energy Product
PHY	PHYsical layer
PIM	Processors-in-Memory
QAM	Quadrature Amplitude Modulation
RENEW	Reference Net Workshop Tool
RTL	Register Transfer Level
RX	Receive
SCO	Sample Carrier Offset
SDR	Software Defined Radio
SESC	Super ESCalar simulator
SIMD	Single Instruction, Multiple Data
SISO	Single Input Single Output
SoC	System on Chip
TINA	Time Petri-net Analyser
TLM	Transaction Level Model
TLP	Transaction Level Parallelism
TVWS	TV White Space
UML	Unified Modelling Language
VLIW	Very Long Instruction Word
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network [72]

Table of Contents

List of Figures

List of Tables

1. Introduction

As this deliverable is the first deliverable of WP2 together with deliverable D2.1, this section describes the scope of WP2 and its tasks in greater detail. It also shows how this deliverable covers several WP2 objectives and finally situates the term "terminal" used in the title of this deliverable within the scope of the CONSERN project.

The WP2 scope is considered in section 1.1. First, in section 1.1.1 it is shown how the WP2 objectives are covered by the different tasks in the WP. Then, in section 1.1.2, the technical scope of the WP is explained and shows the relation with the different tasks. Finally, section 1.1.3 describes how the three deliverables cover all the developed mechanisms and existing and future results from the different tasks.

Section 1.2 then dives deeper into the scope of Task 2.2 as D2.2 received its inputs from this task. The terminal types covered by this deliverable are then put into the objectives of the project by relating the covered terminal types to the use cases proposed in D1.1 [39].

1.1 WP2 Scope

1.1.1 WP2 Objectives

Before going in detail on the technical scope of the WP, tasks and deliverables, it is important to look at the objectives of the WP first. Therefore, Table 1-1 below lists the different objectives for WP2 as described in the DoW.

Table 1-1: The WP2 objectives as listed in the DoW [1]

Objective	Objective description
OBJ_1	Evaluation and selection of appropriate low-power protocols;
OBJ_2	Modelling tools for algorithmic solutions,
OBJ_3	Design and optimisation of protocols for energy efficiency;
OBJ_4	Interfaces and data structures towards measurement and control data for energy optimisation purposes;
OBJ_5	Development of tools for modelling and management of platform energy consumption in a 'real-link' context,
OBJ_6	Design tools targeted towards energy efficiency;
OBJ_7	Development of methods which allow monitoring and adapting the energy consumption of wireless devices radio platforms, resulting in considerable average power savings,
OBJ_8	Coupling of energy optimisation approaches at system and terminal level,
OBJ_9	Run-time calibration and self-adaptation schemes for low energy through network awareness.

Different (parts of the) objectives of WP2 are covered by the different tasks in WP2. Some objectives also have an overlap between the different tasks as they combine several results and mechanisms

obtained by those different tasks. The distribution of where all objectives are covered over the different tasks is shown in Table 1-2 below.

Table 1-2: Distribution of WP2's objectives over the different tasks

	OBJ_1	OBJ_2	OBJ_3	OBJ_4	OBJ_5	OBJ_6	OBJ_7	OBJ_8	OBJ_9
T2.1	X	X	X	X		X			
T2.2		X			X	X	X		
T2.3					X		X	X	X

1.1.2 WP2 Technical Scope

The CONSERN project aims at developing and validating a novel paradigm for dedicated, purpose-driven small scale wireless networks and systems characterized by a service-centric evolutionary approach introduced here as an energy-aware self-growing network.

The technical scope of WP2 is explained in the DoW [1] and depicted again in Figure 1-1. The split into the different tasks is done as follows:

Task T2.1 focuses on all sorts of energy efficiency related aspects at the networking level (depicted as “system wide energy optimisations” in the figure) that are applied at design time. This covers protocols, algorithms, interfaces, monitoring, sensing...

In parallel, task T2.2 focuses on energy efficiency related matters at the networking object or terminal level (depicted as “terminal level energy optimisations” in the figure) that are applied at design time. This includes design and modelling tools, as well as simulation tools for platform verification. This task runs in parallel with the first task T2.1 as they both tackle the energy efficiency at different levels of abstraction.

Task T2.3 focuses on improving the energy efficiency even further by optimizing the networking object through self-adaptation or run-time calibration schemes. These schemes together with the tools, interfaces and protocols form T2.1 and T2.2 allow the networking object to adapt at run-time towards the most energy efficient working point.

The common activity or synchronization point of the three tasks is in the interpretation of the results when all developed mechanisms, methods, tools... would be applied in the same system.

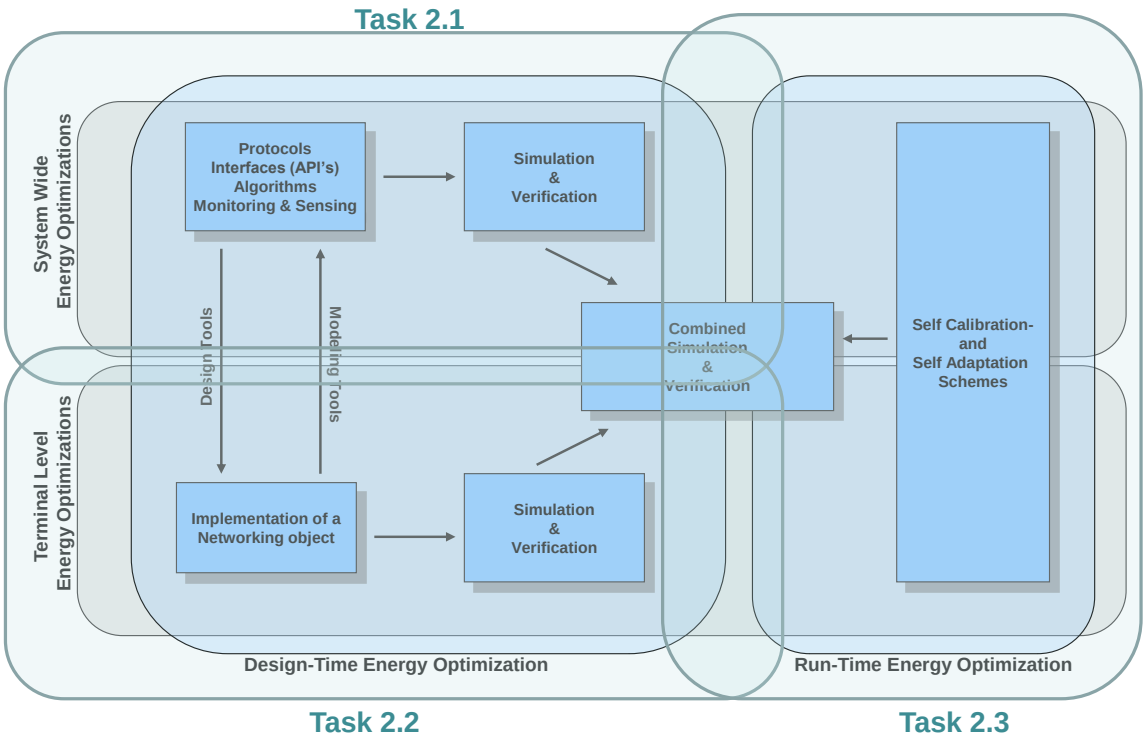

Figure 1-1: The technical scope of WP2 divided by the extend of energy awareness (transparent grey boxes), weather the energy awareness operates at design-time or run-time (transparent blue boxes) and what part the different tasks cover (transparent green boxes).

1.1.3 Scope of Deliverables

Now that it is explained how the different tasks target the different objectives of the work package, it can be shown how the different deliverables and milestones will contain results from the different tasks and, as a result, how they contribute to WP2's objectives. This is shown in Figure 1-2 below. Tasks T2.1 (networking and system level energy awareness) and T2.2 (terminal and architecture energy awareness) have been running in parallel and both cover energy awareness but at a different level. Therefore, M2.1 and M2.2 were used so far as a synchronization point between the two tasks. Then, M2.2 and the results from T2.1 and T2.2 are now used as inputs for the deliverables D2.1 and D2.2 respectively. The final work of T2.1 and T2.2 is to build further on the obtained results and to provide a converged view in M2.3 so that it can be further used by T2.3 when looking at run-time adaptation schemes. The results themselves will be described in D2.3 together with the results and mechanisms derived in Task 2.3.

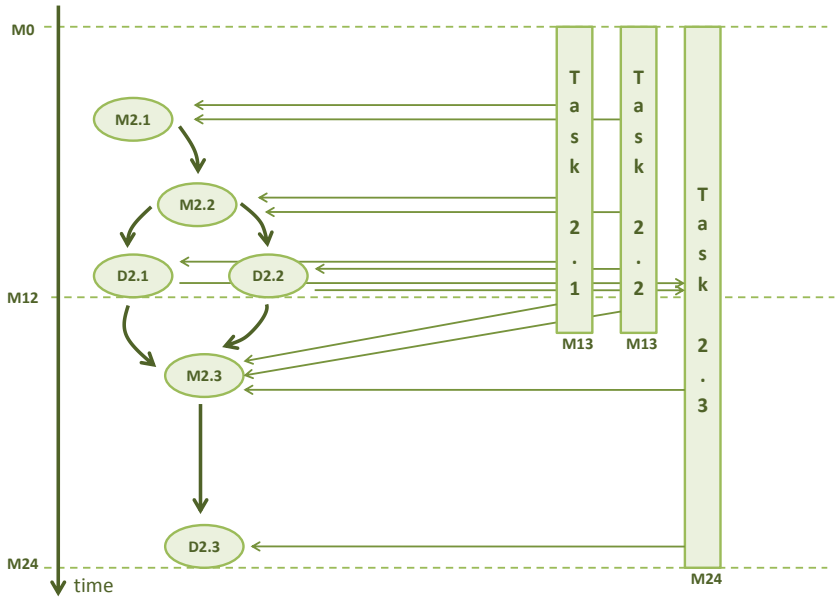

Figure 1-2: Contribution of WP2 tasks to the deliverables and milestones.

1.2 T2.2 Scope

While T2.1 targets the networking and system aspects, Task 2.2 focuses on the optimisation of terminal aspects for energy efficiency. It does this in a broad scope by addressing modelling mechanisms, design tools and simulation platforms for terminal energy awareness. These three aspects cannot be treated independently; the connections between them are illustrated in Figure 1-3.

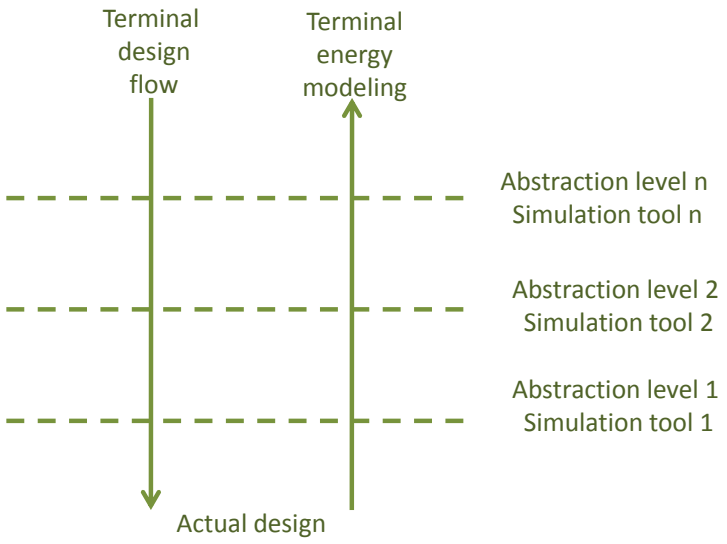

Figure 1-3: Link between simulation tools, design tools and modelling for energy awareness.

Simulation tools always require models to speed up the simulation time and offer a trade off between complexity and simulation fidelity. This trade off generally varies in the different steps of the design process and in different design aspect. In this deliverable energy consumption is the primary design aspect. An overview of the different technical contributions described in this deliverable is shown in Figure 1-4:

At the **highest abstraction level or system level** the Petri-net approach is proposed and described later in this document in sections 2.1, 3.1 and 4.1 that takes two alternative methods for deriving power consumption estimates. Both methods require a carefully designed Petri-net model of the device (such as in Section 4.1.1), annotated with energy consumption information that needs to be measured experimentally, simulated from more detailed modelling or estimated by other means.

The first method (Section 4.1.1.1) uses verification analysis techniques to derive theoretical power estimates.

The second (Section 4.1.1.2) uses a direct execution (or simulation) approach to compute these.

In **the middle level or platform level**, the Cobra virtual platform simulation framework [68] is situated. It is an evolution of the initial BEAR platform [3] and simulated using Transaction Level Modelling (TLM) and SystemC offering a hardware and software simulation environment for a complete Software Defined Radio platform (SDR). In the context of this deliverable, the ready-made hardware (or platform) is used and the focus is on the software development part and extensions towards energy awareness.

By modelling power consumption of the several components based on earlier designs an accurate relative energy consumption number is obtained for the most power hungry components (see section 3.2).

These numbers are then used in the TLM simulations to obtain a very detailed energy profiling of the system allowing optimised algorithm and software design (see section 4.2).

At **the lowest or component level** there is a simulation framework that is based on the SESC simulator. SESC is a microprocessor architectural simulator and it is mainly related to the hardware design flow. However, different software implementations may be simulated in SESC by means of binaries (configuration files) with an aim to study the tradeoffs between hardware and software implementations. Still, however, SESC remains a hardware simulation/design and validation tool rather than a modelling tool. This simulation framework considers overall hardware performance. However, in the CONSERN WP2 scope where focus is on energy accuracy power consumed in different hardware parts during various processes, such as fetching, issuing, executing are considered. Time is also taken into account in terms of execution cycles, simulation time and execution time in real hardware. At the end, functional accuracy is also considered.

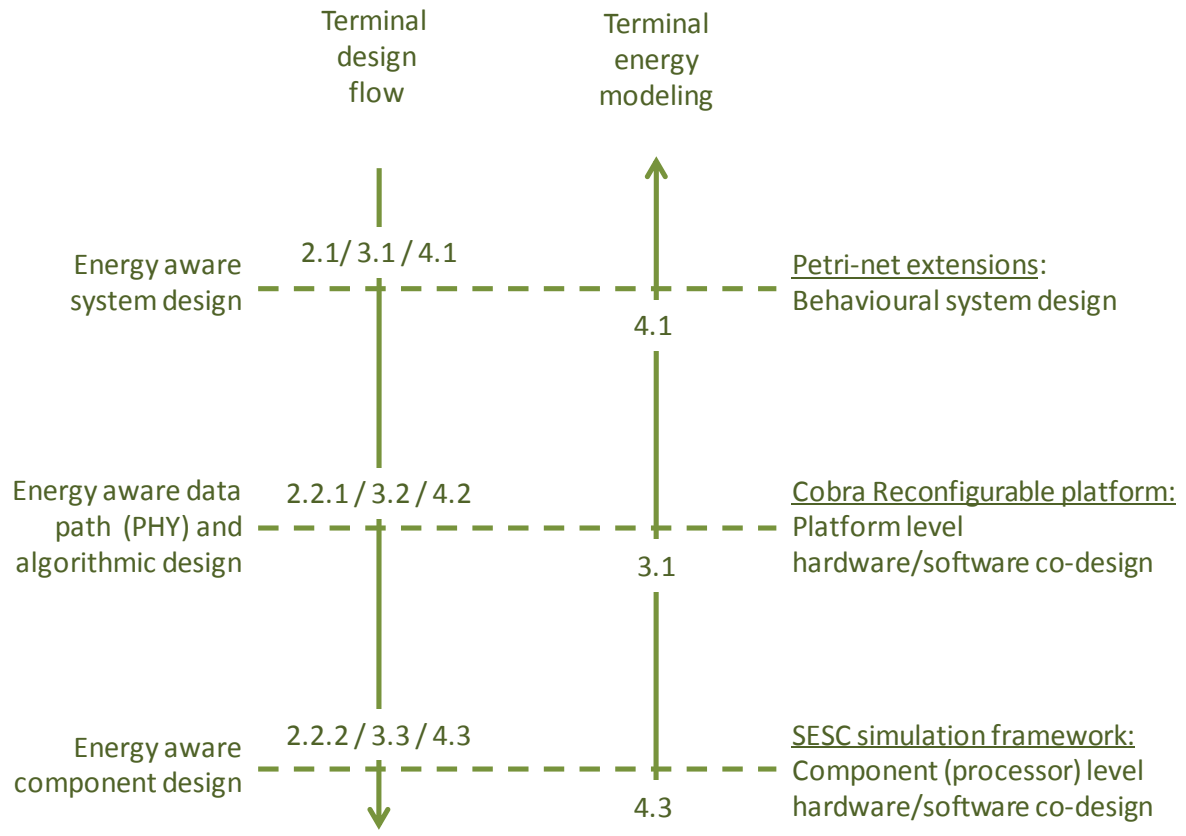

Figure 1-4: The different abstraction levels in the design process linked with the different technical contributions throughout this deliverable.

1.3 Addressing the Objectives in the Deliverables

Now that the scope of both this work package (section 1.1) and Task 2.2 (section 1.2) is explained and to which deliverables it will forward its work (Figure 1-2), the coverage of the objectives (presented in Table 1-2) by the several deliverables will be discussed.

1.3.1 Task 2.2

When looking at Table 1-2 it can be seen that Task 2.2 addresses objectives 2, 5, 6 and 7:

OBJ_2: Modelling tools for algorithmic solutions

This objective is partially covered by this deliverable especially by the work on the Petri-net extensions as this covers the higher layer in the design tool chain. Especially sections 3.1 goes deeper into this. A lot of algorithmic solutions are however designed at the network level rather than the terminal level. Therefore, this objective is also covered by Task 2.1. Most of the work on this can be found in D2.1 [38].

OBJ_5 Development of tools for modelling and management of platform energy consumption in a 'real-link' context

This is one of the main objectives related to Task 2.2. The work presented in this deliverable on the Cobra virtual platform is all about a tool for the modelling of the energy consumption in a platform (see section 3.2). Under real-link context it is understood that the information on energy consumption is available not only in the tool (this deliverable) but also to the platform when actually

running (called at run-time in the context of embedded systems). This part of the work will built further on the work presented in this deliverable and will be presented in deliverable D2.3

OBJ_6 Design tools targeted towards energy efficiency

This objective is a main objective for both Task 2.1 and Task 2.2. All the results on this objective are presented in this deliverable for terminal centric tools (sections 2, 0 and 4) and in deliverable D2.1 for network centric tools.

OBJ_7 Development of methods which allow monitoring and adapting the energy consumption of wireless devices radio platforms, resulting in considerable average power savings

This objective is covered both by the work in Task 2.2 and Task 2.3. It builds further on the methods developed in this deliverable and results will be presented in deliverable D2.3

1.3.2 Task 2.1

When looking at Table 1-2 it can be seen that Task 2.1 addresses objectives 1, 2, 3, 4 and 6:

OBJ_1 Evaluation and selection of appropriate low-power protocols

The selection and evaluation of appropriate low-power protocols are reported in D2.1 [38]

OBJ_2 Modelling tools for algorithmic solutions

See section 1.3.1 above

OBJ_3 Design and optimisation of protocols for energy efficiency

This is one of the main objectives for Task 2.1 and is reported in D2.1 and will be reported in D2.3 for the remainder and end results of the work

OBJ_4 Interfaces and data structures towards measurement and control data for energy optimisation purposes

This is also one of the main objectives for Task 2.1 and is reported in D2.1 and will be reported in D2.3 for the remainder and end results of the work

OBJ_6 Design tools targeted towards energy efficiency

See section 1.3.1 above

1.3.3 Task 2.3

OBJ_5 Development of tools for modelling and management of platform energy consumption in a 'real-link' context

See section 1.3.1 above

OBJ_7 Development of methods which allow monitoring and adapting the energy consumption of wireless devices radio platforms, resulting in considerable average power savings

See section 1.3.1 above

OBJ_8 Coupling of energy optimisation approaches at system and terminal level

This objective is a coupling of the work conducted under Task 2.1 and Task 2.2. It is the final work done in these two tasks and will start after deliverable D2.1 and D2.2. Results will be presented in D2.3.

OBJ_9 Run-time calibration and self-adaptation schemes for low energy through network awareness

This is one of the main objectives of Task 2.3. The major part of the effort is done in the second half of the project and will be reported in D2.3

1.3.4 Summary

Table 1-3 below summarizes this section and gives an overview of where in the deliverables the WP2 objectives are addressed.

Table 1-3: Overview of which WP2 deliverables tackle what WP2 objectives

	OBJ_1	OBJ_2	OBJ_3	OBJ_4	OBJ_5	OBJ_6	OBJ_7	OBJ_8	OBJ_9
D2.1	X	X	X	X		X			
D2.2		X			X	X			
D2.3			X	X	X		X	X	X

1.4 Refinement of selected UCs identified in T2.2 scope

In WP1, a set of use-cases were defined in the first deliverable D1.1 [39] covering the scope of the project and offering an application area for the mechanisms derived in the project. To situate this deliverable D2.2 within the project's scope Table 1-4 below lists the use cases where a terminal is a key part. The definition of the term terminal is interpreted in the broadest sense as any networking node, mobile or static that is part of a bigger communication system. Narrowing this down to the use cases we obtain the following as terminals types: relay nodes, sensors, access points, femtocells, base stations, antenna nodes and mobile devices.

Table 1-4: Relevant use-cases and corresponding terminal types. Use cases defined by partners that are not committing to WP2 are greyed out.

Use Case	Partner	Title	Terminal Types
UC-01	OTE	Energy Optimization in a moving vehicle with capacity and coverage limits.	Relay nodes, sensors
UC-02	NKUA	NKUA Energy Optimization in an Office environment under coverage constraints.	Access points, femtocells, sensors
UC-03	NKUA	Energy Optimization for Self-Growing Office environment under coverage and capacity constraints.	Access points, femtocells, sensors
UC-05	HWSE	Switch on-off of nodes for Energy Efficiency in Heterogeneous Networks	Base stations
UC-06	HWSE	Cooperative DAS nodes configuration.	Base stations, Antenna nodes

UC-07	HWSE	Cooperative relay for Energy Efficiency.	Base stations, relay nodes.
UC-08	Fraunhofer	Energy-aware end-to-end delay optimization.	Sensor devices, Access points
UC-11	IBBT	Energy optimization of co-located wireless networks in a home/office environment.	Mobile devices, Access Points.
UC-12	IMEC	Self-adaptation of a reconfigurable wireless terminal.	Mobile terminals, Access points
UC-13	TREL	Home Monitoring Energy Optimization.	Sensor devices, Access points.

For the **Petri-net extensions**, covering the system design layer as discussed in the previous section, both techniques are general purpose and applicable to all of the terminal types mentioned in Table 1-4, but some constraints may make either more appropriate. Verification analysis will handle simpler, looped behaviour that may make modelling embedded sensor devices a more appropriate target. This is why this application is specifically targeted by the techniques in this deliverable. More complex event-driven algorithms that reside in access points or mobile terminal devices will not be so easily analysed by these techniques, yet should still yield useful simulation estimates from the direct execution approach.

The **Cobra reconfigurable radio platform** and the proposed energy annotations is the most applied technique of the three mechanisms described in this deliverable. The platform itself as it will be introduced in section 2.2 is a single System on Chip (SoC) solution and proofed already that both 3GPP-LTE and 802.11n WLAN physical layers (PHY) can run on it [68]. Moreover, it should also be feasible to implement other OFDM bases standards as the platform is almost completely software defined. In that sense the platform can be mapped to any terminal type as long as it is OFDM based. The power annotations introduced in sections 3.2 and 4.2 are specifically interesting for battery operated devices as they allow for example to switch between different signal processing algorithms within the same standard depending on the energy constraints of the system.

Finally, at **the lowest layer** where energy-aware decision making at the terminal level is covered (see Section 0), optimization techniques for energy consumption during the decision making process for Cognitive Radio (CR) devices are considered. Such devices may be, for example, LTE terminals (UEs in the LTE terminology) acting as secondary users in the unlicensed band (e.g. TV White Space). Therefore, the mechanism described in this section and the simulation results further provided in section 4.3 refer to mobile terminals. It is worth mentioning that the simulation framework used for the appropriate power validation of this mechanism, described analytically in section 2.2, is based on the SESC simulator. Extensions and modifications on this simulator in order to capture with high level of accuracy the power consumption of such a type of mobile terminal are described in detail in the later referred section.

However, it should be noted that, specific alternative modifications on the hardware configurations of the SESC simulator enable the deployment of this energy-aware decision making mechanism in other types of terminals such as sensor nodes. This is due to the broad, but still accurate, characteristics of this simulator that permit the use of multiple configuration files describing various hardware parts targeted for simulation. Thus, providing as input processor architectures with certain parameters each time, in the appropriate configuration files different types of terminal can be simulated and validated in depth in terms of power efficiency.

1.5 Further Outline of the Deliverable

As already mentioned in section 1.3.1, the main objective covered in this deliverable is the integration of energy awareness in terminal design and modelling tools. Section 1.2 introduced three levels of design and simulation tools where 1 tool was depicted at each abstraction level. Section 2 will explain these three existing design tools in greater detail and why they were selected. At the end of the section, the shortcomings towards energy awareness are identified.

Section 3 then covers the actual mechanisms that are developed in the course of the CONSERN project and section 4 shows the achieved results. The deliverable ends with a conclusion in section 5.

2. Simulation Framework

This section describes in detail the selected terminal design tools at the three different abstraction layers defined in section 1.2. First, the choice for the specific design tool is discussed, then the tool itself is explained in greater detail and finally the shortcomings towards energy awareness are given, introducing the actual mechanisms developed in the course of the CONSERN project that are covered in detail from section 3 onwards.

The tools under consideration are:

1. System level: Petri-net Modelling in section 2.1,
2. Platform level: The Cobra virtual platform in section 2.2.1,
3. Component level: The SuperESCalar Simulator (SESC) in section 2.2.2.

2.1 Petri-net Modelling Approach

This section covers the terminal aspects to the Petri-net energy modelling work. Here the terminal aspects relate to the individual operation of sensor devices, and aggregator/servers as independent entities that want to optimise their power consumption individually. This is done this principally from the perspective of design time analysis, but we can verify these estimates by direction execution (simulation).

2.1.1 Design Tool Choice

First, the rationale for using this formalism is considered. The requirements for the formalism were as follows:

1. There is considerable benefit from using design methods that permit a degree of verification and correctness to be proven,
2. The formalism needed to allow simulation of models, and composition of many models into realistic sized networks,
3. Generation of implementation code (for testing on a real sensor platform) was required,
4. Allowed power consumption to be estimated via simulation, and by design-time static analysis.

Considerable expertise had been invested in the Petri-net approach, and a tool-set had already been developed which covered the first three of the requirements above. However, a number of other formalisms were explored in the initial phase of this project to discover a formalism that also allowed power consumption handling. Three possible alternatives, Live Sequence Charts, Hume specifications and Action systems are discussed here.

Message Sequence Charts are a formalism that specifies scenarios as sequences of messages passing between objects. This approach is common for generalizing system use-cases as part of a formal specification process. The extended version of this approach, called *Live Sequence Charts* (LSC) allows for the specification of *anti-scenarios*, sequences that are forbidden to occur within a correctly operating system. This extends the expressive power of the formalism, but at the cost of verification ability. The work in [78] applies LSC to analysis of embedded systems, by using a method of converting LSC representation to Petri-nets which then permits verification analysis to be performed. This means that properties such as deadlock freedom, liveness, reversibility and boundedness can be checked statically, but at the cost of having an additional conversion phase so a Petri-net based approach.

Hume [79] is a domain-specific language for designing embedded systems. It does this by way of a two-layered approach to system specification. A *coordination layer* is based on a finite-state-automaton formalism, whilst an *expression layer* uses a strict pure-functional language to manipulate expressions. This (in principle) allows the coordination layer to be completely verified statically; whilst the expression layer can rely on the strict resource bounding that functional specification can promise. Hume specifications are used in [80] to design control software for embedded systems, and analysis can be performed statically to check properties such as total execution time, and bounding the size of both the stack and heap. Note that both the Hume and LSC approaches avoid explicitly tackling power consumption issues, although it may be possible to add such extensions to either formalism.

Action Systems are an approach that recursively builds a system by composing sub-actions in a number of different ways, including operators for sequential composition, guard conditions and non-deterministic choice. Such a system does not include any notions of resource consumption. In [81] actions systems are extended to allow power consumption models to be included, however the applications are small, and may not scale to the sizeable, practical examples aimed for in CONSERN.

Since there was little existing support for power consumption analysis within these formalisms, the approach taken within CONSERN is to take advantage of the existing tools and expertise we had in Petri-net modeling and make the addition of power analysis one of the main goals of this part of Task 2.2.

2.1.2 Petri-net Formalism

The driving idea in this work, and the reason for considering the Petri-net formalism, is the idea that the active components of a net (the transitions) can be assigned a representation of energy cost, and the subsequent analysis of a net can then be used to determine the cost of the particular design and, indeed, to help guide the design process toward more energy efficient solutions.

Originally defined in the 1960s by (the now sadly departed) Carl Adam Petri in his Ph.D. thesis [40], the Petri-net formalism has subsequently accrued a massive international community. Petri-nets, despite being nearly half a century old, are still relevant and applicable in computer and communications systems today, and are being used and applied in a variety of fields (e.g. from routing protocols for mobile ad-hoc networks [41] to railway network control [42]) as well as having worked themselves into the industry standard Unified Modelling Language (UML) in the form of Activity Diagrams [43].

The fundamental Petri-net formalism [40] provided a means of describing a system in such a way that its inherent parallelism, asynchrony and indeterminism is apparent. A Petri-net is a bi-partite multigraph comprising places (graphically represented as circles, see Figure 2-1), transitions (rectangles) and arcs (the lines between them) joining the places and transitions. An arc can only join a place to a transition, or vice versa—an arc cannot connect a place to a place, for example. Arcs can be weighted, a shorthand notation for multiple parallel arcs, whereby *n* arcs can be replaced by a single arc annotated with the number *n*.

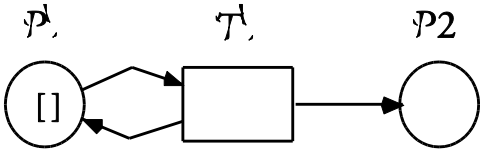

Figure 2-1: Petri-net Example.

Tokens are distributed across the places in the net, and are often, but not always, represented as dots—in the representation in Figure 2-1, square brackets are used because tokens can be much more complex in that particular extended form. The presence of tokens at a place is typically interpreted as the availability of a resource.

If tokens are present at all of the places connected to a transition (allowing for the weighted arcs mentioned above), then the transition may "*fire*" (the term adopted in the literature, paralleling the firing of synapses in the brain), consuming one token (per single, non-weighted, arc) from all connected input places and depositing tokens in all connected output places (again, one per single arc). If more than one transition is enabled, the ordering of transition firings is not defined, and some may never fire at all. Note that there is no "conservation of mass"—transitions can consume or create as many tokens as necessary.

The distribution of tokens at any time is termed the marking. The initial marking of places with tokens determines the initial conditions of the net, i.e. which resources are available at the outset on initialization.

This mathematical basis of Petri-nets coupled with the graphical form opens a world of possibilities largely unavailable to many other FDTs and modeling languages. Two opportunities arise as a result of having a strongly defined semantics behind the intuitive graphical form. Firstly, one of the key criticisms often directed at FDTs is the lack of close coupling between the formal specification and the final delivered program [44]. With the well-defined semantics of the formalism, Petri-nets can be used (with appropriate underlying interpretation and compilation to executable code) as a high-level programming language, with no need for (error-prone) manual intervention to get from the Petri-net specification to the executable software: that is, it can be treated as both the program and the specification. This is valuable, for, to quote P.J. Davis [44], "*The program itself is the only complete description of what the program will do.*"

Secondly, the algebraic, mathematical representation allows rigorous analysis of critical properties through both systematic enumeration of the states of the system and through the application of well-established mathematical techniques from linear algebra and graph theory to formally prove the existence or absence of properties.

The rest of this section describes the syntax of Petri-nets and interesting and some useful extensions to the basic Petri-net formalism from a high-level. Interested readers are recommended to study Bause [49], Jensen [52], Murata [45] and Peterson [61].

2.1.2.1 Petri-net Syntax

Algebraically, the basic Petri-net can be described as a four-tuple, comprising **P**, the (finite) set of places, **T** the (finite) set of transitions (these two sets being distinct), the incidence function **A** and the initial marking $\mathbf{M_0}$.

These elements can be conveniently represented in matrix form, with **P**, **T** and $\mathbf{M_0}$ being vectors. In the case of vectors **P** and **T**, two notations are used: either the vector lists the logical identifiers of each place (e.g. P1, P2...) or transition (respectively), or, in the numerical representation, a vector element of 1 indicates the existence of a place or transition. In this latter case, **P** is represented as a vector with all elements equal to one and as many elements as there are places; the logical names of the places are then recorded separately or implicitly named in sequence if required (*mutatis mutandis* for vector **T** and transitions).

It is possible, by applying techniques such as Murata's state equation [45], to mechanically explore the possible state space of a given Petri-net. Furthermore, it is possible to identify and verify core "safety" properties that a net satisfies to guarantee certain types of required behavior.

For example, a Petri-net is (informally) defined as 'bounded' if an upper limit exists on the number of tokens that can accumulate in any place [[45], [46]] and defined as 'deadlock-free' if the system never gets into a state from which it cannot escape that is not a final state (if the net is intended to terminate—a cyclic net that should proceed indefinitely has no final state) [[47], [48]].

A Petri-net is said to be *safe* if no more than one token can accumulate in any given place (this is then also known as a 1-bounded net—a more general condition is *k*-boundedness for an integer *k*): this can have a direct interpretation to the amount of resources consumed by an implementation [[45], [49]], for example, representing the number of data packets that have to be stored or buffered.

A Petri-net is said to be *live* if there is no state in which *all* transitions will be unable to fire given some sequence of firings. Each transition can have one of a number of forms of liveness, from L_0-live (deadlocked), L_1-live (can potentially be fired at least once) through to L_4-live (no deadlock, progress is always possible apart from in any final state) [45].

Other properties of note include "properness" (the ability of a net to return to its initial marking, also known as "reversability") [[45], [50]], "fairness" (several definitions exist in the literature, one being the idea that a pair of transitions are in a "bounded-fair" relationship if there is a bound or limit on the number of times that one of the pair can fire without the other firing), and "termination" (if there is a final state, the net will always get there eventually)—in the case of there being no final state, then the converse property is that of "cyclic behavior", i.e. that the net can progress (i.e., execute) indefinitely.

Standard Petri-nets as defined so far can be usefully employed, for example, in the modeling and verification of a transport protocol [51]. Nonetheless, Petri-nets can be extended in a number of ways to increase their expressive power, as described in the following sub-section.

2.1.2.2 Extended Petri-nets

Petri-nets have been extended in a number of directions, some offering just short-hand forms for much more complex models, some introducing new functionality and expressive power. We discuss these approaches first, before we cover our own extensions to support power estimates.

The token in the original Petri-net formalism was a simple atomic entity that is indistinguishable from any other token in the net. Jensen's work [52] changes tokens into distinguishable typed objects (known as *colored*[1] tokens) and the corresponding nets then become known as colored Petri-nets.

Tokens in colored Petri-nets become associated with a particular data value, termed "color", and can be complex structures (e.g. a tuple or record with one element being an integer, the next a text-string, and so on, where the component elements can be operated on individually if needed), and places can be restricted to only accepting tokens of particular color by defining an allowable color-set and associating it with the place in question. This is illustrated below :

[1] We adopt Jensen's original (American) spelling of the word "color" for this concept of typed Petri-nets to avoid any ambiguity with the more general word *colour*.

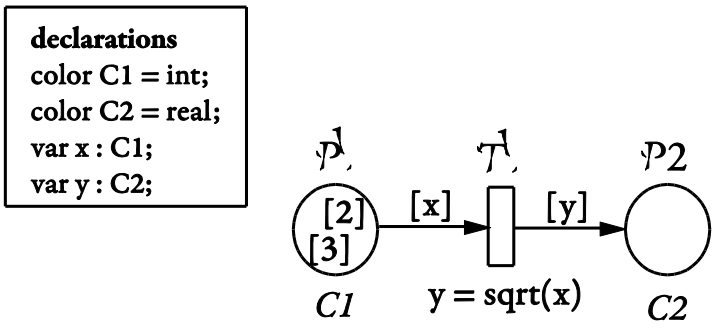

Figure 2-2: Colored Petri-net Example.

Another useful extension to tokens is to make them autonomous entities capable of processing in their own right. One approach is to allow tokens to also be Petri-nets, either directly or by reference to other nets [53] (like a pointer in a programming language, where a value or variable is described by *reference* rather than by absolute value). This allows a complete design to be built on proven sub-modules, provided there are recognized means for combining the proven sub-modules in such a way as to preserve their properties. More generally, it also facilitates the reuse of modules where possible, which can be a boon to designers.

Ramchandani introduced timed Petri-nets [55], adding a representation of *duration* to each transition, defined by a mapping (commonly denoted as Φ) from *P* to the set of real (positive) numbers. This duration changes the firing of a transition from being atomic to comprising three distinct phases: input tokens are consumed, the transition is then "in progress" for the duration and finally output tokens are deposited. Typically, the duration is noted as an inscription on the transition in question (in Figure 2-3 the example value "*duration*" is used, which could be a constant value defined elsewhere in the net, or could simply be replaced with an absolute value).

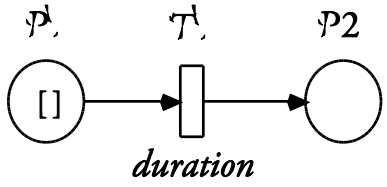

Figure 2-3: Timed Petri-net Example.

In this example, when transition T1 fires, the token on place P1 will be consumed, then the transition T1 will "run" for the indicated duration before an output token is deposited at place P2. In Ramchandani's original work, the duration could be a fixed, deterministic period of time, or be drawn from a "rectangular distribution" (i.e. the duration has a tolerance of +/- a predetermined amount, and the actual value is drawn from that range on each firing); it was also observed that more complex distributions (e.g. Gaussian) could be applied. The Reference-net formalism [[56], [57]] allows not just constants or variables to be used as durations but also more complex expressions.

Petri-nets with stochastic (probability-based) representation of durations are known as stochastic Petri-nets (SPNs) and generalized stochastic Petri-nets (GSPNs) [[49], [58]], the latter allowing transitions to be either stochastically-timed or immediate (i.e. untimed, with priority over the timed transitions). Both forms have equivalent Markov-chain representations, allowing steady-state analyses to be performed, but the additional expressive power of GSPNs permits smaller Markov-models as the immediate (untimed) transitions can be "factored out" as vanishing states. A typical graphical representation of GSPNs is shown in Figure 2-4 below.

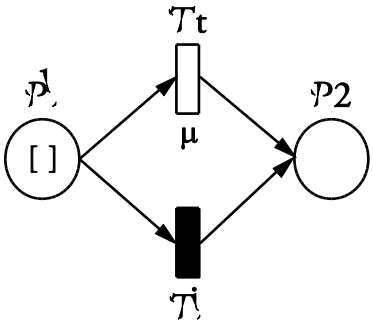

Figure 2-4: GSPN Example.

In this example, the immediate transition (labeled *Ti*) will always fire in preference to the stochastically timed transition (*Tt*, with mean rate μ) as immediate transitions have priority. The representation of immediate transitions as filled rectangles is fairly consistent in the literature (and also appear in non-stochastic timed Petri-net representations), but caution is advised due to the rich variety of Petri-net dialects.

Another transition-focused extension to the basic Petri-net form considers expanding the vocabulary of the inscription language that is available for describing actions that take place when a transition fires. Jensen's colored Petri-nets (as described above) incorporated a bespoke programming language that included declarations, variables, constants and basic expressions, and also function declarations, flow control ("if" and "case" statements) and guards (which will be described in a moment) [52]. Some formalisms take this even further: the Reference-net formalism [56] not only uses the rich Java language for its inscriptions, but also supports the full Java import mechanism, allowing the inclusion of any externally-defined Java methods.

The aforementioned guards are supported in both Jensen's colored Petri-nets and in Reference-nets. A guard is a predicate—a Boolean expression—associated with a given transition that provides an additional constraint that must be met before the transition is enabled. In the Reference-net formalism, transitions may have multiple guard inscriptions, and the transition will only be enabled if all evaluate to true (i.e. this is an implicit conjunction of guards); each inscription is preceded by the keyword "*guard*" to distinguish it from other inscriptions.

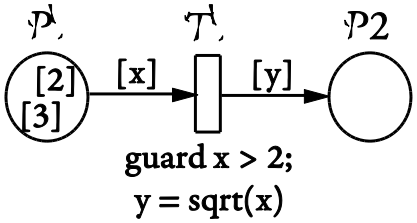

Figure 2-5: Guarded Petri-net Example.

In this example only the token '[3]' will be consumed from place P1 by transition T1 (and, subsequently, a token comprising [square-root of 3] deposited in place P2) because the guard predicate requires that tokens processed by transition T1 have a value greater than two.

A final extended net form is the Synchronous and Interpreted Petri Net (SIP-net), a safe Petri-net (i.e. with explicit upper bounds declared for the number of tokens a given place can accrue) that includes guarded transitions, synchronous firing of transitions (to a global clock tick), test arcs and inhibitor arcs (the addition of the latter permitting full Turing Machine expressiveness). The addition of synchronous transitions somewhat flies in the face of the original parallelism of the pure Petri-net,

and is not very useful for distributed systems, but can be a useful representation of some sub-systems. Enforcing resource-bounding on places (i.e. safety) can be of use when modeling embedded systems with limited resources. This survey of Petri-net extensions is representative and encompasses the major features in the field, with the caveat that the survey is not exhaustive for reasons of space and pragmatism (new extensions and variations are proposed and published frequently).

Extended Petri-nets formalisms adopting combinations of these extensions have successfully been applied to a number of areas, including modeling aspects of the IEEE 802.11 distributed coordination function (DCF) [59], and the benefits of IEEE 802.11e-style Quality of Service (QoS) prioritization in high-reliability medical environments [60].

2.1.2.3 Net Analysis

Analysis of Petri-nets usually involves an expansion of the net to transform it into a finite-state automaton, a process that suffers from state-space explosion, and an issue which has often restricted the take-up of such methods. Techniques exist to mitigate this, including structured design methodologies such as the hierarchical structure alluded to in the preceding sections afforded by Reference-nets, and the careful construction of the reachability state-space to avoid duplicate and/or equivalent markings.

An analytical approach mid-way between these two extremes is to use mathematically proven reduction rules on a net prior to analysis to reduce complex Petri-nets to more manageable sizes. There is a risk that these rules do not simplify or reduce the net to any great extent, or that the rules are not applicable given a particular net structure. However, the most important feature of any such rules is that they must preserve the properties of the original net that are of interest to the analysis stage: there is little value in analyzing a net for, say, deadlock if the reduction rules applied prior to analysis remove or affect the part of the original net that would exhibit that feature.

Many authors have proposed reduction rules that can be applied to different forms of Petri-nets. One common set of rules, defined by Murata [45] preserve liveness, safety and boundedness. The proofs of these reduction rules are well known and available in the literature for the interested reader.

Brute-force exploration of all reachable states from the initial marking of the net can be achieved through firing all enabled transitions from the initial marking, then repeating this process with subsequent markings (with tool assistance in a form of simulation, or even as a pen-and-paper exercise). The results can be represented by a labeled directed graph, comprising nodes labeled with the corresponding marking, linked with arcs labeled with the transition moving from one marking to the next. If every marking is explicitly enumerated, then the representation is usually termed a reachability tree, and, when duplicate markings are removed, a reachability graph.

A complementary (and arguably more elegant) form of analysis is that of “invariant” analysis. There are fundamental properties of a net that always hold, such as deadlock-freedom. An advantage of considering such “invariant” properties is that they do not require analysis with temporal logics to prove that an event eventually (or never) occurs. Mathematical analysis of net structures allow these properties to be derived without exhaustive simulation of the net to derive all possible states, which is the approach adopted in state-space exploration, nor any attempts to manage the size of the state-space to make the exploration viable. The disadvantage is that any deeper mathematical analysis is less straightforward than the recursive application of a simple vector equation and is therefore harder to automate.

Mathematical entities that express these key structural properties are called *invariants* of a net. There are two invariants commonly used in the literature to investigate properties of nets: place

invariants (termed S-invariants herein for historical reasons, occasionally also P-invariants in the literature) and transition invariants (T-invariants), referring to constant properties associated with places and transitions, respectively.

Once obtained, there are numerous applications of net invariants, such as determining the maximum marking on a given place at any time (a “high tide mark”).

2.1.3 CONSERN Contribution

Although a significant amount of study has been done to develop the notion of the “timed Petri-net”, standard Petri-net formalisms lack any specific mechanism to keep track of energy consumption. The traditional mechanism to model general resource usage is through the tokens that are exchanged within the net itself, and a system to model energy consumption could be based on the tokens themselves. The method followed in CONSERN allows annotations to be added to existing Petri-nets to include energy consumption and timing information attached to specific transitions in the models.

The rest of this section describes the extensions to standard Petri-net formalism made in CONSERN to enable power consumption to be included in models. We will describe the analysis we can do to produce consumption estimates from the models. The later Section 3.1 will introduce a model of a sensor application which supports selection between two modes to perform aggregation of results either on the sensor or in a separate aggregation device. We show how these models can be linked together and simulated at the network level in Deliverable 2.1 [38].

Figure 2-6 shows an overview of the tools that make up the power analysis approach we have developed. Firstly an existing Petri-net model can be augmented with information that defines the power consumption characteristics. These annotations are described in Section 3.1.1. The Petri-net compiler tool “snark” can take such nets and produce implementation code (ANSI C) that, together with platform specific libraries, can be built for a number of platforms. The tool originally targeted a simulation platform and one that enabled code to be targeted as a Linux kernel driver. Work within the CONSERN project has extended this to include support for the TelosB mote platform that we aim to target. We can also use this tool to generate an executable version of the model that allows for direct simulation. The energy and timing extensions can be preserved, so that execution of the model produces a trace of these operations and can be post-processed to produce power estimates which can be used to verify the statically computed ones.

The second tool is the PowerNet analyser that takes a Petri-net model with power annotations and produces a report of the relative firing rates and total power consumption of the net. The operation of this is more closely described in Section 3.1.3..

We can also directly simulate the same models using the “RENEW: Reference Net Workshop” tool. This approach is most relevant to the use of these models in simulating a full system, and is discussed in more detail in Deliverable 2.1 [38].

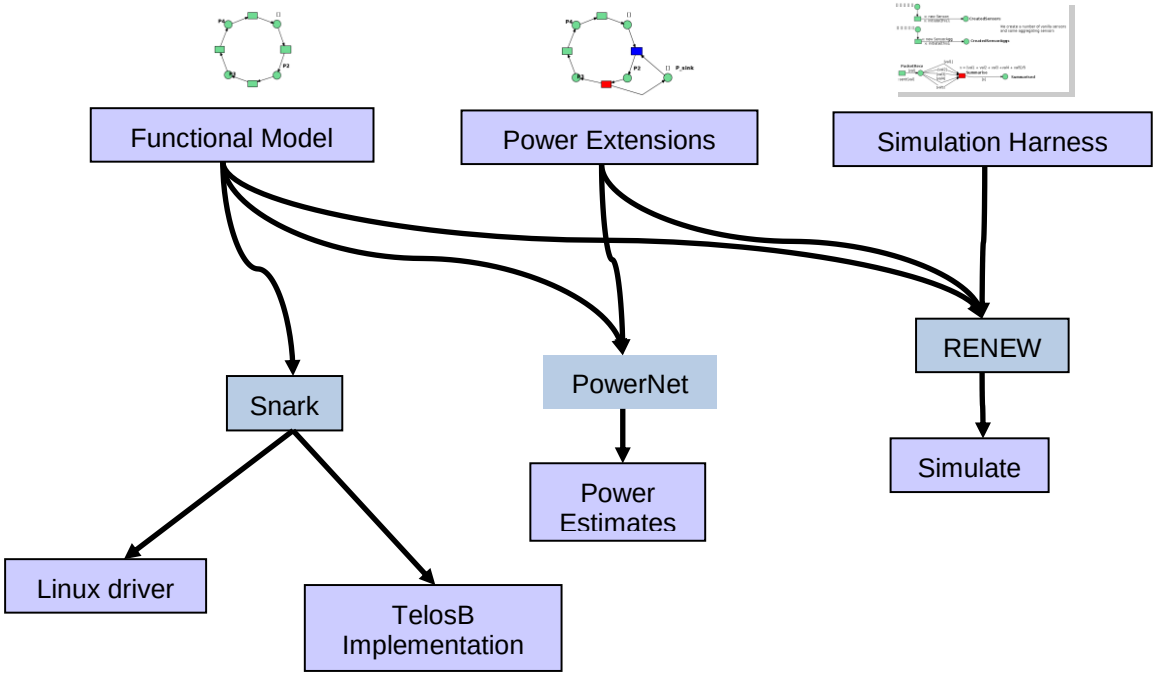

Figure 2-6: Overview of power analysis tools.

2.2 Combined Hardware and Software Simulation

Modern radio systems are more and more software based, allowing increased flexibility and ease of configurability. Indeed, Software Defined Radio (SDR) has emerged as an innovative method for increasing flexibility in radio communication systems, since components that have typically been implemented in hardware are instead implemented using software. This concept is not new, however recent advances in VLSI technology have significantly enhanced the potential of SDR systems, allowing the realization of a wide range of functions in software, something that previously was only theoretically possible.

On the other side of the coin, such software-based systems operate at the expense of increased power consumption due to higher cost of execution cycles. Especially, for future mobile telecommunication environments, in which low energy and “green” aspects are considered of paramount importance, optimized and energy efficient techniques should be developed. Such techniques are important in order to address both environmental issues and ensure reduced operational cost and improved long-term reliability.

At the terminal level, which is the main scope of this deliverable, all the above mentioned issues gain even higher importance, due to reduced battery resources, and necessitate the development of more sophisticated solutions in terms of energy efficiency. Based on this observation, Cognitive Radio (CR) systems can be significantly benefited from such solutions. Cognitive Radio leverages on the capabilities offered by SDR that facilitates the implementation of some key functions in software instead of dedicated hardware. CR devices, which may be for example LTE mobile terminals (UEs in LTE terminology) operating in an unlicensed band, e.g. in TV White Space (TVWS), have reduced battery resources and in parallel should perform a series of energy-intensive functionalities. For the simulation of SDR systems, this section covers two aspects: platform and components. Platform is interpreted as an implementation of a system covering at least all baseband data processing of the physical layer where a component is a single processor performing one or several tasks on a platform. On an SDR system this is generally an Application Specific Instruction set Processor (ASIP).

The first section 2.2.1 explains in detail the simulation of a heterogeneous multiprocessor system-on-chip (MPSOC) architecture, named Cobra Virtual Platform. This platform contains several ASIP's for performing the data processing. The ASIP's are simulated in different ways (instruction accurate, cycle accurate). There exists however other methods to simulate processors used in SDR systems. One such a simulator is the SuperESCalar Simulator (SESC). This simulator is covered in detail in section 2.2.2.

2.2.1 Cobra Virtual Platform

2.2.1.1 Design Tool Choice

Platform design tools for large reconfigurable radio system in general do not really exist at the moment. A platform is generally made from different components that all use their own specific design tools and simulation tools. Rather than talking about design tools it is more important to look at how the tools are simulating the component when looking at platform design and simulation. For a general purpose processor or ASIP's there are different levels of simulation:

- Behavioural: the internals of the component are not simulated, but its input to output behaviour is correct in time (to a certain extend) and functionality
- Instruction accurate: all instructions that are run on the actual processor are simulated. As such, the software written for the processor can be validated and debugged. When the component offers this level of simulation, we can generally speak about platform simulation rather than system simulation as here the distinction between hardware and software has been made
- Cycle accurate: All clock transitions of the component are simulated. At this level, the bitwise behaviour of the component can be tested within the platform.
- Register Transfer Level (RTL), netlist... : the following layers of accuracy are for validation and testing of the platform when going for tape-out (the actual chip design of the platform). At these levels, the architecture of the platform is already fixed and the focus is the actual translation of the platform into the placement of transistors on a chip.

When designing a platform (the architecture), the components are simulated using behavioural, instruction accurate and cycle accurate models of the components. Some tools for component design will offer the possibility to generate simulation models for different accuracy levels. The design tool company Retarget [7] for example can generate both a cycle accurate model as well as an RTL model of your ASIP design. Other vendors like Carbon [77] can generate cycle accurate models written in SystemC from RTL models written in VHDL in order to speed up the simulation of a component.

The Cobra virtual platform explained in this section uses several design tools for its components and uses also several accuracy levels in the simulation of the components. The joined simulation of all these components is possible through the use of SystemC and the concept of Transaction Level Modelling (TLM).

A good explanation of TLM can be found on Wikipedia [78]: *"Transaction-level modeling (TLM) is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture. Communication mechanisms such as busses or FIFOs are modeled as channels, and are presented to modules using SystemC interface classes. Transaction requests take place by calling interface functions of these channel models, which encapsulate low-level details of the information exchange. At the transaction level, the emphasis is more on the functionality of the data transfers -*

what data are transferred to and from what locations - and less on their actual implementation, that is, on the actual protocol used for data transfer. This approach makes it easier for the system-level designer to experiment, for example, with different bus architectures (all supporting a common abstract interface) without having to recode models that interact with any of the buses, provided these models interact with the bus though the common interface."

The use of TLM is thus not a tool choice, but a choice on how to simulate different components together that were modelled at different abstraction levels. The Cobra virtual platform below has been designed with the Synopsis Platform Architect [2] tools. This tool does actually the gluing together of the different models by the concept of TLM. This way, it can be seen more as an aid than an actual design tool. Therefore, the mechanisms that are developed in section 3 can be applied to any platform that is designed according to the principles of TLM. Moreover, the mechanisms introduced here can be seen as a use case on how energy awareness can be applied at platform design level when designing a reconfigurable radio platform.

2.2.1.2 Overview

The Software Defined Radio platform architecture template developed at imec is a heterogeneous multiprocessor system-on-chip (MPSOC) architecture. It is called Cobra, an acronym for Cognitive Baseband Radio, and It targets primarily OFDM based wireless standards as WiMAX (IEEE 802.16), WLAN (IEEE 802.11a/b/g/n), DVB-T(2) and 3GPP-LTE(A). All the main data-processing cores are in house developed ASIP's or ASIC's.

For faster development of this architecture template, a virtual platform model is maintained that integrates hardware and software components at different levels of abstraction, such as Transaction Level Models (TLM) written in C++ or SystemC, instruction or cycle accurate processor models and RTL models. This platform template is developed and simulated within Synopsis Platform Architect [2]. As a proof of concept, the predecessor of the Cobra platform template, the Baseband Engine for Adaptive Radio (BEAR) was actually refined and processed to a physical chip implementation [3].

The platform architecture is a template, i.e. through configuration options; multiple instantiations can be generated by modifying the number of instantiated cores, allowing flexibility in both performance and energy consumption. For example, a single or multiple antenna (SISO or MIMO) capable platforms can be built or the choice to add the optional WLAN 802.11n Low Density Parity Check (LDPC) block coding can be made.

The typical functionality it offers in a wireless reception chain starts after the analog to digital conversion of the baseband signal and includes:

- Digital filtering with down sampling,
- Signal detection and time synchronization,
- Demodulation of the OFDM modulated time domain samples into soft or hard coded payload data. This includes for example: FFT (Fast Fourier Transformation), CFO (Carrier Frequency Offset) estimation and compensation, I/Q imbalance compensation and estimation. Channel estimation and equalization, demapping ...,
- Decoding of the payload data for a given coding rate and decoding algorithm. For example: LDPC (Low Density Parity Check), Turbo decoding, Viterbi decoding ...,
- Inner modem operations other than decoding: de-interleaving, de-puncturing, CRC checks ...

An instantiation of the platform is shown in Figure 2-7.

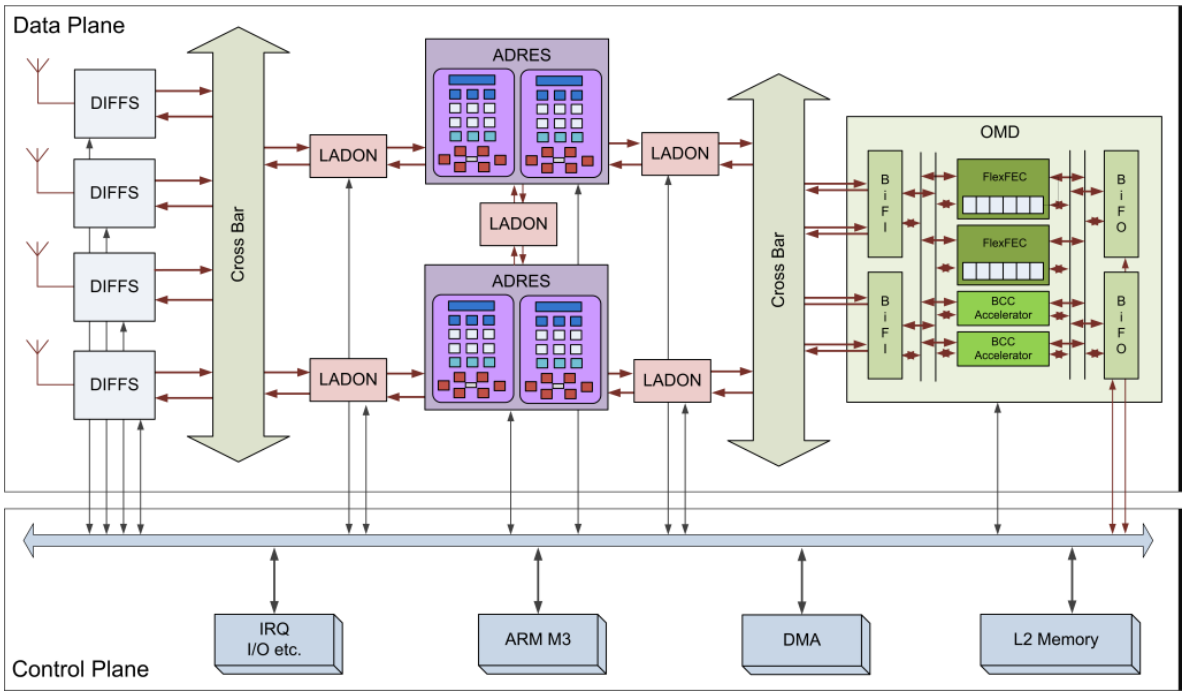

Figure 2-7: An instantiation of the Cobra reconfigurable platform.

This instantiation shows how the platform is organized in a data plane and a control plane. The transmitted or received data flows through the data plane. The flow itself is programmed and controlled by the control plane with an ARM Cortex M3 [4] processor as the controlling heart of the platform. The following sections provide details about the main platform component present in the data path for a typical receive scenario.

2.2.1.3 DIFFS

Every antenna interface (read analogue to digital converter) is connected with the DIFFS component. It is responsible for the following functionality:

1. Detection of the received signal,
2. Digital time domain filtering of data,
3. Time domain synchronization,
4. I/Q imbalance compensation,
5. Coarse CFO (carrier frequency offset) compensation,
6. Coarse SCO (sample carrier offset) compensation,
7. It also contains a "sensing engine" capable of performing spectrum sensing in a frequency band with a width up to 100Mhz.

The DIFFS architecture is depicted in Figure 2-8. For normal reception it contains a low power data path using the low power fixed filter branches, while for sensing a more flexible, more power consuming filter branch is available. It contains two programmable controllers: the AGRAC and the Sync/Sense engine. The AGRAC (AGC and Resource Activity Controller) is responsible for the AGC (Automatic Gain Control) of the Analog Front-End connected to the DIFFS and for the overall control of the DIFFS (e.g. selection of the filter branch). It is a simple 8-bit microcontroller compatible with a Microchip PIC16F84A controller [5] . The Sync/Sense Engine is an ASIP (Application Specific

Integrated Processor) specifically designed for synchronisation and sensing purposes. A version of the DIFFS was taped out to a physical IC[6]. Minor modifications to this chip instance of the DIFFS were still needed for integration in the platform, among others the integration of an AHBLite slave interface to the RX buffer (denoted as the "Host data Rx interface" in the figure) and an adaptation of the control interface (not shown in the figure) to an AHBLite slave interface.

The model of the DIFFS used in the virtual platform is a systemC model. The Sync/Sense Engine was developed using the Synopsys tool Processor Designer [2], where a high level description of the processor is input to the tool that emits both the hardware description as a systemC model for it. For the AGRAC, the existing hardware description was taken as a functional specification to develop a processor model with the IC designer tool from Target Compiler Technologies [7]. With this tool, an ASIP is modelled at a higher level and also a systemC model and a hardware description is emitted. The hardware description coming out was functionally compatible with the original hardware description, but in terms of performance and area, it was better. As such, the SystemC model is integrated in the platform as an upgrade of the DIFFS.

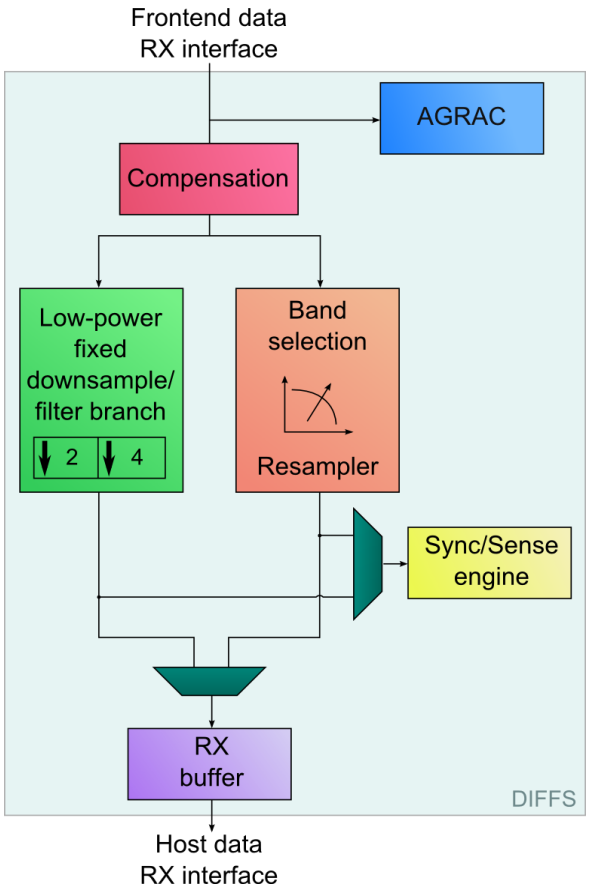

Figure 2-8: DIFFS architecture overview.

2.2.1.4 Cross bar

This is a 256bit wide AMBA[4] based bus type that connects every DIFFS to every connected LADON (see below) without causing any interference or blocking when two data streams occur in parallel.

2.2.1.5 LADON

This ASIP can be seen as a programmable (in C or assembly) DMA controller that transfers the data between the several DIFFS instances and the ADRES baseband processing cores (see below). This is

done in a synchronized and pipelined way taking into account the availability of data at the DIFFS side and the permission to write at the ADRES side. The LADON can typically transfer a complete packet without interference from the ARM M3 controlling processor and hence offloading this processor. A more detailed view on the LADONs and how they are integrated in the platform is presented in Figure 2-9. In this figure, the LADONS between the ADRES and the DIFFS are presented, but the LADONs between the ADRES and the OMD are similar. As can be seen, there is one LADON per ADRES and a single LADON is connected to a single ADRES. As such, a LADON can be seen as a DMA controller dedication to a specific ADRES. Every LADON has two AHBLite master interfaces, to ensure that transfers can be executed in a pipelined way, i.e. reading on one interface while writing previously read data words on the other interface.

As can be seen, the LADONs share a single control interface that is the connected to the control bus. Through this interface, the ARM M3 can either set parameters in a parameterised firmware image of each of the LADONS by writing into its data memory, or it can reload a full firmware image by writing to the instruction memory. Each of the LADONS is connected with the synchronisation signals from the DIFFS and the ADRES. The LADON has an instruction allowing the firmware developer to stall the execution until one or more of these synchronisation signals are raised. These signals are used to prevent overwriting data in the ADRES that has not yet been processed and reading samples from the DIFFS before they are available. For the LADONs at the OMD side, it prevents reading data from ADRES memory before this data is ready, as well as writing data to the OMD before it is ready to take this data in (e.g. because the buffers are full). There are also status signals from a LADON to an ADRES that are used to inform the ADRES that new input data was provided and that it can resume its processing, or, on the OMD side, that the output data was transferred and that the output memory locations can be reused. The status signal is also used as an acknowledged for the synchronisation signal from the ADRES. Note that there is a connection from the status signal of a LADON to the synchronisation signal of another LADON, allowing the LADONs to synchronise there operation. These synchronisation mechanisms allow flow control on the data stream ensuring correct operation without needing excessive buffer space, both if the LADONs are operating asynchronously on independent stream, as well as if the LADONs are co-operating on synchronous streams for a single communication mode.

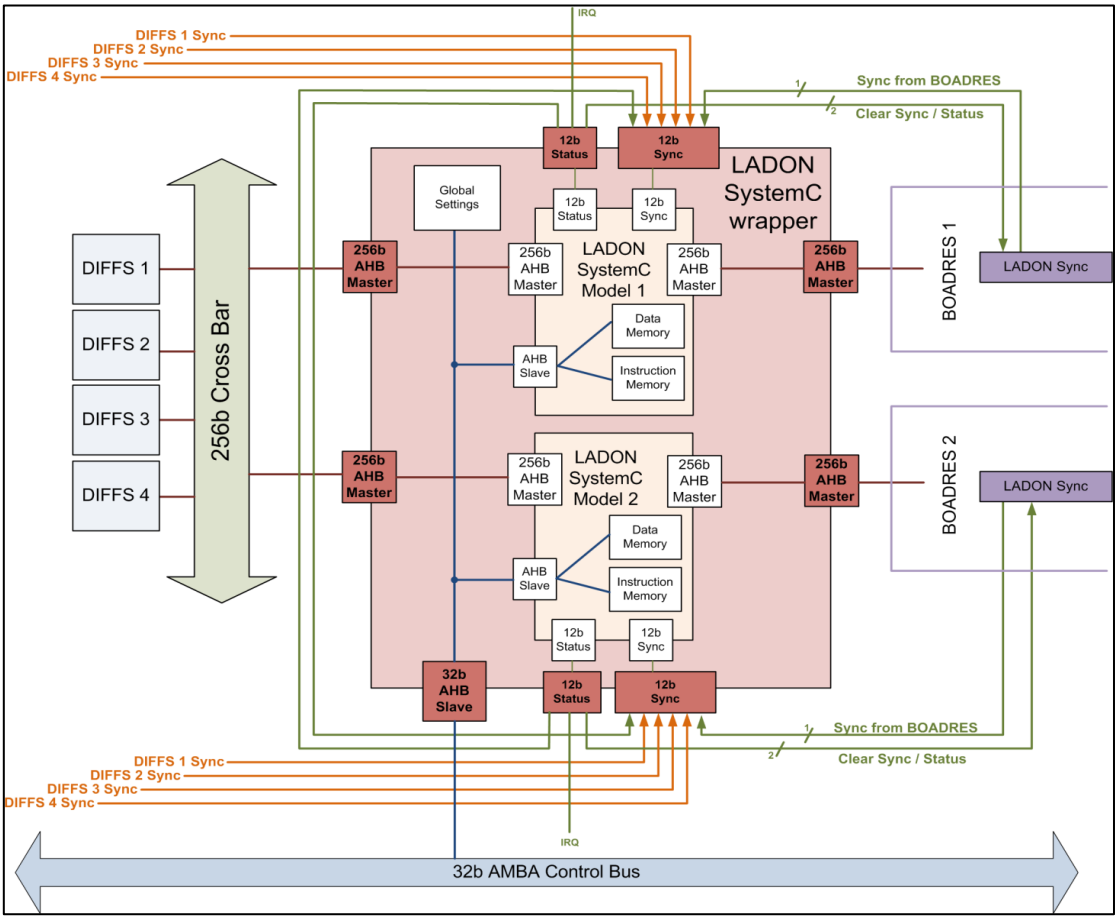

Figure 2-9: LADONs and integration in the platform (DIFFS side).

2.2.1.6 ADRES

The ADRES (ADRES stands for Architecture for Dynamically Reconfigurable Embedded Systems) performs most of the Inner MoDem (IMD) processing. It is an optimized instantiation of the ADRES processor template [8]. The BEAR [3] implementation used a two dimensional VLIW processor (also known as CGRA or Coarse-Grained Reconfigurable Array) with 4x4 (16) Functional Units (FU) featuring SIMD instructions. In the cobra, an improved instance called the BOADRES is used.

Using the in house developed DRESC compile chain different baseband functionality can be mapped to this processor: CFO or Carrier Frequency Offset compensation (in time or frequency domain), SCO or Sample Carrier Offset compensation, Fast Fourier Transform (FFT), channel estimation, channel equalization, soft and hard demapping, normalization, averaging, (de)interleaving etc. Each ADRES has a scratch pad memory connected to the AMBA control bus for communication with the ARM M3 control processor. This mechanism allows the ARM M3 firmware to configure the communication mode (number of antennas, modulation scheme, packet length ...) through the exchange of parameters. For the transfer of payload data, dedicated 256-bit wide memories are connected to the LADONs. For the reception path the input samples are received from the LADON on the side connected to the DIFFS, the soft output demodulation data is moved out by the LADON on the side connected to the OMD. Note that each partition is connected to two such memories, one connected to the LADON towards the DIFFS and one connected to the LADON towards the OMD. This ensures that separate threads can run on either of the two partitions, with the LADON being able to feed in and out data. For the transmission path the input is received from the OMD through the LADONs in

the form of coded data, the output is sent to the diffs through the LADONs in the form of samples to be transmitted.

2.2.1.7 OMD

The Outer MoDem performs the so called outer modem processing for the targeted standards: (de)interleaving, (de)puncturing, padding, CRC checks, (de)interlacing, rate (de)matching, adding/removing shortening and repetition bits, etc. The main part of the OMD is the FlexFEC.

2.2.1.8 FlexFEC

The Flexible Forward Error Correction coder (FlexFEC) ASIP core can perform both LDPC and Turbo decoding onto a unified architecture. Both decoding algorithms (based on the max* approximation [76]) are mapped onto a SIMD engine with support for data permutation. The controls for these permutations as well as the memory control signals are generated by an Address Generator Unit (AGU). The FlexFEC core is also a template that can be tuned to reach a good trade-off between power, area and throughput. It is a SIMD architecture with an amount of slots that can be parameterized.

A chip instance with 96 SIMD slots capable of performing WiFi (802.11n), WiMAX (802.16e) and 3GPP-LTE channel decoding has been designed and taped-out. More details about the core can be found in [9]. The Cobra reconfigurable platform uses a model of this instance for the optional 802.11n LDPC codes and the LTE Turbo codes. Input and output FIFO interfaces are connected to the FlexFEC's internal memory. The 256bit wide input interface is connected to the crossbars, allowing the LADONs to feed in data. The 32bit wide output interface is connected to the control, allowing the ARM M3 controller to read out decoded data.

2.2.1.9 CONSERN Contribution

As said in the beginning of this section (2.2.1.1), most components in the Cobra virtual platform are designed with different design tools. The TLM concept and more specifically SystemC and the Platform Designer tool from Synopsis [2] do not address power or energy reporting. Therefore, it is very difficult to estimate the power consumption when simulating a platform at this level of abstraction. However, the platform level is used for the design of the embedded software and the way this code is written has a considerable impact on the system's energy consumption. Therefore, it would be good that the platform and especially the software designer had already an idea of what impact his software will have on the overall power consumption of the platform. Therefore, section 3.2 will introduce ways to bring energy reporting to this level of simulation giving the software designer and platform architect a way to validate his code for power consumption and energy efficiency.

2.2.2 SuperESCalar Simulator (SESC)

2.2.2.1 Design Tool Choice

Without doubt, today's chip design requires extensive terminal-level simulations in order to ensure that the right architectural trade-offs are made. In most cases, these simulations require that a substantial amount of software is executed on the simulation model of the chip to cover the required functionality. System on Chip (SoC) verification has become more complex than ever as applications converge to offer more features for consumer products. This new level of complexity is presenting SoC development teams with many challenges including verifying their designs in a mixed-language and mixed-abstraction level environment while meeting compressed schedules.

SESC was primarily developed by Jose Renau at the University of California and the IACOMA Group at the University of Illinois at Urbana-Champaign. Currently, SESC is widely used by several research groups worldwide occupied in the field of microprocessor simulation. SESC is written in C++, since it is faster than Java and has good object-oriented programming support and may be run on many UNIX systems as Linux and Darwin/MacOS X (including both big-endian and little-endian processors). SESC is a very fast simulator since during the whole design performance and clarity have been the main focus. As an aftermath, the simulator is able to execute over 1.5MIPS on an Intel Pentium4 at 3GHz.

SESC is an event-driven simulator and the actual instructions are executed in an emulation module, which emulates the MIPS Instruction Set Architecture (ISA). This emulation module is based on MINT, a MIPS emulator that emulates the instructions in the application binary in order. The emulator returns instruction objects to SESC which are then used for the timing simulator. These instruction objects contain all the relevant information necessary for accurate timing. This includes the address of the instruction, the addresses of any loads or stores to memory, the source and destination registers, and the functional units used by the instruction. The bulk of the simulator uses this information to calculate how much time it takes for the instruction to execute through the pipeline. The justification for this is twofold. First, it is much faster to have the actual instruction executed in a simple emulator. Second, it is easier to program and debug when execution and timing are separated. The timing simulator, which is very complex, does not need to be 100% accurate if it does not affect the computation of instructions. If a bug causes the simulator to have a 0.1% error in timing accuracy, this is perfectly acceptable. In some instances, the programmers of SESC could deliberately ignore extremely rare race conditions which would be difficult to program correctly and would have minimal impact on timing.

SESC supports two different instruction types:

- The first type represents actual instructions (static instructions) in the application binary. Instructions are created during SESC initialization. A function reads the application binary and decodes each instruction into an internal representation. The internal representation contains information to make executing the instruction fast,
- The second type represents short-lived instructions (dynamic instructions) as they flow through the pipeline. Such instructions are specific instances of a static instruction. Each time a static instruction is executed, a dynamic instruction is created.

Additionally, SESC supports pipelines, as well as other (optional) advanced performance aiding mechanisms (e.g. superscalar processors, Simultaneous Multi-Threading, Chip Multi-processor and Thread Level Speculation capabilities, etc) that are typically not used for embedded processors. Regarding the details of the execution driven simulator core, the GProcessor (generic processor) object type coordinates interactions between the different SESC pipeline stages. The upper-level interface to the GProcessor object is the advanceClock() function, which is responsible for advancing each stage in the pipeline one clock cycle. This process is realised by calling a function to fetch instructions into the instruction queue and subsequently calling a function to issue instructions from the queue into a scheduling window. Instruction scheduling and execution is handled in other parts of the simulator.

Instruction fetch is the first stage of the pipeline. In this stage, instructions are brought into the pipeline from the instruction cache. In a typical configuration, the fetch unit will fetch up to 4 instructions per cycle. The fetch unit will also predict which direction a branch will go, and fetch instructions down the predicted path of the branch. Decoding (transforming instructions from the ISA format into an internal format) is performed when the simulated program is read and each instruction in the binary is decoded into an Instruction object. Thus, this class simply adds a decode

delay penalty to the time that the bundle is passed to the next stage of the pipeline. Moreover, SESC supports several different branch predictors. The choice of predictor and its size is selected at run-time.

Architectural Simulators are, in fact, software models which actually replicate hardware/software resources of an examined micro architecture system. The core purpose of such simulators is to predict system behaviour and performance metrics relevant to specific inputs. Architectural simulators can either model a standalone microprocessor or a complete system comprising microprocessor, memory, and I/O devices. More specifically, the main aim of microprocessor simulators is to help researchers test and validate their micro architectures, before these are passed to the design and fabrication level with actual hardware production, which is an expensive procedure. Especially, in the T2.2 of the CONSERN project, where the trade-off between hardware-based and software-based solutions is examined this is of paramount importance and can lead to significant performance evaluations that will in turn lead to low cost design and energy-aware terminals.

However, it should be noted that many approaches to microprocessor simulators have been proposed. Some are trace-driven, i.e., they use instruction traces of applications. Most are execution-driven, i.e., they actually execute the simulated application. Many simulators differentiate simulation into an emulator, which is responsible for executing the simulated application, and a timing simulator, which models the timing and energy of the simulated application. This approach provides significant performance gains in the simulation speed and at the same time constitutes the code of the simulator easier to read and modify. It is also common to have at least part of the simulator to be event-driven, again as a means to increase simulation speed. This implies that parts of the simulator can schedule an event, i.e., a function call with parameters, to occur at some time in the future. To this direction, the SESC simulator is used in the CONSERN scope since it offers high configurability, high simulation speed without losing accuracy and adapts to the above mentioned needs. Additionally, the availability of the source code makes it an excellent tool for evaluating research proposals and projects, such as the CONSERN project.

2.2.2.2 SESC

SuperESCalar Simulator (SESC) is a cycle accurate, microprocessor architectural simulator. It models a very wide set of architectures, including single processors, dynamic superscalar processors, chip multi-processors (CMPs), processors-in-memory (PIMs), and speculative multithreading architectures and provides comprehensive results regarding performance metrics at the architectural and especially the micro-architectural level. Additionally, SESC is highly configurable and supports a wide range of architectural features that can be used for increasing performance by exploiting either Instruction Level Parallelism (ILP) or Thread Level Parallelism (TLP).

Most importantly, SESC incorporates a detailed model for power/energy consumption, thus it is particularly effective for generating energy metrics. Special care is taken so that the results are accurate (e.g. keeping statistics at a low level of abstraction for all micro-architectural components, including the memory hierarchy, the interconnect network and the cache coherency protocols) while at the same time maintaining a simulation speed that is adequate for an architectural simulator. For all these reasons, SESC is well suited for generating detailed performance and energy reports.

For the purpose of the CONSERN Project, and more specifically in WP2 where terminal level energy optimizations is the main focus, modified configurations of microprocessor that comply with CR terminal characteristics are created. Such embedded architectures are not included in the original version of SESC. Therefore, we have modified and extended the functionalities of SESC in order to produce analytical and accurate simulations for CR terminals. . This entails modifying a series of parameters (such as cache size, block size, number of processors, memory hierarchy, hit/miss

latencies in different levels). This way, through the different architecture configurations, it is possible to simulate and further evaluate trade-offs between a plethora of hardware-based and software-based solutions.

As it will be presented later in Section 0, we will validate the energy consumption of the microcontroller of CR terminals. Such microcontrollers control and orchestrate all main functionalities that should be performed by a terminal, including the core decision making process. Therefore, the energy consumed in this hardware part is a very crucial issue that needs to be calculated accurately, in order to further identify enhancements on decision making strategies.

2.2.2.3 Energy Validation in SESC

In order to validate energy with the SESC simulator and extract power statistics, the enable-power pre-process configuration is considered. Additionally, the *wattchify* and *cactify* tools are taken into modified. Specifically, *wattchify* may be used to calculate the average power dissipation per access of each cores and *cactify* for calculating the average power dissipation per access of each cache in the system. The outcome of SESC simulation regarding the energy validation is the power dissipated by all discrete components of the processor. For example, energy consumed during fetching, issuing, and executing for all different processors as well as power dissipated in the clocking network may be extracted and further examined for possible power optimization opportunities identification.

The power model in SESC is based on SIM-WATTCH3. Specifically, SIM-WATTCH considers the dynamic power consumption $Pd = C*V^2_{dd} *a*f$ as the main source of power consumption, where C is the capacitance, V_{dd} the supply voltage, α is the chip activity factor and f the operation frequency. From these variables only the load capacitance C and the clock frequency f can be controlled directly. Note that more complex logic and bigger caches may increase C, while f can be manually modified by changing the Frequency parameter of the configuration file.

The aforementioned report is handled in order to extract essential power-delay design metrics. More specifically, three metrics can be calculated:

1. Energy-Delay Product (EDP),
2. Power-Delay Product (PDP), and,
3. Power-Energy Product (PEP).

Comparing the three metrics, it may be concluded that EDP places a higher weight on delay reduction, PEP places a higher weight on power reduction and PDP tries to strike a balance between the two. From a broader perspective, these three metrics are members of the generalized family of metrics of the form PmDn where m,n are regulating factors that determine emphasis on power or delay accordingly [10].

2.2.2.4 Cacti Tool

Cacti is a cache analysing tool estimating access time, cycle time, area, aspect ratio and energy consumption of a modelled cache structure. Considering all these, a designer can be confident that the trade-offs between time, power and area computations are based on the same assumption.

Cacti supports fully-associative caches, multi-ported caches, fully-independent banking, feature size scaling, power and area modelling. It takes in cache capacity, associativity, cache-line size, number of read/write ports, clock frequency, and the feature size. It uses an analytical model to compute the access time, consumed energy and efficiency of occupied layout and aspect ratio for different configurations. It also returns the configuration which has the best trade-off for access time and energy consumption, determined by its optimization function.

Cacti calculates the wire capacitance and resistance associated with the driving wire to different parts of a cache-like structure. This equivalent capacitance and resistance helps to model the system energy consumption. Cacti estimates the dynamic power and access time using such models; it also reports the summation of static and dynamic powers. Using Cacti enables computer architects to better understand the performance trade-offs inherent in different cache sizes and organizations [11], [12].

2.2.2.5 Wattch Simulator

Wattch is a fast, usefully-accurate, high-level architectural simulator that estimates CPU power consumption and allows the designer to evaluate energy dissipation in the early design stages. The software is a complement to the existing low-level design tools. Its power estimations are based on cacti, a suite of parameterizable power models for different hardware structures, and on per-cycle resource usage counts generated through cycle-level simulation. It is capable to quantify the power consumption of all the major units of the processor, parameterize them where possible and show how these power estimates can be integrated into a high-level simulator. Wattch is orders of magnitude faster than existing lay-out level power tools, while also offering accuracy in power estimates to within 10% of lower-level approaches.

The foundations of Wattch's power modelling infrastructure are parameterized power models of common structures present in modern superscalar microprocessors. These power models are integrated into superscalar simulators, such as SESC, to provide power estimates.

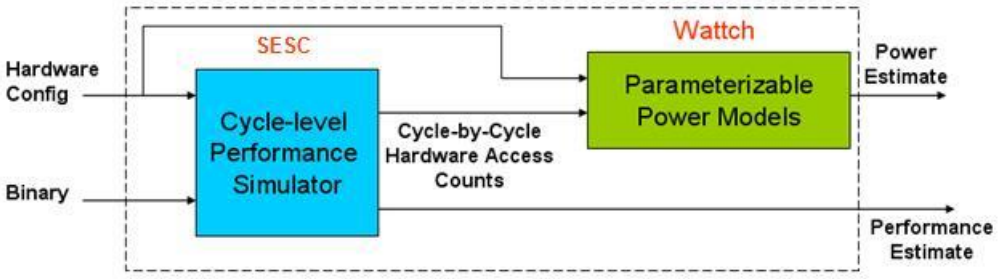

Figure 2-10: Overall Structure of the Power Simulator.

Figure 2-10 illustrates the overall structure of Wattch and the interface between the performance simulator and the power models. Wattch starts with computing the power model parameters for each unit at program start up. Then, during cycle-level simulation the per-cycle usage of units is counted. The energy estimation is generated based on the counted accesses and the computed parameters.

Architectural units in the processor are modelled differently based on their structure. The main four categories used for power modelling are: array structure, fully associative content addressable memory, combinational logics and wirings, and clocking. Table 2-1 summarizes major hardware structures and the type of model used for each of them [11], [12].

Table 2-1: Common processor hardware units and the type of model used by Wattch

Type of Structure	Associated Hardware Units
Array Structure	Data and instruction cache, cache tag arrays, all register files, register alias table, branch predictors and large portions of the instruction window and load/store queue.
Fully Associative Content-Addressable Memory	Instruction window/re-order buffer wakeup logic, and load/store order checks.

Type of Structure	Associated Hardware Units
Combination Logics and Wires	Functional unit, instruction window selection logic, dependency logic, and result buses.
Clocking	Clock buffers, clock wires, and capacitive loads

2.2.2.6 CONSERN Contribution

The accurate energy validation on the terminal level is a critical factor that is needed to be fully examined before the final implementation of the CR processor and the software implementation on top of it. All the tools described in the above sub-sections are imported in the SESC simulator and along with certain modifications are used to obtain energy characteristics on the terminal level. More specifically, on the one hand modified processor architectures are created (differentiated from the general purpose architectures originally found on SESC) and on the other hand modifications on the energy tools are implemented that enable the fast, concurrent and accurate simulation of the mechanism examined in Section 3.3.

This way, the power dissipation per access of each cores and the power dissipation per access of each cache in the system are extracted. The outcome of SESC simulation regarding the energy validation is the power dissipated by all discrete components of the processor. For example, energy consumed during fetching, issuing, and executing for all different processors as well as power dissipated in the clocking network may be extracted and further examined for possible power optimization opportunities identification.

3. Mechanisms

This section will propose mechanisms to enhance the simulation and design tools introduced in section 2 and add energy awareness to them. Section 3.1 covers the Petri-net Modelling, section 3.2 the Cobra virtual platform and section 3.3 mechanisms for energy optimization in the SESC simulator.

3.1 Petri-net Power Modelling

Here we focus on the energy extensions to the existing formalism which we introduced earlier in Section 2.1.3 and explain the further analysis that can be done.

3.1.1 Petri-net Power Extensions

This section described how to include some notion of energy consumption into the traditional Petri-net formalism. An executing (or simulating) Petri-net consists of a sequence of fired transitions. We can mark specific transitions in two different ways:

Energy transitions: these are transitions whose firing is associated with a corresponding consumption of energy. The associated value is the energy consumption in appropriate units.

Clock Transition: this transition can be thought of as marking time for the net, with an associated value that represents the time elapsed.

In this way we can consider the energy consumption (total value of the energy transitions) per clock transition. Another way of interpreting the meaning of the clock transition is that it is a transition that signifies that useful work has been done by the net, corresponding to a networking event such as a packet reception or successful message exchange.

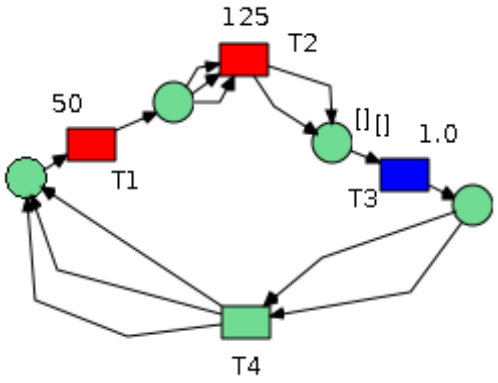

Figure 3-1: Example Petri-net with energy consumption annotations.

Figure 3-1 shows an example standard Petri-net that models a simple looped or repeated behaviour, with transitions **T1**, **T2**, **T3**, **T4** firing in order. Multiple arcs have been added to force transitions to fire at different rates, for each circuit of the net, **T1** will fire three times, **T3** will fire twice and both **T2** and **T4** will fire once. It has been annotated, with **T1** and **T2** denoted as energy consuming with relative consumption figures 50 and 125. **T3** is denoted as the clock transition with the annotation 1.0 to show that it fires once per time unit.

These models are created using the RENEW Petri-net modelling tools and are marked up by colouring transitions red (energy) or blue (clock), with the appropriate weightings added as text labels to the corresponding transitions. Our subsequent analysis is performed on the PNML output of the model, which is obtained by directly exporting from the RENEW tool.

3.1.2 Static Analysis of Power Models

This approach uses a standard transition invariant analysis of these annotated Petri-nets to derive power consumption information.

First a refresher on what the transition invariants of a Petri-net represent. Each firing of a transition removes tokens from at least one place and adds tokens to another set or places. Typically this changes the Petri-net's *marking* (the number of tokens in each place), although if a token is removed and added to the same place it can leave the marking unchanged. However, sequences of firings can return the marking back to its original state, and these are *transition invariants*. Since we can compose any combination of transition invariants and get a sequence that is itself transition invariant, we are interested in the base (shortest and simplest) set of sequences from which all the others can be derived. These invariants correspond to repeatable cycles of in the net, and, provided that the net is itself bounded we can think of them as defining the complete behaviour of the net.

We assume that the net is annotated in the same way as described previously: energy and clock transitions are marked and relative power consumptions are attached to the appropriate transitions. As an example we can analyse the net in Figure 3-2 directly. Our PowerNet tool does not perform transition invariant analysis itself; it makes calls into a separate tool - [2]*tina*, and interprets the results.

This gives the single transition invariant as output:

T1 * 3 + **T2** * 1 + **T3** * 2 + **T4** * 1 (3.1)

As mentioned before the transition invariants give a sequence of firings that together return the net to the same initial state. The above invariant (**T1** with coefficient 3, **T2** with coefficient 1, …) corresponds to a sequence where **T1** fires three times, followed by **T2**, then **T3** twice and **T4** once. The entire behaviour of this net is therefore simply this sequence of actions looped repeatedly. This tells us that for every single firing of **T4**, **T2** fires once, **T3** fires twice and **T1** fires three times. From this the relative firing rates can be used to compute the total power consumption.

A net can have more than one transition invariant, in which case they each correspond to independent parallel cycles. As an example, consider the Petri-net in Figure 3-2 which has a pair of cycles.

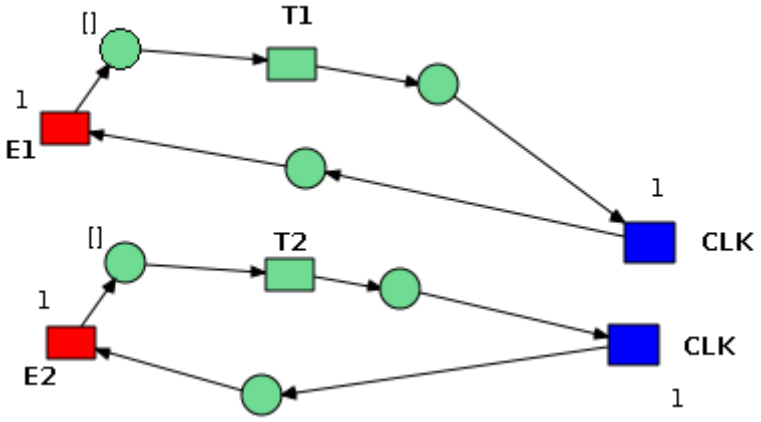

Figure 3-2: A Petri-net with two transition invariants.

[2] TINA, the Time Petri-net Analyser : http://homepages.laas.fr/bernard/tina

In this case analysis yields the two independent invariants:

E1 * 1 + **T1** * 1 + **CLK** * 1 (3.2)

E2 * 1 + **T2** * 1 + **CLK** * 1 (3.3)

These invariants completely describe the behaviour of the net as two independent cycles. The firing rates of energy transitions within each cycle can then be computed separately.

Our tool can take an arbitrary Petri-net model annotated with the energy and timing information, locate sets of looped firings and compute the overall power estimates of the algorithms.

3.1.3 Direct Simulation of Power Models

Our existing 'snark' tool can produce C code appropriate for the platforms shown in Figure 2-6, and in addition can generate standard ANSI C code for simulation of the net. The tool has been extended to handle the power extensions described above and produce trace output that logs whenever an energy consumption or timing event occurs. This trace can be post-analysed to compute total and average power consumption for an execution run of the net. This simulation estimate can be verified against the static estimates we derive using the methods outlined in Section 3.1.2.

3.2 Cobra Virtual Platform Energy Reporting Mechanism

3.2.1 Infrastructure for energy reporting

Before being able to optimise the energy of a terminal, it is necessary to get a detailed and accurate view of the (relative) power consumption of the terminal. Since a software defined radio platform such as Cobra (see section 2.2) can run a variety of communication modes with orders of magnitude difference in throughput and power, any optimisation decision maker should be provided with correct metrics.

Timing annotation mechanisms are provided by the SystemC simulation infrastructure and implemented by the building blocks within the platform. However the library models for off the shelf components such as the ARM cortex M3, or home-grown models such as the DIFFS and the OMD do not provide any infrastructure for power profiling. Therefore, a new power and energy reporting mechanism was developed.

At a higher level of abstraction than offered by the Synopsys environment (the transactional level), a client-server architecture is used for reporting and presenting the state of the platform [75]. This framework was reused an enhanced by adding energy reporting functionality to it with an API (Application Programming Interface) for the client side that can be used to report at a given time stamp the energy consumption since the previous report. The server will forward this information to a tool that acts as another client to the server, but a passive client that will only receive messages and display the reported information in a graphical way. Another possibility is to log the consecutive energy consumption reports and their timestamps to a log file that is post-processed after a simulation run with a Matlab [74] script to generate a power profile of the simulation. An example of this is shown in section 4.2.

With this infrastructure in place, it is possible to annotate each of the components in a platform with its energy consumption. It is important to note that this power annotation is provided by the hardware models within the platform, the firmware developer cannot directly influence these numbers but he can investigate the impact of firmware choices and selected communication modes on the energy consumption of the platform. Therefore, this power annotation is an important development tool to allow the system software developer to optimise his firmware not only taking into account performance, but also energy consumption. Also at the network level, optimisation engines looking at total network power can and should take into account the energy consumption of the terminal and the way this is affected by the selection of the communication modes.

3.2.2 Obtaining the numbers

Next to developing the infrastructure for the power annotations, also using the infrastructure with reliable relative numbers is a key to a correct deployment of the terminal. As the biggest energy consumers in the platform are the OMD (the FLEXFEC in particular) and the ADRES processors, these two are the first to quantify the energy consumption of. In later stages, the power consumption of secondary energy consumers (DIFFS, LADONs, ARM, interconnect) can be added to the model.

For the ADRES, the energy consumption for every functional unit per cycle, depending on the operation it is executing, is obtained through power annotated gate level simulation of the hardware description. Comparison of measured power on the BEAR[3] ADRES instance with the simulation results proved the accuracy of this approach which is within the limits of the measurement accuracy.

Within the ADRES model, per functional unit the energy per cycle for its current operation is taken and these energy numbers are aggregated over all functional units. In this way the energy consumption of all functional units in a given cycle is obtained. However, reporting the energy consumption every cycle to the server and writing it to a file would reduce the simulation speed of

the virtual platform dramatically; therefore the energy consumption is aggregated over time and only reported at context switches, i.e. when entering a function or exiting a function. This typically results in a report with a granularity of a few 100 ns (compared to the 1.25ps cycle time of the ADRES). This is still more than fine-grained enough but it has no significant impact on the simulation speed.

Moreover, since the functional reporting has the same granularity, it can be used as power profiling information informing the ADRES firmware developer where in the firmware most energy gets consumed.

For the FlexFEC's energy consumption, the energy is reported per decoded codeword as the power level will be stable during the decoding of a codeword. Depending on the amount of iterations in the decoding process, this leads to a reporting granularity of 1 to a few μs. To obtain the energy for decoding a single codeword, power measurement from the FLEXFEC chip were taken and transformed into the energy per codeword and per iteration. Then, the energy numbers were recalibrated taking into account the architectural changes to the FLEXFEC and the technology scaling. This approach gives energy and power figures for a limited set of modes. The power and energy of the other modes was estimated by linearly interpolating on the number of active SIMD slots, an approach justified by the regularity of the FlexFEC structure. When assuming that the energy evolves linearly with the number of iterations, reliable energy figures can be reported for all modes. This latter assumption is validated by the fact that for a realistic number of iterations, the time consumed in the decoding process is linear with the amount of iterations and the power during consecutive iterations can be assumed constant, as the ASIP is performing exactly the same operations.

3.3 Terminal Decision Making Energy Optimization Mechanism

Decision making in the context of Cognitive Networks constitutes a key issue [14] that is expected to trade-off between multiple and potentially conflicting objectives (e.g. response time, QoS requirements, memory restrictions, etc.). Coupled with the low energy constraints, as imposed in the context of next generation heterogeneous wireless access networks, energy-aware decision making and power management in CR devices should be examined in depth. As an aftermath, the process of the greater flexibility and more fine-grained information available to software but also the cost of execution cycles committed to power management when software based solutions are used must be taken into account during this process.

However, in order to properly optimize energy awareness of CR decision making, it is, first of all, mandatory to be able to accurately compute power consumption. Useful results that will drive the optimization process can emerge only through analytical validation of the consumed energy in each part of the decision making cycle. In a later stage, and based on this power validation, calibration techniques will be easier to be produced and most importantly will aid to target specific aspects to optimize. For this purpose, the SuperESCalar Simulator (SESC) that was presented earlier in Section 2.2.2 will be used for the power validation of the hardware parts of a CR terminal. Fuzzy Logic optimizations that are next presented are expected to lead to significant energy savings of the decision making process on the terminal level. More specifically, we highlight and examine in depth the impact of fuzzy logic based decision making process on cognitive radios in terms of performance/energy consumption.

3.3.1 Fuzzy Logic Optimizations

As it was previously referred, we consider a Cognitive Radio system in which the decision engine is realized using fuzzy reasoners. Fuzzy Logic is selected among other possible decision making mechanisms because it is very efficient for dealing with complex, multi-objective or not well defined

decision problems such as the ones that typically appear in Cognitive Radio systems. Network-related decision making (e.g. [15]) and RAT selection (e.g. [16]) based on techniques that utilize fuzzy logic have been used in numerous works (i.e. [21] and [22]) with very promising results.

Fuzzy logic, compared to other decision making methods has the advantage that it can handle unclear or complicated requirements more efficiently than Boolean algebra. Proper definition of the linguistic rules can be used to reduce signalling overhead by avoiding the Ping-Pong effect in the case of handover (i.e. [19], [20]) but also in several other cases, such as when decisions about triggering a reconfiguration process are taken. Another important advantage of fuzzy logic is that it can be applied transparently in combination with other well-known decision methods, such as multi-objective genetic algorithms [17] and game theoretic approaches [18].

Fuzzy logic is based on fuzzy set theory in which every object has a grade of membership in various sets. Inputs are mapped to membership functions, or sets (this is the fuzzification process). Knowledge of a restricted domain is captured in the form of linguistic rules. Relationships between two goals are defined using fuzzy inclusion and non-inclusion between the support and hindering sets of the corresponding goals. Finally, the required output is defuzzified (to a numerical form) from the 'THEN' part of the rules in order to produce the consequent.

In order to minimize the energy consumption in a decision making approach that utilizes fuzzy logic reasoners, a Hierarchical Fuzzy Logic (HFL) reasoner structure having as a primary objective minimal triggering of the fuzzy reasoners and consequently reduced energy consumption can be investigated and utilized. More specifically, in the context of the CONSERN Project, decision making for cognitive radio, utilizing a Hierarchical Fuzzy Logic (HFL) approach consisting of reasoners organized in tree a form will be assessed in terms of energy consumption.

More specifically, the decision making engine examined is based on the cooperative power control algorithm for Cognitive Radio Networks presented in CONSERN Deliverable D3.1. The considered algorithm addresses aspects of the local and situated decision making in the Cognitive Radio terminals. Fuzzy Logic is particularly effective for these tasks, since it can handle efficiently uncertainties, as well as multi-objective optimizations. Additionally, corresponding solutions are characterized by relatively low computational complexity. The goal of this scheme is to facilitate optimal selection of the nodes' uplink transmission power. The main idea is that the terminals set their transmission power level so as to maximize the overall utility of the network taking into account also the interference they cause to others. Therefore, one of the main characteristics is its distributed nature that aims on one hand to mitigate interference, while on the other hand to maximize the overall network utility. According to this algorithm, Fuzzy Logic is applied in order to improve situation aware decision making in the presence of uncertainties such as large update time intervals and potential problems in message exchange due to high mobility. More specifically, the inputs considered for this engine are topology environment, density and mobility level of nodes (i.e. mobile terminals), number of cognitive groups, as well as update time interval.

3.3.2 Hierarchical Fuzzy Logic

The fuzzy reasoner engine, as previously referred, can be either flat or hierarchical. Next in this section, the process of transforming a flat reasoner to an HFL reasoner will be analytically described, accompanied with the energy savings and the performance benefits emerging from this process. As an alternative, and in parallel enhancement, to the previous topology we will investigate and evaluate a Hierarchical Fuzzy Logic (HFL) reasoner structure having as primary objective to minimize the triggering of the fuzzy rules and at the same time increase thread level parallelism. The goal is to initially evaluate these different fuzzy reasoner topologies in terms of performance and energy consumption using the SESC simulator and later highlight the importance for advanced optimizations in terminal level decision making.

In a flat reasoner approach the single fuzzy reasoner *rn1* receives all the input signals as described in the previous section (Figure 3-3). However, instead of having single parameter inputs, it is beneficial to group them by taking into consideration their cohesion properties, in order to further determine the optimal position of the reasoners in the structure. HFL reasoners have been used for reducing the number of rules (typically increasing exponentially with the number of variables [23], providing better tuning capabilities and improving scalability [24]. The reduced number of rules is a characteristic desirable also from the low energy perspective, which is imperative to address in a battery powered Cognitive Radio device; however, further optimization can be achieved by exploiting the relation between the input parameters. Such an approach reduces the number of rule activations in comparison with a monolithic flat reasoner, thus better exploiting the existence of energy preservation features (e.g. clock gating).

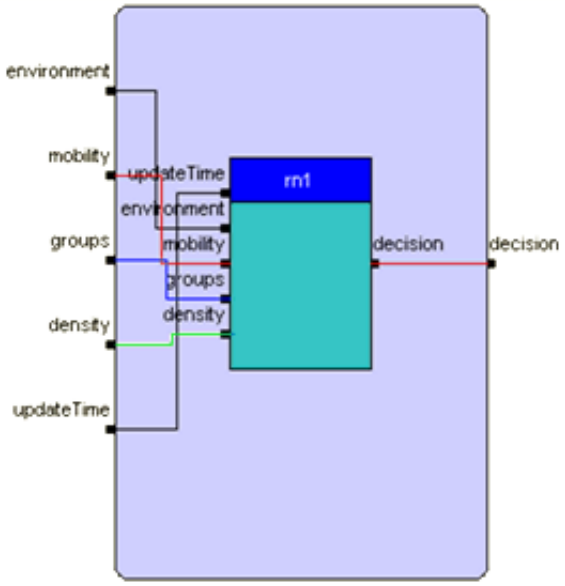

Figure 3-3: Flat Reasoner Topology.

Towards the direction of exploiting the characteristics of the input parameters of the fuzzy logic reasoners, the correlation between them will be studied, in order to define an optimal taxonomy and to minimize the number of reasoners and rules that are triggered by a change in one of the inputs. For example, topology environment and mobility are correlated, meaning that a change in one of these parameters is often accompanied by a change in the other parameter as well (e.g. high mobility cannot be realized in an indoor environment, while a terminal that moves out is likely to also increase his movement speed). By grouping such parameters in the same reasoner we reduce the number of reasoners and fuzzy rules that are triggered and expose thread level parallelism. Both of these properties reduce the response time of the system as well as the energy consumption.

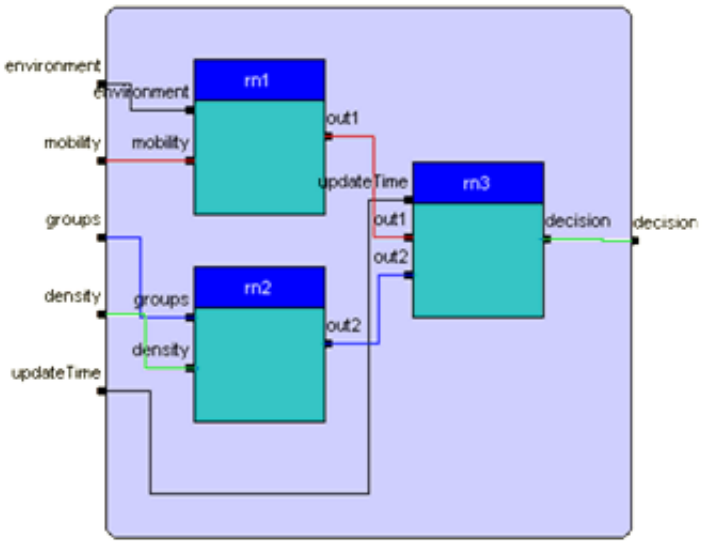

Figure 3-4. Two-level Reasoner Hierarchy.

For example, in the case of a two-level reasoner hierarchy (Figure 3-4), reasoner *rn1* receives as inputs the type of environment and mobile terminals mobility level while *rn2* receives the number of cognitive groups and terminals' density. The final reasoner *rn3* is receiving as inputs the outputs of the reasoners *rn1* and *rn2*, as well as the update time interval. The effect of the intermediate reasoner outcomes in the final outcome is derived by sensitivity analysis. Specifically, following the derivation of the optimal taxonomy, sensitivity analysis is deployed to evaluate the impact of intermediate reasoner results to the final outcome in order to refine the rule base and fine-tune the behaviour of the system. However, in order to minimize energy consumption it is also very important to group the input parameters of the system in the fuzzy reasoners according to their correlation/dependence, since an optimal grouping will minimize the number of triggered rules.

Correlation and dependence in statistics are used to describe any one among a broad class of statistical relationships between two or more random variables or observed data values. Correlations are helpful because they can indicate a predictive relationship that can be exploited in practice. For example, an electrical utility may produce less power on a mild day based on the correlation between electricity demand and weather. Correlations also often imply possible causal or mechanistic relationships; however, statistical dependence is not sufficient to demonstrate the presence of such a relationship [25].

Formally, dependence describes any case in which random variables do not explicitly satisfy a mathematical condition of probabilistic independence. In general statistical usage, correlation can refer to any deviation of two or more random variables from statistical independence, but more often it is used to describe a more specialized type of relationship between mean values. There are several correlation coefficients, often denoted ρ or r, measuring the degree of correlation. The most widely used of these is the Pearson correlation coefficient, which is sensitive only to a linear relationship between two variables (the latter may exist even if one is a nonlinear function of the other). Pearson's correlation is obtained by dividing the covariance of the two variables by the product of their standard deviations. The population correlation coefficient $\rho_{X,Y}$ between two random variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y is defined as:

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

where *E* is the expected value operator, *cov* means covariance, and, *corr* is a widely used alternative notation for Pearson's correlation.

In order to derive the correlation between the inputs of the fuzzy reasoner engine we use the *corrcoef* function in MATLAB. The latter generates the p-value by transforming the correlation to create a statistic having n-2 degrees of freedom, where n is the number of rows of X. The confidence bounds are based on an asymptotic normal distribution of 0.5*log((1+R)/(1-R)), with an approximate variance equal to 1/(n-3). These bounds are accurate for large samples when X has a multivariate normal distribution [26].

Summarizing the process, we perform the following steps in order to define the two-level HFL reasoner taxonomy:

1. Identification of the input parameters of the system,
2. Calculation of the correlation between all pairs of input parameters and grouping of those pairs that are more correlated in the same reasoner,
3. Specification of the fuzzy rules for each reasoner in the first level of the hierarchy,
4. Specification of the fuzzy rules for the second level reasoner, that receives the intermediate outcomes of the reasoners in the first level, based on sensitivity analysis in order to fine-tune the behavior of the system.

Assuming that we have five input variables with three membership functions defined for each variable, as in the case of the reference scenario, then for a complete single output implementation the flat topology requires a total number of rules equal to 3^5=243. On the other hand, for the two-level hierarchical structure the total number of rules in a complete single output implementation equals $3^2+3^2+3^3$=45. In general, if the number of membership functions is *m* and the number of input variables is *i,* in the flat hierarchy the total number of rules, as well as the number of triggered Fuzzy Rules (*FR*) are given from the following formula:

$$FR = m^i \quad (1)$$

In the two-level hierarchical structure the total number of rules that are required for fuzzy completeness is:

$$FR = \left\lfloor \frac{i}{2} \right\rfloor \cdot m^2 + m^{\left\lceil \frac{i}{2} \right\rceil} \quad (2)$$

In this case the number of triggered rules is even less than the (already reduced) number of rules expressed by equation (2). The additional improvement stems from the fact that the inputs of the fuzzy reasoner are correlated. Exploiting this correlation between inputs means that only a single path in the reasoner hierarchy will be triggered in most cases and therefore the remaining rules in the other paths of the first level of the topology will remain idle. Specifically, for the considered case if for example terminal mobility is modified, only the 3^2 rules for the reasoner *rn1* and the 3^3 rules for the reasoner *rn3* are triggered, for a total of 36 rules. However, in the original flat reasoner all 243 rules would be triggered because they have mobility as a parameter.

Another option, that progresses the hierarchical approach to its extreme, is to use a multi-level hierarchy consisting exclusively from two-input reasoners (Figure 3-5). In this case, a four-level HFL structure is composed and the total number of rules is $3^2+3^2+3^2+3^2$=36, because every one of the reasoners requires 3^2 rules (since they have three membership functions and receive two inputs, one primary and the output of the previous reasoner). In general, for the multi-level hierarchical structure comprised of two-input reasoners the total number of rules that is required is:

$$FR = (i-1) \cdot m^2 \quad (3)$$

Therefore, by adopting this approach we can further reduce the number of rules, but the number of triggered rules is not necessarily reduced, since often frequently changing parameters are also the ones with the greatest weight. On the other hand, this structure limits flexibility compared to the previous case and cannot expose thread level parallelism as effectively as the two-level topology.

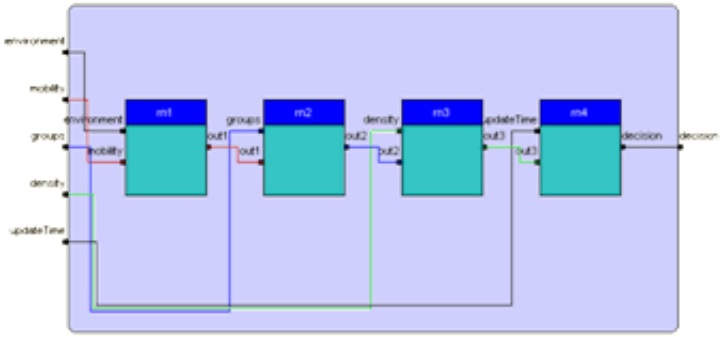

Figure 3-5. Multi-level Reasoner Hierarchy.

The Design Space Exploration (DSE) outcomes, as well as the energy gains from the optimization of the fuzzy logic reasoner structure will be derived using the SESC simulator presented in previous section. Experimental results are expected to show considerable energy savings for the decision making at the terminal level that can be as much as an order of magnitude.

4. Results

Section 2 described several simulation tools and frameworks that aid in the design of a terminal. Section 3 then showed both mechanisms to add energy awareness to this tools and methods to optimise the energy consumption of a terminal. This section will now show the results of these mechanisms and methods by showing the simulation results of the different tools and frameworks.

4.1 Petri-net Models

Here we will report on the models developed, the power consumption estimates and the verification of these estimates by direct execution.

4.1.1 Sensor Device Power Modelling

In this section a Petri-net model of a sensor device is developed and the power analysis technique is developed with this example. The application we are modelling here is based on use-case UC-13 outlined in Deliverable 1.1 of the CONSERN project. The network consists of a number of sensor devices that can record a number of different environmental variables such as light intensity, temperature and humidity. These values are then communicated back to an aggregator device, which in turn forwards them to a central collector device for logging.

From a power consumption perspective there are a number of implementation approaches that may greatly affect the resource usage of this application. Firstly, the sensor itself could perform some of the aggregation of the data before forwarding it to its parent aggregator. Secondly, the aggregator functionality could live on selected sensors, and these could (as described in UC-13) move between different nodes depending on location and environmental conditions.

The net in Figure 4-1 shows a simplified initial model of the functionality of the sensor node. Once initiated (the **Init** transition is fired), the node goes through a brief connection phase that identifies the device to its parent aggregator, culminating in the **Connected** transition firing. Power and timing estimates have not been included in this phase, since it fires infrequently, essentially only once during device setup, but may be called whenever the network forces a reconfiguration. As such contributes little to the ongoing power consumption of the device.

This model senses only one type of data, here the collection of the light intensity data is modelled by generating a random value[3]. The model selects between alternative functionality either streaming all collected data to the parent aggregator, or performing some local summarisation (here averaging over a window size of three) before sending the smoothed values back to the parent.

In the model the functionality can be switched between these two behaviours by placing a token in either the **YesAgg** or **NoAgg** places. A more sophisticated future model could permit switching between these modes as the sensor runs, but for now the selection is made from the initial marking of the net.

[3] Generated uniformly on the interval [0,1], but more sophisticated stochastic models could be applied.

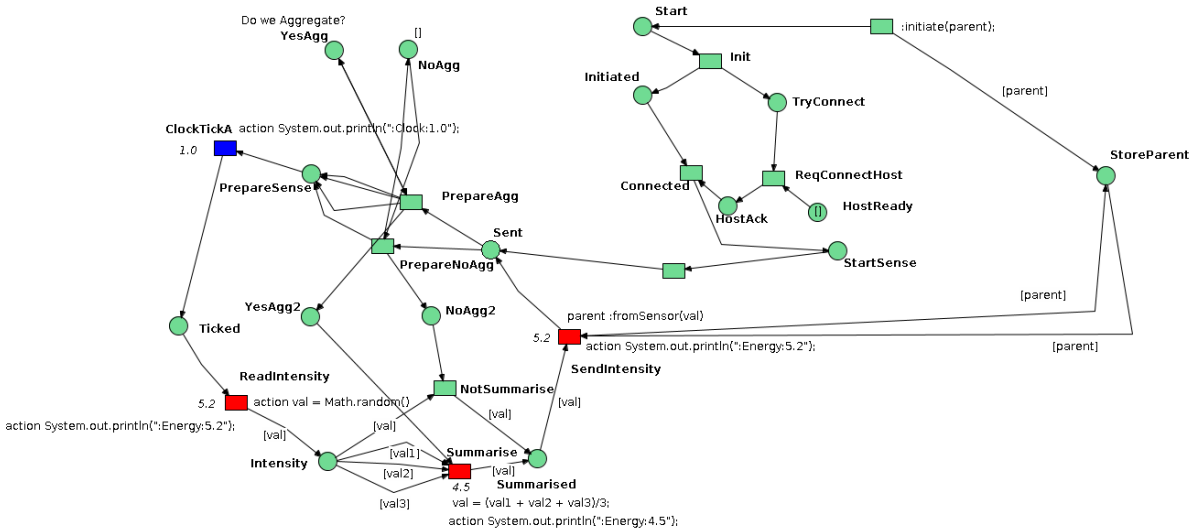

Figure 4-1: Sensor net model.

Table 4-1: Power draw figures for TelosB platform

Action	Energy consumption	Estimate/mA
ReadIntensity	R	5.2
Summarise	C	4.5
SendIntensity	S	5.2

Table 4-1 shows the consumption estimates of the various actions within the model. The placeholder variables (R, C, S) are used below to derive general expressions about the power consumption, the approximate relative power consumption figures are taken from [67] which are measured from the same sensor platform as the one we plan to use in this work. We plan to integrate more detailed power consumption figures that are developed elsewhere within this project.

4.1.1.1 Static Analysis of Model

One of the original drivers for using the Petri-net modelling approach was that models can be analysed and traditional correctness properties derived in addition to the power consumption properties we have described above. Place invariant analysis of the net shows that the above model is bounded in all places, so in more practical terms will run without overflow errors.

The power analysis phase uses the initial marking of the net to determine power consumption and can thus compute estimates for both modes. When the model is run through the tool it yields the following pair of transition invariants:

ReadIntensity * 3, SendIntensity * 1, ClockTick * 3, Summarise *1 (4.1)

ReadIntensity * 1, SendIntensity * 1, ClockTick * 1 (4.2)

We can obtain some useful results without having to attach explicit energy values to the functions in Table 4-1. Per clock tick the summarising version on the right (the upper of the two invariants above) uses R + S/3 + C/3. The second invariant gives that straight version uses R+S. So a power minimisation strategy would switch to using local summarising when . Since the power cost of transmission would depend on the transmission strength, which would in turn depend on

environmental conditions such as device separation and interference from other devices, this switching would need to happen dynamically.

4.1.1.2 Verification by Direct execution of model

In addition each sensor device model can be augmented to output a log of its power consumption activities. These can be post analysed to verify the static model estimates, and to provide dynamic simulation estimates of consumption across the whole network. These augmented models can be derived directly from the annotated power models. Table 4-2 shows the results from a comparison of the simulated and theoretically derived estimates obtained when using the estimate values inTable 4-1, for both the aggregate and non-aggregate modes of the sensor. This shows close agreement between the two methods of estimating power consumption.

Table 4-2: Comparison of model estimates with simulation

Loop	Estimate from Model	Simulation
Aggregate	10.40	10.38
Non-aggregate	8.43	8.40

4.2 Cobra Virtual Platform Simulation Results

In this section, we discuss the power annotation graphs resulting from running a simulation on the Cobra platform (described in section 2.2) with the power annotation mechanisms described in section 3.2. For a better understanding of the results, we also show the corresponding functional plot of the platform activity during this simulation run.

As the application running on the platform we selected the reception of a wireless LAN (IEEE 802.11n [28]) packet using 4x4 MIMO in a 40 MHz band, with LDPC encoding, since this is quite a demanding mode causing a lot of activity on the platform. Figure 4-2 shows displays the platform activity in the graphical logger throughout the simulation. The horizontal axis represents the simulation time, and on the vertical axis one finds the components of the platform as described in section 2.2.

Without going into the details of all components, one notices quite some variations in platform activity during the packet reception simulation. The only blocks that do not change their function very frequently, are the 4 DIFFS', that are trying to synchronise on an incoming packet when started (point 1 in Figure 4-2), and after finding synchronisation (point 2 in Figure 4-2) continue the reception until switched off (point 3 in the figure).

One can also notice that most OMD sub blocks (denoted with omdv3 in the figure) and the FlexFEC start their activities a long time (point 6 in the figure) after the packet reception has started (point 2) and still continue with it up to point 7 in the figure, a few µs after the reception has ended (marked by the DIFFS going into "off" state, point 3 in the figure). This is especially true for the FlexFEC since it is the last block in the OMD chain. An exception to this are the Viterbi, the deinterleaver and the AdresDeformatter, because these are involved in the processing of the signal field, whereas the other blocks and the FlexFEC can only start after the signal field has been processed and parsed by the ARM M3 to reprogram the outer modem. The relevant points in time with respect to the OMD processing are denoted in the figure from 4 to 7 and have following meaning:

4. Start of signal field processing (first signal field symbol),
5. Continue signal field processing (second signal field symbol),
6. Start of data processing,
7. End of data processing.

One sees also the periodical activity on the ADRES (denoted as BBE_T0 and BBE_T1 in the figure, for the two threads). The green parts in the figure denote activity on the ADRES Threads, whereas the white spaces in between denote that they are idle. Relevant time instances for the baseband processing are denoted with numbers 8 to 17 and have following meaning (this would become clear to a developer after zooming in and reading out the function names):

8. Preamble processing (first part of first Long Training Field (HT-LTF)),
9. Preamble processing (last part of first Long Training Field),
10. Signal field processing (first signal field symbol, HT-SIG1),
11. Signal field processing (second signal field symbol, HT-SIG2),
12. Preamble processing (second HT-LTF2),
13. Preamble processing (third HT-LTF3),
14. Preamble processing (fourth HT-LTF4),
15. First data symbol processing,
16. Consecutive data symbol processing,

17. Last data symbol processing.

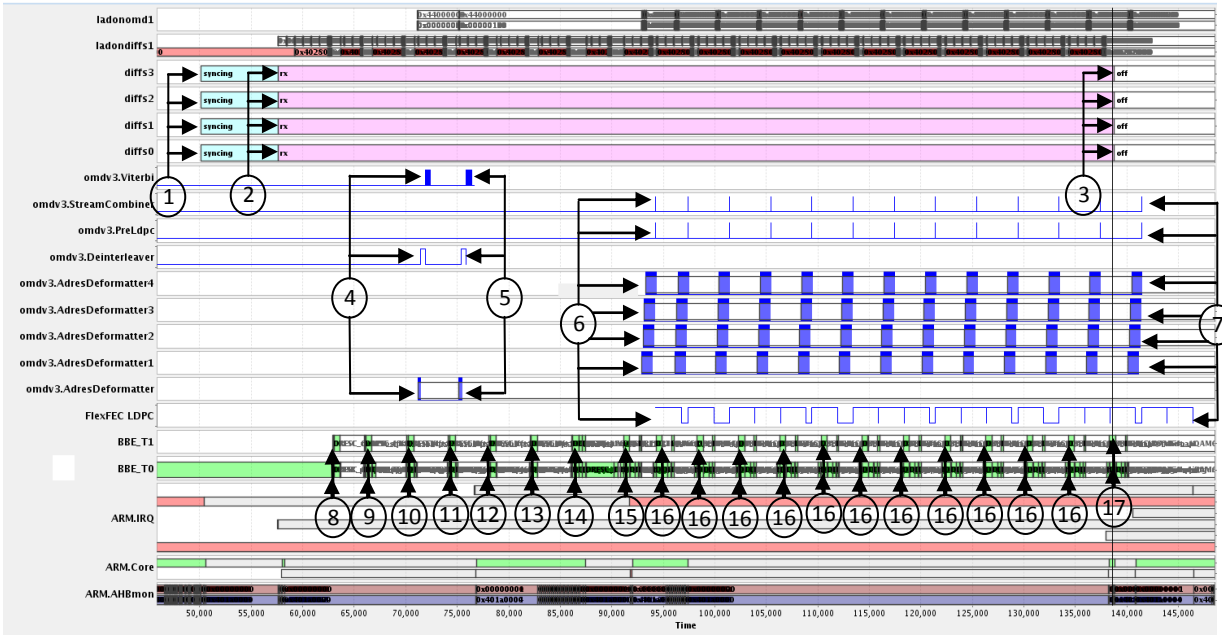

Figure 4-2: Platform activity during a WLAN 40 MHz 4x4 packet reception.

If we now zoom in on the ADRES threads and the FlexFEC and we also zoom in on the time axis, somewhere in between points 6 and 14, where both FlexFEC and ADRES are active, we look at the finer granularity Figure 4-3. In this figure, we see the flow control mechanism at work, causing the FlexFEC when it doesn't have sufficient data to process. Point 1 in the figure denotes points where the FlexFEC has finished decoding a block and has already enough data to start decoding the next block, so it restarts almost immediately. Points 2 are time points where the FlexFEC has finished decoding, but has not yet received enough data to continue, so it will wait until enough data is received to decode the next code block. If the signal is high, the FlexFEC is in operation, i.e. decoding a codeword. For the baseband, the situation is more complex: threads can be idle because of three reasons: they can be waiting for input data; they can be waiting until the output buffer is freed, or one thread can be waiting for the other. Finding out which of these three situations is occurring requires an in-depth knowledge of the firmware, e.g. knowing in which function the synchronisation with the input and output buffers is performed.

Figure 4-3: Detailed view on ADRES and FlexFEC activity.

While this is for firmware developer useful information in itself, more useful information is provided when the corresponding power profiles are taken into account.

For displaying the power, one can study the power at a per component level. For the ADRES, the power profile throughout the simulation is shown in Figure 4-4 per thread and Figure 4-5 in total, where one notices that depending on the exact functionality the ADRES is executing, one or another thread is consuming more energy, whereas also the total power consumption varies a lot during the simulation. By combining the power information with the information from the previous functional plots, the designer can find out where most of the power and energy is consumed. The numbers on the figure have the same meaning as in Figure 4-2. From the figure combining the two threads (Figure 4-5), it can be seen that the processing of the fourth LTF (HT-LTF4) is the most power hungry. This is caused by the complex MIMO processing involved. We can also see that the decoding of the data symbols requires more power than the signal field symbols. This is caused by the fact that the signal field has a simple single antenna scheme modulation index (SISO BPSK) and can be received on a single antenna, whereas the data symbols have a more complex modulation scheme (QAM64) and are received in MIMO mode on 4 antennas.

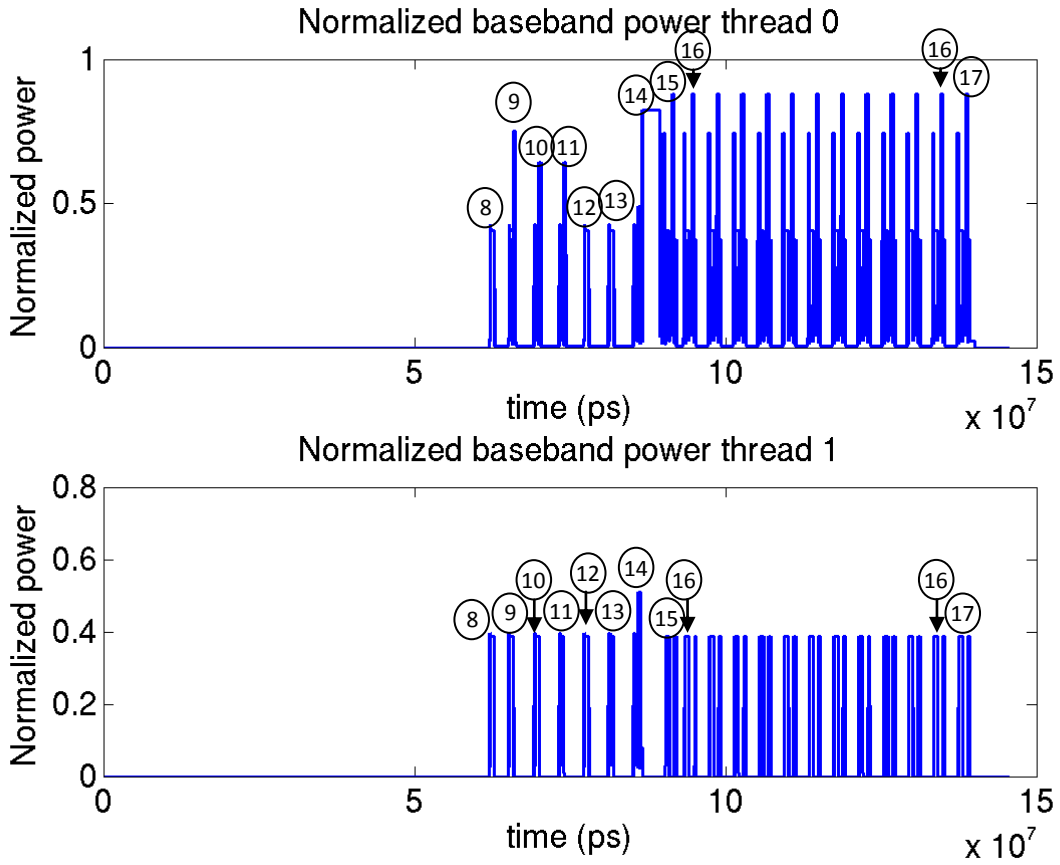

Figure 4-4: Power profile per ADRES thread.

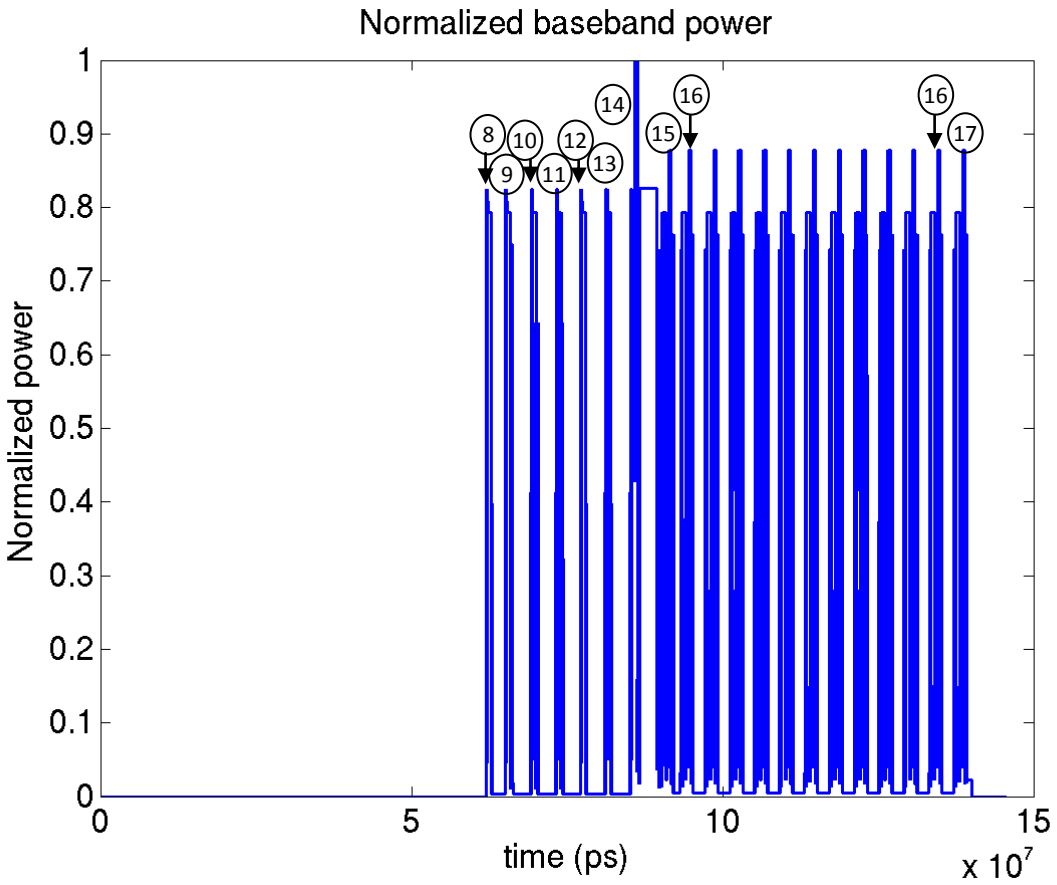

Figure 4-5: Total ADRES power profile.

For the FlexFEC, the variation for this simulation is a change between on or off (Figure 4-6), since the decoding is always the same mode, and the firmware is also the same. In a simulation with more different decoding modes, one would also notice the variation in power consumption when switching between modes.

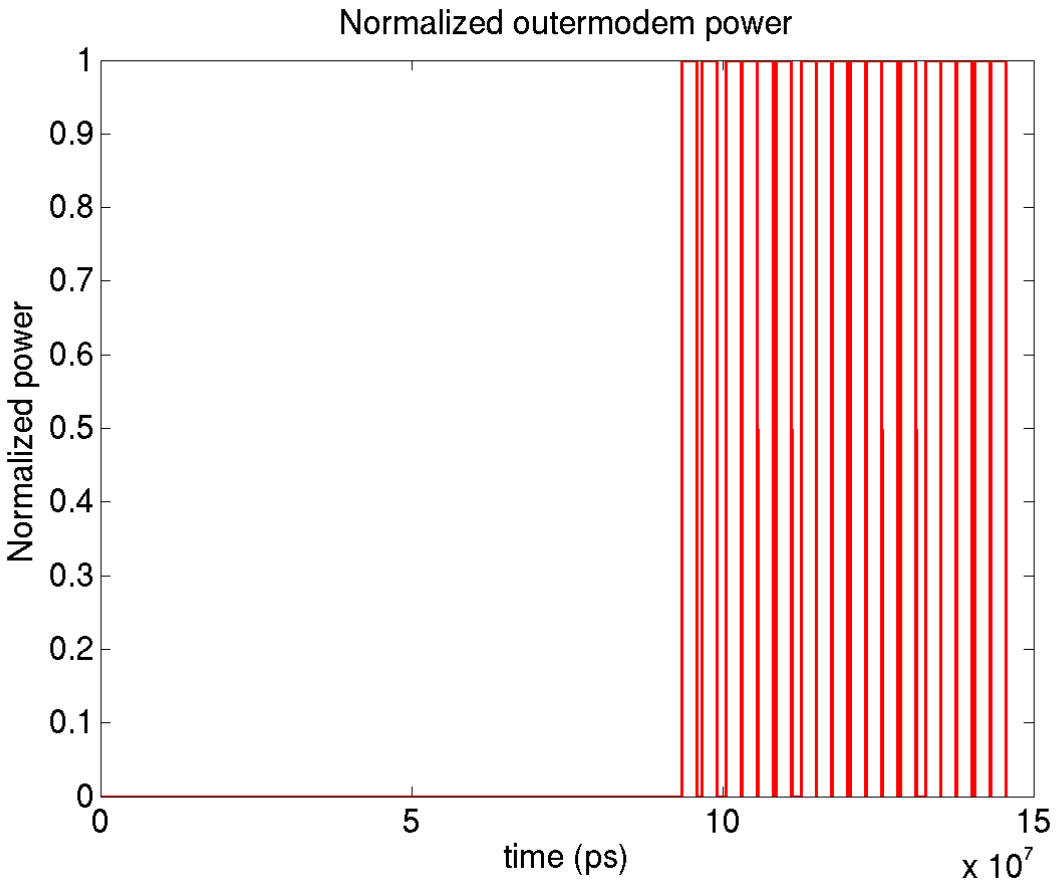

Figure 4-6: FlexFEC power profile.

4.3 Hierarchical Fuzzy Logic Simulation Results

In order to validate the energy savings in the decision making process we have used a framework comprised by a combination of the SESC architectural simulator (that was described previously in Section 2.2) and tools for designing and implementing software that realizes fuzzy logic algorithms.

In order to better capture the energy tradeoffs between software-based and hardware-based decision making and power management initially we assess the performance and energy consumption of common SDR-related functions executed in different embedded processor architectures (e.g. different frequencies, with or without multicore capabilities, different cache sizes etc). The considered functions are representative of many telecommunications processes and therefore their efficiency is of paramount importance for energy aware Cognitive Radios build around off-the-shelf embedded processors. Later, we provide analytical simulation results that consider decision making in the context of Cognitive Radio, utilizing a hierarchical fuzzy logic approach as described in Section 0.

4.3.1 Simulation Framework

The overall framework that we have used for generating performance and energy metrics is depicted in Figure 4-7.

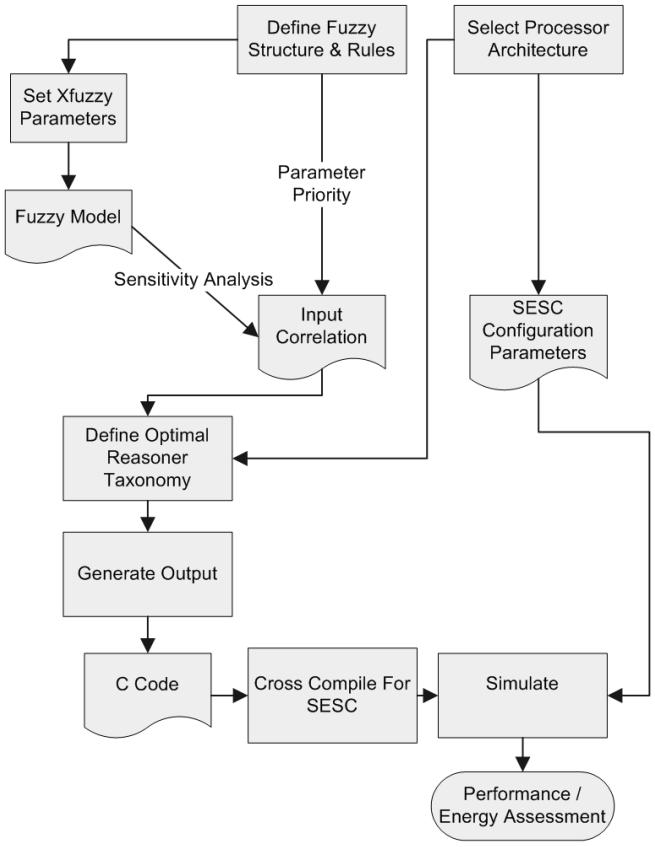

Figure 4-7. The simulation framework for fuzzy reasoning.

Initially, the fuzzy structure and rules are selected according to the specific details of the application where the decision making engine will be used. Following this step, the parameters of the Xfuzzy toolset [27], [29] are set and the fuzzy model is generated. Xfuzzy is a development environment for fuzzy-based inference systems. It is composed of several tools that cover the different stages of the fuzzy system design process, from their initial description to the final implementation. Parameter

priorities and sensitivity analysis outcomes are considered for deriving the input correlation and consequently defining the optimal reasoner taxonomy. The latter is also affected by the targeted processor architecture since architectures with support for multiple cores/threads favor hierarchical reasoner structures, because additional TLP is becoming available. After the final fuzzy model is generated, the Xfuzzy toolset is configured to generate C output and the resulting code is cross-compiled for the SESC simulator, using the provided macros that realize the thread API. Finally, the target processor architecture is also utilized to set the architectural parameters for SESC (presented in the following subsection) and the generated binary derived from cross-compilation is simulated for the considered processor architecture.

4.3.2 Simulated Processor Architectures

For the reasons mentioned previously SESC is well suited for generating detailed performance and energy reports and performing Design Space Exploration (DES) for different software and hardware implementations. For the purpose of the CONSERN Project, and more specifically in WP2 where terminal level energy optimizations is the main focus, modified configurations of microprocessor that comply with CR terminal characteristics are created. Such embedded architectures are not included in the original version of SESC. Therefore, we have modified and extended the functionalities of SESC in order to produce analytical and accurate simulations for CR terminals. It is worth mentioning that the alternative architecture configurations considered in this work comprise features targeted for embedded systems, such as mobile terminals with cognitive radio characteristics.

The six considered configurations fall in two groups, with the members of each group differing in the operational frequencies and in the size of caches. The processors in the first group are single core while in the second group are dual-core. For all configurations we evaluate three implementations, operating at 170, 250, and 450 MHz. All models are configured for in-order execution (typical for DSP processors). The caches are configured for a writeback policy with LRU replacement algorithm. Typical clock gating with linearly scaled power for each unit according to port usage and 10% of the base power when the unit is not accessed is assumed.

Table 4-3: Simulated Processor Configurations

	Conf1	Conf2	Conf3	Conf4	Conf5	Conf6
Decode Width	1	1	1	1	1	1
Issue Width	1	1	1	1	1	1
Memory Ports	2	2	2	2	2	2
L1 Data Cache	16	32	16	32	16	32
L1 I-Cache	32	32	32	32	32	32
L2 Unified Cache	0	0	0	0	0	0
L1 D-Cache Lat.	2	2	2	2	2	2
L1 I-Cache Lat.	32	32	32	32	32	32
Frequency (MHz)	170	170	250	250	450	450
Memory Latency	8	8	8	8	8	8
ALUs	1	1	1	1	1	1
L/S Queue Size	2	2	2	2	2	2
Register Unit Size	32	32	32	32	32	32

4.3.3 Design Space Exploration for SDR Systems

Various approaches have been proposed in the literature for DSE and optimization of multi-configuration processors (e.g. [30], [31]), however most of them address generic processors

architectures and do not deal with the specific properties of SDR and CR devices. SDR systems are characterized by the fact that they implement in software a range of functions that are typically realized using dedicated hardware. Such functions may include filters, mixers, amplifiers, modulators /demodulators, etc. In this work we consider two kernels of the SPLASH-2 benchmark suite [32]. The selected kernels implement functions that are commonly used in SDR solutions, and together with efficient decision making algorithms, constitute key enablers for Cognitive Radio devices.

Specifically, the FFT kernel is a complex 1-D version of the six step FFT algorithm described in [33], which is optimized to minimize interprocessor communication in multiprocessor systems. To avoid memory hotspotting, submatrices are communicated in an alternated fashion, with processor i first transposing a submatrix from processor i+1, then one from processor i+2, etc. Additional details are provided in [34]. On the other hand the LU kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. This transformation is often used for scientific as well as SDR applications (e.g. [35]). The dense n × n matrix A is divided into an N × N array of B × B blocks (n = NB) to exploit temporal locality on submatrix elements. More details are given in [34].

Normalized Power Delay Product (PDP) for the FFT and LU SPLASH-2 kernels for the selected configurations are given in Figure 4-8 to Figure 4-11. The results presented in these figures depict that if both the power and the delay are considered with equal weights, as expressed by the power-delay product (PDP), then the best choice among the considered options is Conf. 5 in both its single and dual core implementations for the FFT and LU, although for the latter the dual core implementation is marginally inferior in terms of normalized PDP. More specifically, as depicted in all figures, Conf. 5 and Conf. 6 have similar levels of normalized PDP. However, Conf. 5 is finally the optimum choice since it uses half data cache size compared to Conf. 6. Larger cache sizes (Conf2, Conf4 and Conf6) reduce slightly the execution time but at the expense of higher power consumption, thus the effect on the power delay product is marginally unfavourable.

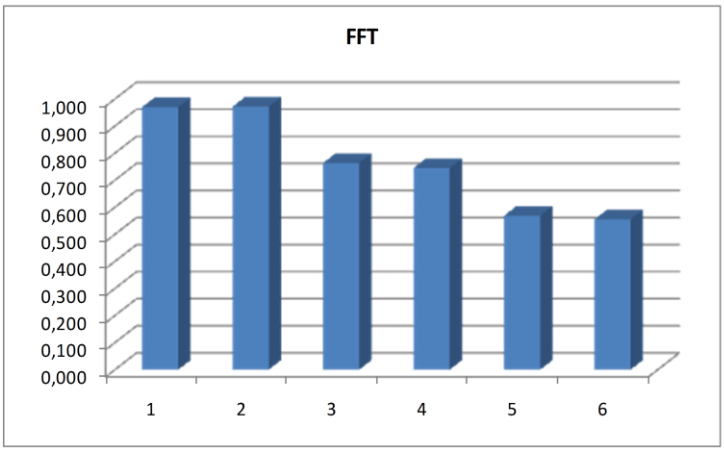

Figure 4-8. Normalized PDP of FFT for single core configurations.

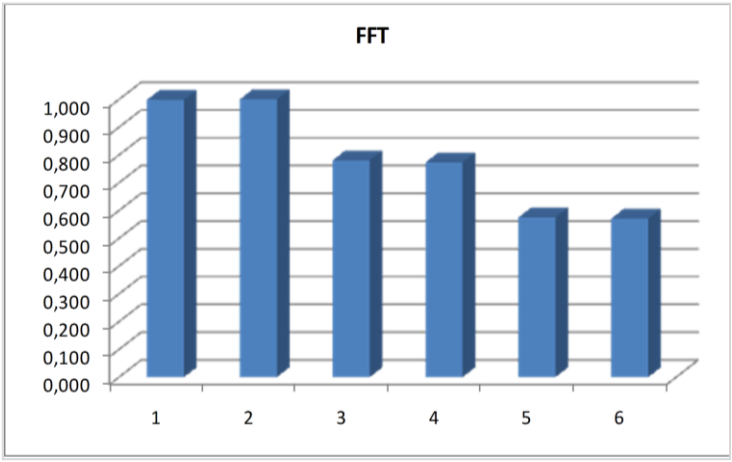

Figure 4-9. Normalized PDP of FFT for dual core configurations.

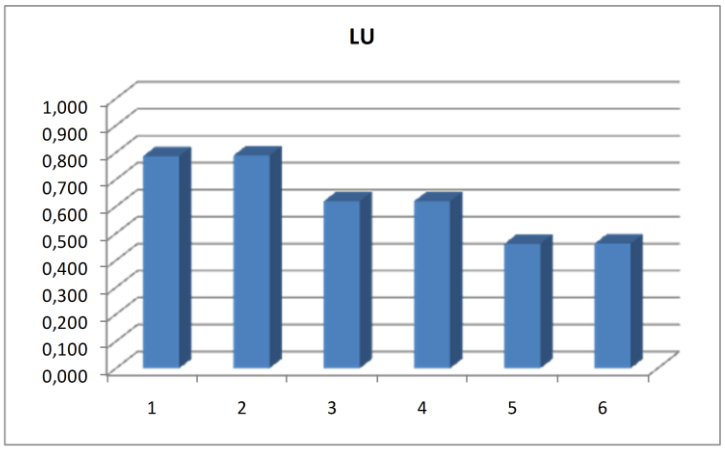

Figure 4-10. Normalized PDP of LU for single core configurations.

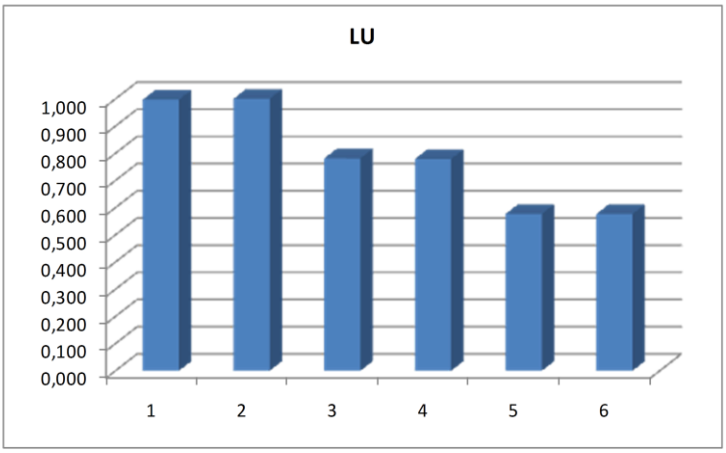

Figure 4-11. Normalized PDP of LU for dual core configurations.

Another important observation is that both of the considered benchmarks have better behaviour in terms of PDP for greater frequencies, at least for the range of frequencies that is typical for embedded processors. This outcome is in compliance with the results of [36] for the same

benchmarks. It is also important to note that the FFT kernel is known to scale well with regard to the number of threads, thus providing freedom for the scalability of the system according to the desired value of the delay and/or power, but this is not the case for the LU kernel (e.g. [37]), for which additional threads beyond two provide diminishing returns in terms of delay reduction and thus the power-delay product is significantly worse.

Therefore, the mix of SDR applications that will be executed in the system should be studied extensively before selecting a specific configuration, since the Thread Level Parallelism can differ significantly even between applications that are optimized for multithreaded systems.

4.3.4 Energy Validation of the Fuzzy Reasoning Engine

The normalized (according to the largest value) PDP results reported by the power evaluation framework for the two-level hierarchical and the multi-level hierarchical structures compared to the flat structure for the single core processor architectures are given in Figure 4-12 The same results for the dual-core architectures are given in Figure 4-13. The first bar (blue coloured) represents the flat reasoner structure, while the second and third bars (red and green coloured) represent the two-level and multi-level HFL structures respectively.

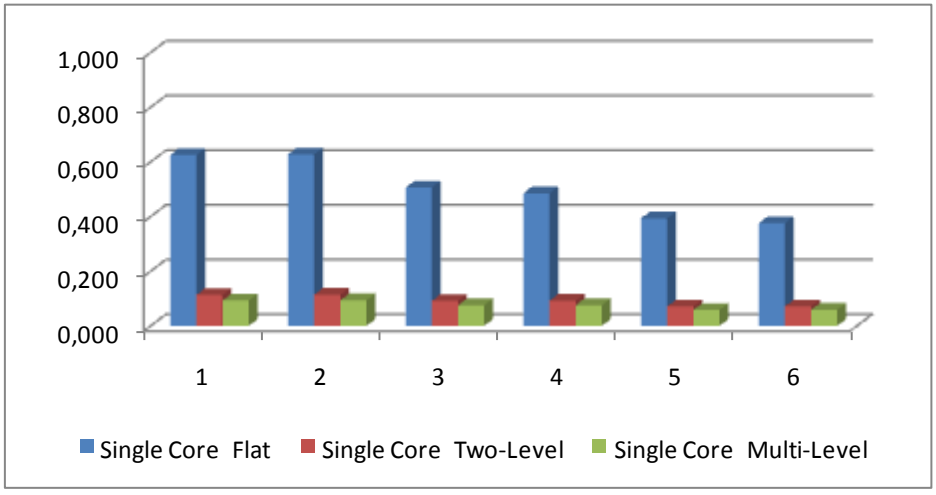

Figure 4-12. Normalized PDP for single-core configurations.

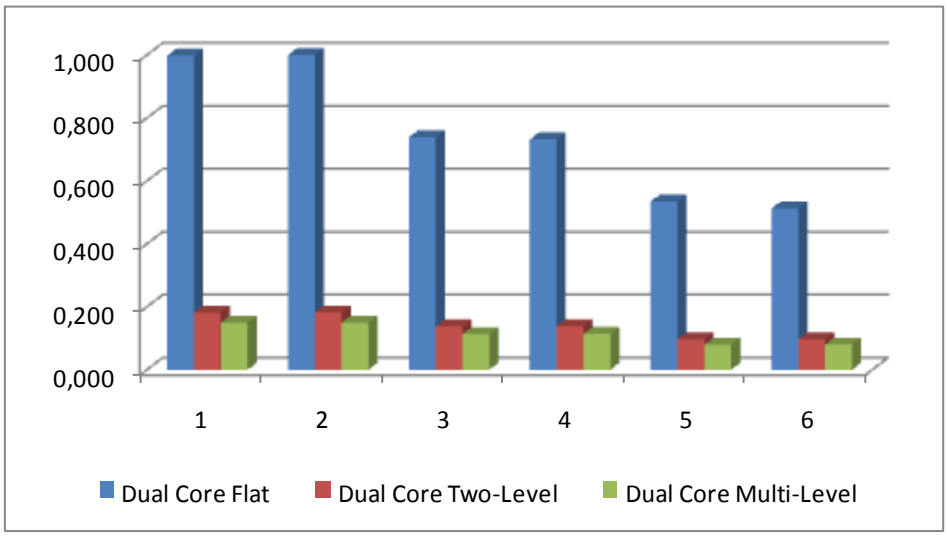

Figure 4-13. Normalized PDP for dual-core configurations.

The results reveal that considerable energy savings (up to an order of magnitude) can be achieved over the flat reasoner approach for each of the hierarchical fuzzy reasoner structures. It is also important to note that the PDP improvements in every topology are related to the number of processors and different topologies exhibit different behaviours according to the number of cores/threads in the architecture.

Specifically, while for the single-core, singe thread architectures the multi-level reasoner is preferable because it has the least number of fuzzy rules, for the multi-core architectures the two-level structure is more efficient. This is justified because the two-level structure has a comparable number of rules and is also able to expose and provide additional thread level parallelism in relation to the multi-level hierarchical structure, where every reasoner except from the one in the first level of the hierarchy receives one input, plus the output of the previous reasoner. However, in the two-level structure all the reasoners that reside in the first level of the hierarchy (the leafs of the reasoner hierarchy tree, e.g. *rn1* and *rn2*) are implemented as separate threads executed in parallel, provided that the system has enough statistically independent inputs. Thread level parallelism is also enhanced by our approach to group the input values and define the reasoner taxonomy according to their cohesion, because it will further decrease the correlation of values between reasoners and increase parallelism.

Therefore, it is evident that the decision making at the terminal level as previously described can be significantly optimized through the deployment of HFL reasoning. As an aftermath, the decision making process is further improved, leading in turn to lower computational energy needs, which is favourable (and imperative) for battery powered Cognitive Radio devices with limited energy budget.

5. Conclusion

This deliverable showed three mechanisms to introduce energy awareness in the terminal design process and aid the developer in the design of energy efficient terminals. The three mechanisms are situated at the three abstraction levels in the design process:

- The Petri-net Models can be applied at the system level,
- The Cobra virtual platform gave a use case for energy awareness a platform level, aiding the system and software designer to optimize a platforms power consumption when developing software,
- At component level, the SESC simulator was enhanced with energy awareness and energy optimizations were obtained using Fuzzy Logic.

Together with deliverable D2.1 energy awareness has been considered both at networking level as well as at terminal level giving a plethora of possibilities to enhance the design process of networks in general towards energy efficiency.

These two deliverables have mainly been focussing on design time energy awareness; i.e. how to optimize the energy consumption when designing a system. In the second year of the CONSERN project, the focus will shift more towards run-time aspects for energy efficiency. How can a system learn to be energy efficient at run-time by for example calibration methods based on learning schemes? Another focus will be on what the implications are when all the methods developed in WP2 are used together.

6. References

[1] INFSO-ICT-257542 CONSERN Project, http://www.ict-consern.eu

[2] http://www.synopsys.com

[3] V. Derudder, B. Bougard, A. Couvreur, A. Dewilde, S. Dupont, L. Folens, L. Hollevoet, F. Naessens, D. Novo, P. Raghavan, T. Schuster, K. Stinkens, J.-W. Weijers, and L. Van der Perre, "A 200mbps+ 2.14nj/b digital baseband multi processor system-on-chip for sdrs," in Proc of Symposium on VLSI Circuits, June 2009.

[4] http://www.arm.com

[5] 'PIC16F84A datasheet', available at http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf

[6] S. Pollin, E. Lopez Estraviz, A. Antoun, P. Van Wesemael, L. Hollevoet, A. Bourdoux, A. Dejonghe, L. Van der Perre "Digital and analog solution for low-power multi-band sensing", IEEE International Dynamic Spectrum Access Networks Symposium – DySPAN, April 2010

[7] http://www.retarget.com

[8] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in Proc. IEEE Conf. on Field-Programmable Logic and its Applications (FPL), Lisbon, Portugal, Sept.2003, pp. 61–70.

[9] F. Naessens, V. Derudder, H. Cappelle, L. Hollevoet, P. Raghavan, M. Desmet, A.M. AbdelHamid, I. Vos, L. Folens, S. O'Loughlin, S. Singirikonda, S. Dupont, J.-W. Weijers, A. Dejonghe, and L. Van der Perre, "A 10.37 mm2 675 mw reconfigurable ldpc and turbo encoder and decoder for 802.11n, 802.16e and 3gpp-lte," in Proc. of Symposium on VLSI Circuits, 2010.

[10] Dipanjan Sengupta, Resve Saleh **"**Generalized Power-Delay Metrics in Deep Submicron CMOS Designs**",** IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 26, NO. 1, JANUARY 2007

[11] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A frame work for architectural power analysis and optimizations", in Proceedings of the 27th international Symposium Computer Architecture, pp. 83–94, 2000.

[12] Jokar Deris, Kaveh "Analysis and optimizations for modern processors' branch target buffer and cache memory", PhD thesis 2008

[13] http://www.hpl.hp.com/research/cacti/

[14] E. Patouni, A. Lilis, A. Merentitis, N. Alonistioti, C. Beaujean, D. Bourse, E. Nicollet, "Protocol Reconfiguration Schemes for Policy-based Equipment Management", in the Proceedings of the Vehicular Technology Conference (VTC), 25 – 28 September 2006, Montréal, Canada.

[15] A. Merentitis, E. Patouni, N. Alonistioti, M. Doubrava, "To Reconfigure or Not to Reconfigure: Cognitive Mechanisms for Mobile Devices Decision Making" IEEE Vehicular Technology (VTC) conference, 21 – 24 September 2008, Calgary, Canada.

[16] L. Giupponi, R. Agustí, J. Pérez-Romero, O. Sallent, "Joint radio resource management algorithm for multi-RAT networks", IEEE Global Telecommunications Conference (GLOBECOM), 2005.

[17]S. Buljore et al. “Introduction to IEEE P1900.4 Activities”, IEICE Transactions on Communications, Special Section on Cognitive Radio and Spectrum Sharing Technology (Invited Paper), Vol. E91-B, pp 2-9, Jan. 2008

[18]W. Wei and M. J. Wang, “Fuzzy-MOGA-based Traffic Signal Control at Intersection,” in Proc. of the International Conference on Machine Learning and Cybernetics, 2-5 Nov. 2003

[19]B. Arfi, “Linguistic Fuzzy-Logic Game Theory”, Journal of Conflict Resolution, Vol. 50, Feb. 2006, pp. 28-57.

[20]F. C. Rodriguez, D. M. Rodriguez, R. Soto and H. Maturino, “Position Location Assisted Multi-Valued Logic Handoff Algorithm”, in Proc of the Vehicular Technology Conference, 1999

[21]H. Dubreil, Z. Altman, V. Diascorn, J.-M. Picard, M. Clerc, “Particle swarm optimization of fuzzy logic controller for high quality RRM auto-tuning of UMTS networks” Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st.

[22]GIUPPONI (L.), Agustí (R.), Pérez-Romero (J.), Sallent (O.), “An Economic-driven Joint Radio Resource Management with User profile Differentiation in a Beyond 3G Cognitive Networks”, IEEE Global Telecommunications Conference (GLOBECOM), November, 2006.

[23]G. Raju, J. Zhou, and R. A. Kisner, “Hierarchical fuzzy control,” Journal of Applied Soft Computing, vol. 54, no. 5, pp. 1201–1216, 1991.

[24]F. Cheong , R. Lai, “Designing a hierarchical fuzzy logic controller using the differential evolution approach” International Journal of Control, vol. 7, no. 2, pp. 481–491, 2007.

[25]Aldrich, John (1995). "Correlations Genuine and Spurious in Pearson and Yule". Statistical Science 10 (4): 364–376.

[26]http://www.mathworks.com/help/techdoc/ref/corrcoef.html

[27]F.J.M. Velo, L. Baturone, S.S. Solano, A. Barriga, “Rapid design of fuzzy systems with Xfuzzy”, in Proc. Of the International Conference on Fuzzy Systems, pp. 342- 347, May 2003.

[28]“Part 11, Wireless Lan MAC and Phy specifications, Amendment 5: enhancements for higher throughput” , IEEE standard 802.11n, approved september 2009

[29]http://www2.imse-cnm.csic.es/Xfuzzy

[30]F. Vandeputte, L. Eeckhout, and K. D. Bosschere, “Offline Phase Analysis and Optimization for Multi-configuration Processors”, Proceedings of the international workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2005, pp. 202-211, Samos, Greece.

[31]K. Sigdel, M. Thompson, A. D. Pimentel, T. Stefanov, K. Bertels, "System-Level Design Space Exploration of Dynamic Reconfigurable Architectures", Proceedings of the international workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2008, pp. 279-288, Samos, Greece.

[32]S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta “The SPLASH-2 Programs: Characterization and Methodological Considerations”, Annual International Symposium on Computer Architecture (ISCA), pp. 24-36, 1995.

[33]David H. Bailey. FFT’s in External or Hierarchical Memory. Journal of Supercomputing, 4(1):23-35, March 1990.

[34]Steven Cameron Woo, Jaswinder Pal Singh, and John L. Hennessy. The Performance Advantages of Integrating Block Data Transfer in Cache-Coherent Multiprocessors. In Proceedings of the

Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 219-229, October 1994.

[35]V. Daga, G. Govindu1y, V. Prasanna1, S. Gangadharpalli and V. Sridhar, “Efficient Floating-point based Block LU Decomposition on FPGAs”, Proceedings of the international symposium on Field-programmable gate arrays, Monterey, California, USA, pp. 63-74, 2005.

[36]Y. Ding, M. Kandemir, M. J. Irwin, and P. Raghavan, “Adapting Application Mapping to Systematic Within-Die Process Variations on Chip Multiprocessors”, Proceedings of the International Conference on High Performance Embedded Architectures and Compilers (HiPEAC) 2009, Paphos, Cyprus, pp. 231-247.

[37]http://www.ee.lsu.edu/koppel/proteus/states.html#LU

[38]CONSERN Deliverable D2.1, ‘System Level Energy Optimization Solutions’

[39]CONSERN Deliverable D1.1, ‘Scenarios, Use Cases, and System Requirements’

[40]C. A. Petri, "Kommunikation mit Automaten (English translation: Technical Report RADC-TR-65-377, Vol.1, Suppl 1, Applied Data Research, Princeton, N.J.)," PhD thesis, University of Bonn, 1962

[41]K. Espensen, M. Kjeldsen, and L. Kristensen, "Modelling and Initial Validation of the DYMO Routing Protocol for Mobile Ad-Hoc Networks," Lecture Notes in Computer Science (LNCS)—Applications and Theory of Petri Nets: Springer-Verlag, vol. 5062, pp. 152–170, 2008.

[42]A. Giua and C. Seatzu, “Modeling and supervisory control of railway networks using Petri nets,” IEEE Transactions on Automation Science and Engineering, vol. 5, iss. 3, pp. 431–445, 2008.

[43]Object Management Group (OMG), "Unified Modeling Language (OMG UML), Superstructure Specification (ID: formal/2007-11-02)," vol. 2.1.2, 2007.

[44]R. A. De Millo, R. J. Lipton, and A. J. Perlis, "Social processes and proofs of theorems and programs," Communications of the ACM, vol. 22, iss. 5, pp. 271–280, 1979.

[45]T. Murata, "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, vol. 77, iss. 4, pp. 541–580, 1989.

[46]N. J. Dingle, "Production of the Extensible Petri Net Editor/Animator "Medusa"," MSc Degree in Computing Science thesis, Department of Computing, Imperial College of Science, Technology and Medicine (University of London), 2001.

[47]G. von Bochmann and C. A. Sunshine, "Formal Methods in Communication Protocol Design," IEEE Transactions on Communications, vol. 28, iss. 4, pp. 624–631, 1980.

[48]D. Sidhu, A. Chung, and T. P. Blumer, "Experience with formal methods in protocol development," ACMSIGCOMM Computer Communication Review, vol. 21, iss. 2, pp. 81–101, 1991.

[49]F. Bause and P. S. Kritzinger, Stochastic Petri Nets: An Introduction to the Theory, 2nd ed: Vieweg, 2002.

[50]W. Jürgensen and S. T. Vuong, "Formal specification and validation of ISO transport protocol components, using Petri-nets," Proceedings of the ACM SIGCOMM symposium on Communications architectures and protocols: tutorials & symposium, (Montréal, Quebec, Canada, United States), 1984.

[51]G. Berthelot and R. Terrat, "Petri Nets Theory for the Correctness of Protocols," IEEE Transactions on Communications, vol. 30, iss. 12, pp. 2497–2505, 1982.

[52]K. Jensen, “Colored Petri nets. Basic concepts, analysis methods and practical use”, 2nd edition, Springer-Verlag GmbH, 1997.

[53]R. Valk, “Petri Nets as token objects – an introduction of elementary object nets”, Lecture Notes in Computer Science (LNCS)—19th Int. Conf. Application and Theory of Petri Nets (ICATPN’98), Lisbon, Portugal, Springer-Verlag, vol. 1420, pp. 1–24, 1998.

[54]J. Hillston, "Process algebras for quantitative analysis," 20th Annual IEEE symposium on Logic in Computer Science, (LICS 2005), Chicago, IL, United States, 2005.

[55]C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Ph.D. thesis, Dept. Of Electrical Engineering, Massachusetts Institute of Technology, 1974.

[56]O. Kummer, "Introduction to Petri Nets and Reference Nets," Sozionik Aktuell, vol. 1, pp. 1–9, 2001.

[57]O. Kummer, F. Wienberg, and M. Duvigneau, "Renew 2.1—User Guide," University of Hamburg, Department for Informatics, Theoretical Foundations Group, Distributed Systems Group, 2006.

[58]M. Ajmone Marsan and G. Chiola, "On Petri Nets with Deterministic and Exponential Transition Firing Times," Lecture Notes in Computer Science (LNCS)—Advances in Petri Nets: 7th European Workshop on Application and Theory of Petri Nets, vol. 266, pp. 132–145, Springer-Verlag, 1987.

[59]A. Heindl and R. German, “Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets”, Elsevier’s Performance Evaluation, W. Bux (Ed.), Elsevier Science B.V., vol. 44, pp. 139–164, 2001.

[60]E.B. Sloane and V. Gehlot, “Ensuring patient safety by using colored Petri net simulation in the design of heterogeneous, multi-vendor, integrated, life-critical wireless (802.x) patient care device networks’. IEEE-EMBS 2005. 27th Annual Int. Conf. Engineering in Medicine and Biology Society, pp. 162–165, 2005.

[61]J. L. Peterson, "Petri Net Theory and the Modeling of Systems," ISBN ISBN 0-13-661983-5, PrenticeHall, 1981.

[62]J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Derivation of passage-time densities in PEPA models using ipc: the imperial PEPA compiler," 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS), pp. 344–351, 2003.

[63]N. Akharware, "PIPE2: Platform Independent Petri Net Editor," MSc Degree in Computing Science thesis, Department of Computing, Imperial College of Science, Technology and Medicine (University of London), 2005.

[64]B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA—Construction of Abstract State Spaces for Petri Nets and Time Petri Nets," International Journal of Production Research, vol. 42, iss. 14, pp. 2741–2756, 2004.

[65]B. Berthomieu and F. Vernadat, "Time Petri Nets Analysis with TINA," Proceedings of 3rd Int. Conf. on The Quantitative Evaluation of Systems (QEST), Riverside, California, USA, pp. 123–124, 2006.

[66]T. A. Lewis and R. J. Haines, “Formal Verification to Enhance Evolution of Protocols”, Genetic and Evolutionary Computation Conference (GECCO), Montréal, Canada, July 2009.

[67]A. Dunkels, F. Osterlind, N. Tsiftes, Z. He, *“Software-based On-line Energy Estimation for Sensor Nodes”*, Proceedings of Fourth Workshop on Embedded Networked Sensors (EmNets) June 25-26 207, Cork, Ireland.

[68]Declerck, J.; Raghavan, P.; Naessens, F.; Vander Aa, T.; Hollevoet, L.; Dejonghe, A. and Van der Perre, L., "SDR platform for 802.11n and 3-GP-LTE", International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation - SAMOS X, Samos, Greece, July 2010.

[69]http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

[70]http://www.arm.com

[71]http://www.3gpp.org/

[72]http://grouper.ieee.org/groups/802/16/

[73]http://www.dvb.org/

[74]http://www.mathworks.com/

[75]Declerck J., Umans E., Dejonghe A., Trautmann M., Glassee M., Van der Perre L., "A software development and validation framework for SDR platforms", SDRForum Technical Conference, Washington, USA, Octobre 2008

[76]Berrou, C.; Glavieux, A.; Thitimajshima, P.; "Near Shannon limit error-correcting coding and decoding: Turbo-codes"; IEEE International Conference on Communications, 1993

[77]http://carbondesignsystems.com/

[78]http://en.wikipedia.org/wiki/Transaction-level_modeling Leonardo, A., et al., Mapping live sequence chart to coloured petri nets for analysis and verification of embedded systems. SIGSOFT Softw. Eng. Notes, 2006. 31(3): p. 1-25.

[79]Hume: a Concurrent Language with Bounded Time and Space Behaviour Kevin Hammond. Proc. 7th IEEE International Conference on Electronic Control Systems (ICECS 2K), Lebanon, 2000, pp. 407-411

[80]Kevin, H., F. Christian, and H. Reinhold, Towards formally verifiable resource bounds for real-time embedded systems. SIGBED Rev., 2006. 3(4): p. 27-36.

[81]Tuominen, J., T. Säntti, and J. Plosila. Towards a Formal Power Estimation Framework for Hardware Systems. in Proceedings of the International Symposium on System-on-Chip. 2005.