FARAMIR
Mapping Your Radio Environment

2007 - 2013

Flexible and Spectrum Aware Radio Access through Measurements and Modelling in Cognitive Radio Systems

FARAMIR

Document Number D3.2

Spectrum Sensing Engine and Prototype Measurements

Contractual date of delivery to the CEC:	30.06.2011
Actual date of delivery to the CEC:	30.06.2011
Project Number and Acronym:	248351 – FARAMIR
Editor:	Antoine Dejonghe (IMEC)
Authors:	Antoine Dejonghe (IMEC), Peter Van Wesemael (IMEC), Mattias Desmet (IMEC), Sofie Pollin (IMEC), Lieven Hollevoet (IMEC), Daniel Denkovski (UKIM), Valentin Rakovic (UKIM), Mihajlo Pavloski (UKIM), Konstantin Chomu (UKIM), Vladimir Atanasovski (UKIM), Liljana Gavrilovska (UKIM), Stephen Wang (TREL)
Participants:	IMEC, RWTH, UKIM, TREL
Workpackage:	WP3
Security:	PU
Nature:	R, P
Version:	1.0
Total number of pages:	78

Abstract:

The description of the hardware platform and the major software components used for the FARAMIR spectrum sensing engine are the main items covered in this deliverable.

Sensing performance is evaluated and compared with high-end spectrum measurement-capable device.

Keywords: sensing prototyping, measurement-based performance evaluation

Document Revision History

Version	*Date*	*Author*	*Summary of main changes*
0.1	01.06.2011	A. Dejonghe (IMEC)	Initial table of content
0.2	06.06.2011	A. Dejonghe (IMEC)	Initial inputs section 2
0.3	15.06.2011	A. Dejonghe, L. Hollevoet, P. Van Wesemael, M. Desmet (IMEC)	Extended inputs section 2
0.4	24.06.2011	P. Van Wesemael, D. Denkovski, V. Rakovic, M. Pavloski, K. Chomu, V. Atanasovski, L. Gavrilovska, S. Wang (IMEC, UKIM, TRL)	Initial Merge of IMEC, UKIM and TREL input
0.5	27.06.2011	P. Van Wesemael, M. Desmet (IMEC)	Correct and add figure labels, references and links. Completed IMEC contribution on SPIDER board
0.6	28.06.2011	A. Dejonghe (IMEC)	First overall consolidation
1.0	30.06.2001	J. Riihijärvi (RWTH)	Final version
1.0f	30.06.2001	P. Mähönen (RWTH)	Coordinator review and approval

Table of Contents

1 Introduction

The objective of the FARAMIR project is to research and develop techniques for increasing the radio environmental and spectral awareness of future wireless systems. The key enabling technology developed in the project towards this purpose is Radio Environment Maps (REMs), which can be understood as knowledge bases in which cognitive radios and store and access information on the environment and other wireless systems.

To enable distributed sensing and eventually map the spectrum environment, radio platforms should ideally enable flexible spectrum sensing according to different scenarios, bands and algorithms. Similar engines are being developed for military applications, but there has not been much development work towards affordable low-power solutions that would be suitable for integration into commercial wireless terminals having very different boundary conditions in terms of price and performance compared to military equipment. In FARAMIR, an important target is the development and prototyping of such a reconfigurable engine for multi-purpose spectrum sensing within the power and cost constraints of mobile devices. Another key issue is the development of the necessary software components for integration of the spectrum sensing engine into the overall FARAMIR Radio Environment Map architecture and the corresponding prototype.

The focus of this document is to describe the prototype design and performance of the hardware platform and the major software components used for the FARAMIR spectrum sensing engine. The designs of the hardware and software components of the engine are presented in Chapter 2 of the document. Sensing performance of the prototype is also evaluated and compared with high-end spectrum measurement-capable devices. The corresponding measurement setups and results are documented in Chapter 3, together with an overview of various commercial off-the-shelf sensing technologies used in our prototyping efforts and measurement-driven discussion on their performance. Finally, the conclusions are drawn in Chapter 4.

2 FARAMIR Sensing Engine Prototype

2.1 Reconfigurable Sensing Engine

The proposed FARAMIR sensing engine builds on a flexible analog front-end and a digital front-end for sensing, as depicted in Figure 1. It can be connected further to a baseband processor, resulting in sensing capable wireless devices.

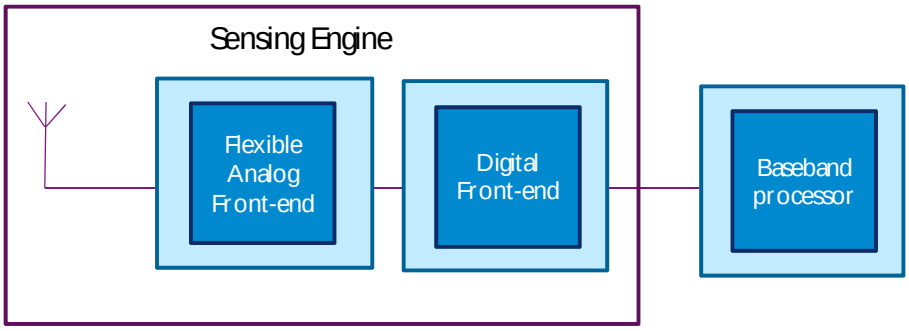

Figure 1: The structure of the integrated sensing engine.

The reconfigurable analog front-end allows scanning the spectrum in a broad range of frequency bands. Since the basic task of a spectrum sensing engine is to scan all frequency bands for the presence of signals with several possible bandwidths, the functionality of the analog front-end actually requires the use of a Software-Defined Radio (SDR).

The programmable digital front-end features the processing capabilities needed to analyze the signals in the scanned band, such as:

- Being the interface between the analog frontend and the baseband (front-end data and control interfaces).
- Performing the signal acquisition and coarse time synchronization for the targeted standards.
- Multiple channelization branches, for simultaneous sensing and reception of different frequencies.
- Support for multi-band energy detection and feature-based sensing, relying on autocorrelation.

The roadmap for the development of the sensing engine is depicted in Figure 2. It is organized in three phases that will be described in the next subsections:

- Phase 1: Standalone analog front-end prototype (section 2.2) [1]

- Phase 2: Standalone digital front-end prototype (section 2.3)
- Phase 3: Integrated sensing engine prototype (section 2.4)

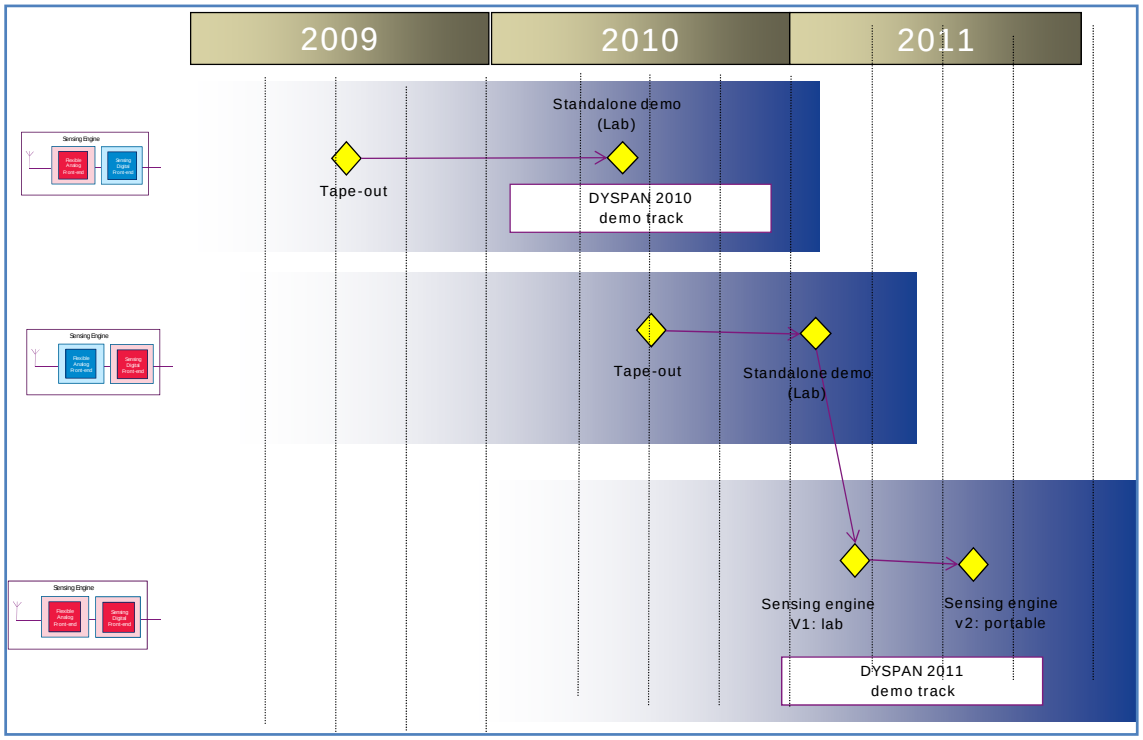

Figure 2: Integrated sensing engine: Prototyping roadmap.

2.2 Phase 1: Standalone Analog Front-end Setup

Scaldio Chip: The flexible analog front-end is implemented with the Scaldio RF IC, and was already demonstrated at Dyspan 2010 [1]. Scaldio is a fully reconfigurable analog transceiver implemented in 40nm CMOS technology. It covers all functionality from the LNA to the ADC in a programmable way [2], as illustrated in Figure 3. The receiver RF operating frequency is programmable from 0.1 to 6 GHz and the channel bandwidth is programmable between 1 and 40 MHz. The high-speed Network-On-Chip used for configuration and the fast settling time of the PLL enable fast switching between different RF frequencies and channel bandwidths. The low noise figure, 2.4 to 4 dB below 3 GHz, together with low power consumption, 30 to 90 mA, makes this reconfigurable analog front-end very well suited for low power flexible sensing.

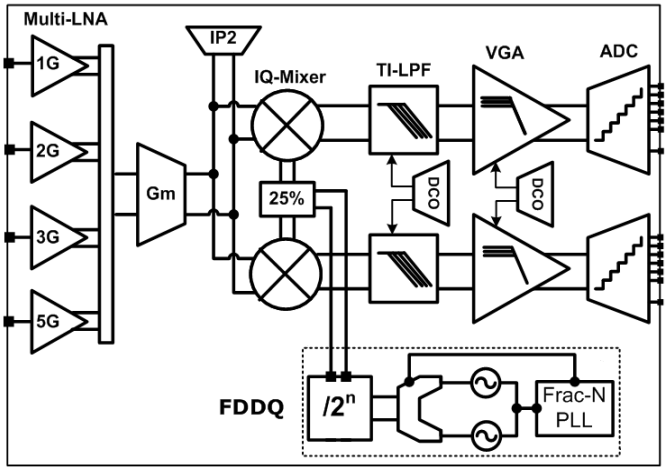

Figure 3: Scaldio flexible analog front-end architecture.

Scaldio board: The corresponding prototype board is depicted in Figure 4. This board does not contain any active components besides the Scaldio chip.

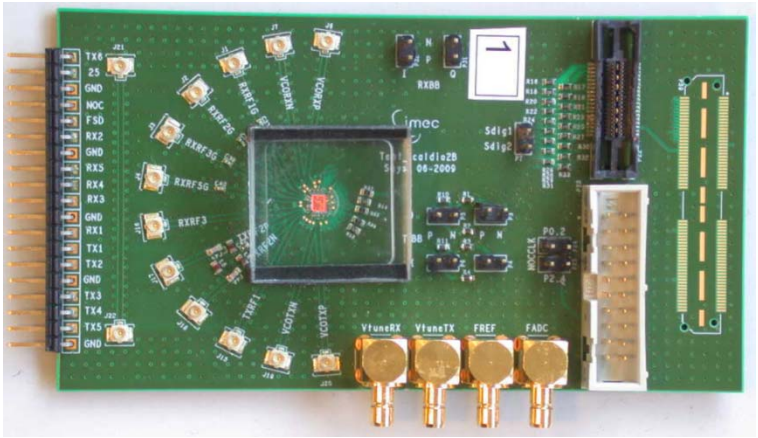

Figure 4: Scaldio hardware board.

For testing and debugging the receiver section of the chip the following connections are available:

- 4 UFL connections to connect the antenna to the corresponding LNA
- 2 UFL connections which can be used for
    - Differential output of the receiver LO signal. This signal can be used to connect to another Scaldio board for MIMO operation with common LO signal. Obviously monitoring this signal can also be very useful during initial debugging.
    - Differential input of receiver LO signal. This input can be used during MIMO operation in case a common LO signal is desired. During initial testing this input can also be used to connect high accuracy test equipment to generate the LO signal

- 2 pin headers with 2 pins which can used to:
    - Monitor the differential baseband signal at different stages of the baseband section
    - Inject differential baseband signals into different stages of the baseband section, including just before the ADC. It is also through these pins that ADC input stage is put at the correct bias voltage during the ADC calibration routine
- 118 pin QSH Samtec connector. This connector contains the digital output signals from the ADC, including a clock signal on which the data can be sampled. The data signals consists of 10 bits plus one correction bit for the LSB.
- 38 pins Mictor connector which also contains the ADC data signal including clock signal. This connector can be directly connected to a logic analyzer for testing and ADC performance evaluation.

For testing and debugging the transmitter part of the chip the following connectors are available:

- 2 UFL connectors connecting to the corresponding PPA output on chip
- 2 pin headers with two pins to inject the differential complex baseband signal in the RFIC
- 2 UFL connections which can be used for
    - Differential output of the transmitter LO signal. This signal can be used to connect to another Scaldio board for MIMO operation with common LO signal. Obviously monitoring this signal can also be very useful during initial debugging.
    - Differential input of transmitter LO signal. This input can be used during MIMO operation in case a common LO signal is desired. During initial testing this input can also be used to connect high accuracy test equipment to generate the LO signal

The RFIC has on Network On Chip (NOC) for configuration, this interface consists of 5 signals which can be accessed either through the 118 pins QSH Samtec connector or through the 20 pins IDC header. A 2 pins pin header is available to connect to the multipurpose sdig signals. These signals are also available on the 118 pins QSH Samtec connector. The signals are needed for the calibration of the Voltage Controlled Oscillators (VCO) and can be used for monitoring the Phase Locked Loop (PLL) operation. The PCB also contains 4 SMB jacks, these contain the following signals:

- VtuneRx: tuning voltage of the VCO of the receiver, this can be used for monitoring purposes or for connecting an external PLL.
- VtuneTx: tuning voltage of the VCO of the transmitter, this can be used for monitoring purposes or for connection an external PLL.
- FREF: input for reference clock for both receiver and transmitter PLL.
- FADC: input for ADC sampling clock

Scaldio setup: The resulting prototyping setup is depicted in Figure 5 and Figure 6. We refer the reader to [1] for additional details.

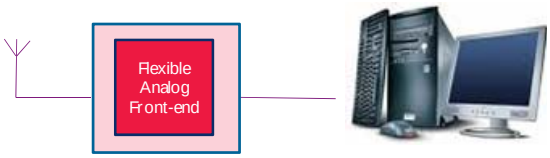

Figure 5: Scaldio standalone prototyping setup.

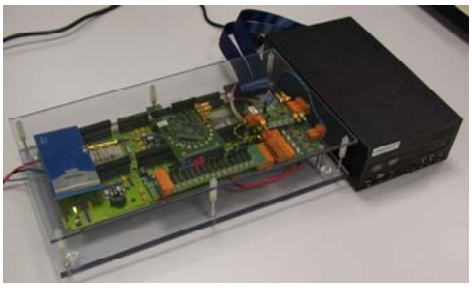

Figure 6: Scaldio standalone prototyping setup.

2.3 Phase 2: Standalone Digital Front-End Setup

This section presents all aspects of the DIgital Frontend For spectrum Sensing (DIFFS) standalone prototype, which is depicted schematically in Figure 7. This setup is used for debugging, testing and demonstrating the synchronization and sensing capabilities of the DIFFS chip. It consists of the following components that will be detailed in the next sections:

- Component 1, DIFFS chip
- Component 2, DIFFS PCB: The DIFFS PCB was designed and produced specifically to host the DIFFS chip and to enable communication with it.
- Components 3, FPGA interconnect board: An FPGA based motherboard is used to make an interconnection between the PC and the PCB.
- Component 4, Control system on host PC: A host PC is used to configure the DIFFS and the PCB and to control the data-streams to and from the chip.

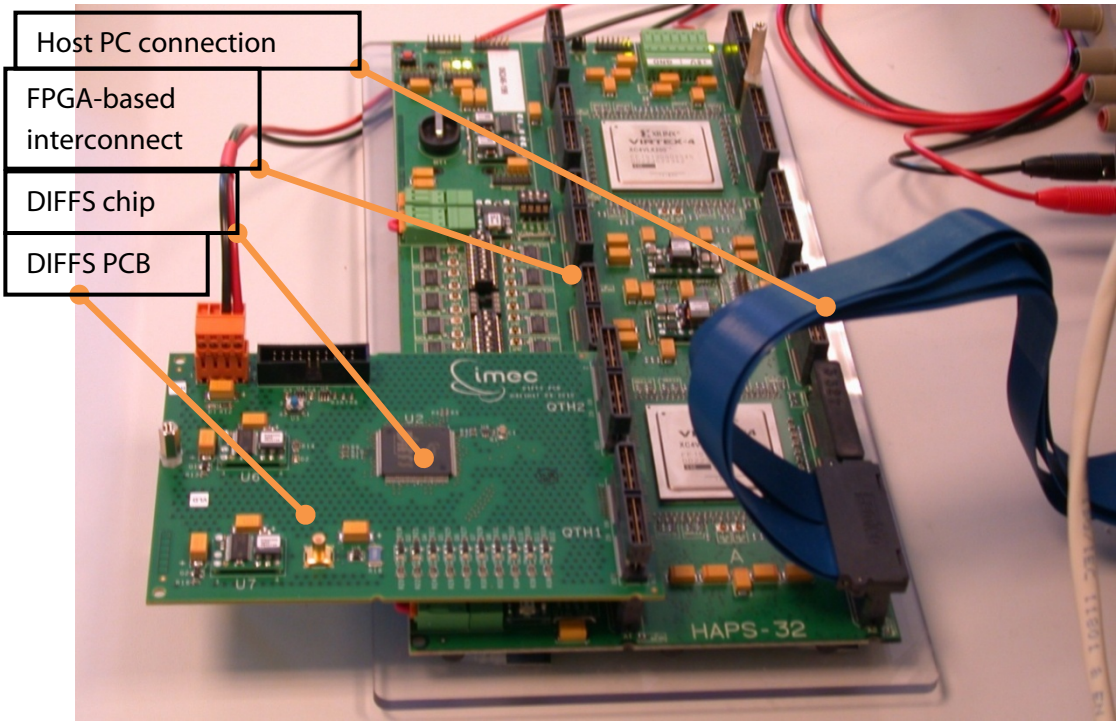

Figure 7: Picture of the DIFFS Prototype hardware.

2.3.1 Component 1: DIFFS chip

The sensing digital front-end is implemented with the DIFFS architecture, which has been implemented as a chip in 65 nm CMOS technology. The DIFFS fits in the radio receiver between the analog frontend and the host interface. Figure 8 depicts the high-level block diagram of the device. The analog frontend is connected to the top input, and received data after sensing can be sent to the host from the buffer. Sensing results from more advanced sensing algorithms are communicated through a control interface.

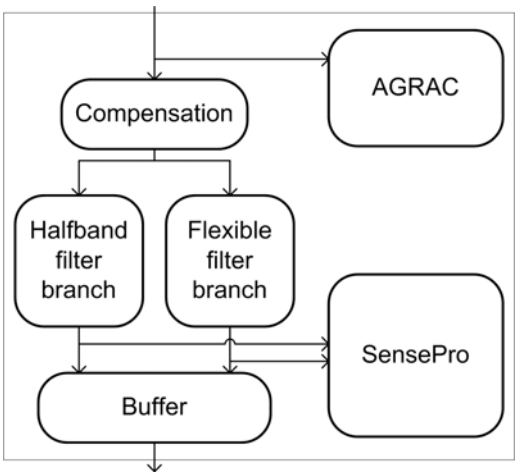

Figure 8: Digital frontend for sensing block diagram.

The internal dataflow is organized as follows:

- Data is received from the Front End
- This data is monitored for signal presence (power-based) by the Automatic Gain and Resource Activity Controller (AGRAC).
- The AGRAC takes care of setting the gain of the analog frontend to ensure that the received signal amplitude does not exceed the dynamic range of the ADC.
- When a signal of sufficient power is received, DC offset and IQ mismatch are compensated and the filters, Sensing Processor (SensePro) and Host Interface are enabled. The Sensing Processor is responsible for packet detection, time synchronization and sensing.
- In case of synchronization, the Sensing Processor will communicate the result of the synchronization operation to the AGRAC and the AGRAC will interrupt the host. The host can then read data from the data interface of DIFFS.
- In case of a sensing operation, the Sensing Processor can interrupt the host and communicate the results of the sensing operation to the host through the control interface.

The DIFFS has 4 functional interfaces: front end interface, host interface, AGC/NOC interface and control interface. Furthermore there is a JTAG debug interface. The DIFFS has 2 power domains: the core domain, at 1.0 V, and the I/O domain, running at 2.5 V. Each power domain has multiple dedicated supply pins. In the next subsections, the various components of the DIFFS are described more in detail.

AGRAC: The Automatic Gain and Resource Activity Controller (AGRAC) controls the DIFFS. It serves two tasks: determining the gain settings of the analog frontend upon reception of a radio signal, and controlling the enable signals of the other subblocks to implement the hierarchical wakeup mechanism. This principle of hierarchical wakeup exploits the fact that only the blocks consuming minimum power are always active. The synchronization and sensing engine (SensePro), which has an inherently higher power consumption, is only activated once signal power is detected. The final stage of the hierarchical wakeup, the notification of the host, only happens after synchronization is found or a sensing result is available. The AGRAC is implemented as an 8-bit microcontroller that is compatible with an industry-standard C toolflow to ease programming. The core of the processor runs at the speed of the incoming samples and contains a peripheral that measures the received signal power and the DC offset. Depending on the monitoring objectives of the CR, the AGRAC will enable sensing and/or synchronization only when energy is detected, or always.

Compensation and filtering: The incoming samples are first sent through a block that compensates for frontend non-idealities. In this block, the remaining DC offset is digitally removed and I/Q imbalance compensation is applied. Subsequently, two parallel filter branches can operate on the data. One filter branch consists of two stages of power-optimized 23-taps half-band filters

that perform low-pass filtering and downsampling by factor 2 or 4. These filters are implemented as polyphase structures. Canonic Signed Digit recoding and Common Sub-expression Elimination were applied to minimize power consumption. The second filter branch is a fully programmable flexible filter branch. This branch supports down-conversion, band selection, and non-integer rate conversion (Figure 9). The down-conversion is performed by mixing the received signal with the output of a programmable CORDIC. The CORDIC output quantization is 12 bits and 16 iterations are required to reach the desired 1 kHz accuracy for LTE synchronization for a received signal bandwidth of 100 MHz. The subsequent CIC filter implements the first downsampling step. The filter can use at most 3 stages, 4 delay elements and a decimation factor of 4. The next stage in the filter chain is a programmable FIR filter with a 21-taps symmetrical implementation. This filter provides the additional filtering capabilities required for adjacent channel suppression. The last step in the flexible filter branch is a cubic form Lagrange interpolator. This block supports non-integer down-sampling rates down to 1/16th. All filter blocks are implemented in full precision, and a quantization selection block after each stage allows the algorithm designer to select the most relevant bits from the output of the filter.

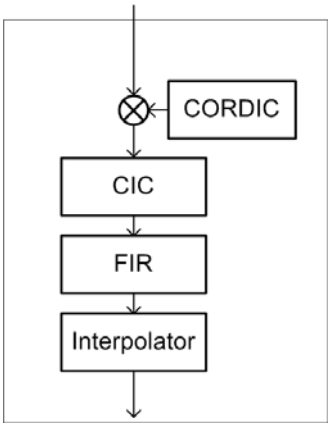

Figure 9: Flexible filter branch block diagram.

Synchronization and Sensing engine: After passing through the filters, the data enters the SensePro synchronization and sensing engine. This is an ASIP that contains a 32-way SIMD processor with scalar slot and three accelerator cores (Figure 10). The SIMD processor is used for all the flexible operations that are required for different sensing algorithms. The scalar slot is used for loop control. The accelerator cores handle the tasks that do not map efficiently on the SIMD, such as 128-point Fourier transform, vector rotation and correlation. The correlator is specifically used for synchronization, as this function needs to happen in a low power fashion for an efficient hierarchical wakeup of the radio. The SIMD processor datapath supports complex data samples and every SIMD slot is 2x16 bits wide. All vector instructions are targeted towards synchronization and sensing specific functionality so that the various algorithms can be mapped efficiently. As the processor only contains a scalar and a SIMD slot, firmware development for the SensePro is fairly straightforward. The SensePro runs at the incoming sample speed and can clock-gate itself when it is waiting for data from the correlator that generates the input vectors from the input samples.

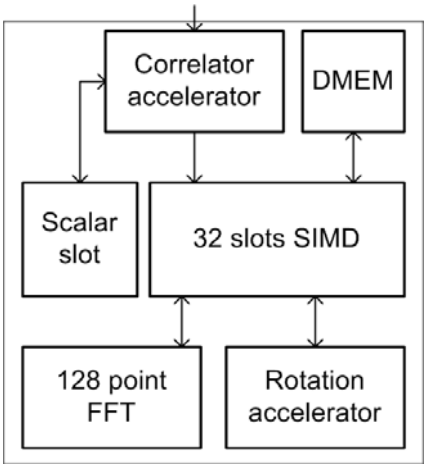

Figure 10: SensePro block diagram.

Output buffer: The output buffer is a key element in the architecture when the DFE is used for data reception. The buffer is implemented as a circular data buffer with a programmable read pointer. When synchronization is required, both the SensePro and the buffer are enabled simultaneously by the AGRAC. The SensePro then searches for sync while the received and filtered samples are being stored in the buffer. When synchronization is detected, the SensePro passes the absolute synchronization point index to the buffer to update the read pointer. Subsequently, the AGRAC interrupts the host to notify that data is available for reading. When the host starts reading out samples from the buffer, only the samples starting from the synchronization point on are transferred.

2.3.2 Component 2: DIFFS PCB

The PCB hosts the DIFFS and enables communication with the chip and with all peripherals present on the PCB. It has an oscillator, some DC-DC convertors with current-sensing monitors and a number of connectors on board. The purpose of the DIFFS PCB is threefold.

- First of all, the PCB has DC-DC convertors to generate the voltages for the supply pins. For this purpose, 2 DC-DC convertors are present on the PCB to generate the required voltages, together with some decoupling capacitors.
- Secondly, all input and output signals of the DIFFS are routed to two HAPS I/O connectors that enable an interconnection with the FPGA motherboard. This way, the DIFFS signals can be read out and/or controlled via the FPGA-interface. Some of the output signals are also routed to a connector to allow reading them out with stand-alone measurement equipment like an oscilloscope or a logic analyzer.
- Thirdly, the PCB provides the possibility to monitor the power consumption of the core power domain of the DIFFS. This is done by measuring the current that is drawn from the power supply. The current is represented by a voltage drop over a current-sense resistor.

This voltage is then measured by a current-sense amplifier and can be monitored through an SMB-connector.

A picture of the DIFFS PCB with the DIFFS chip mounted is shown in Figure.11. The function of each component is explained in this section.

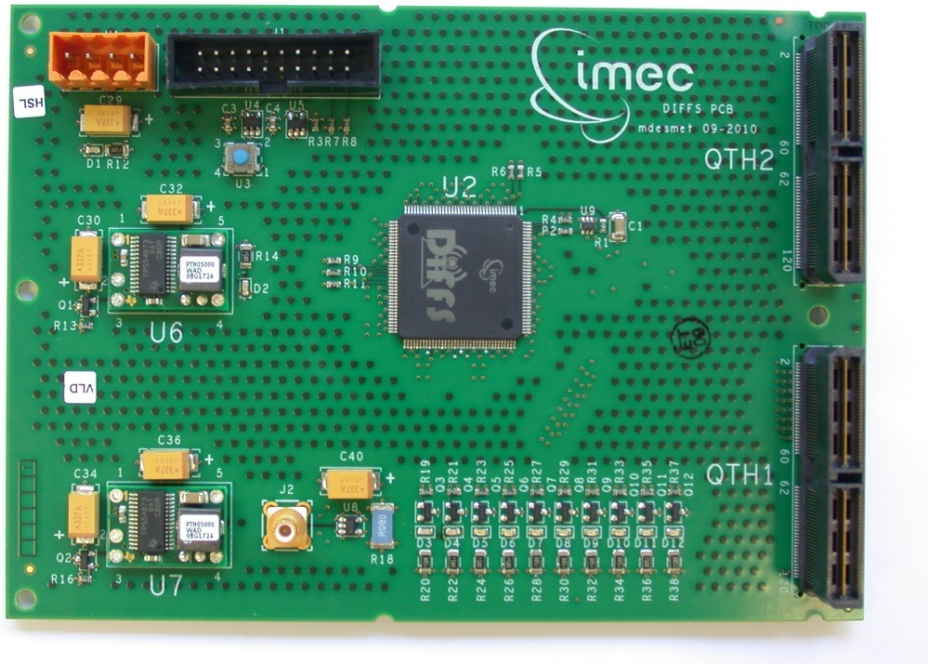

Figure.11: The DIFFS PCB.

2.3.3 Component 3: Interconnect FPGA platform

The HAPS-32 FPGA board connects the different components of the prototype. How this is done and how the FPGA-interface looks like is discussed in this chapter. An off-the-shelf HAPS-32 FPGA board has been selected for implementing the interconnect system between the host PC and the DIFFS PCB. The board features 2 Xilinx Virtex-4 LX200-10 FPGAs, and runs on a single 5V supply voltage. It is depicted in Figure 12.

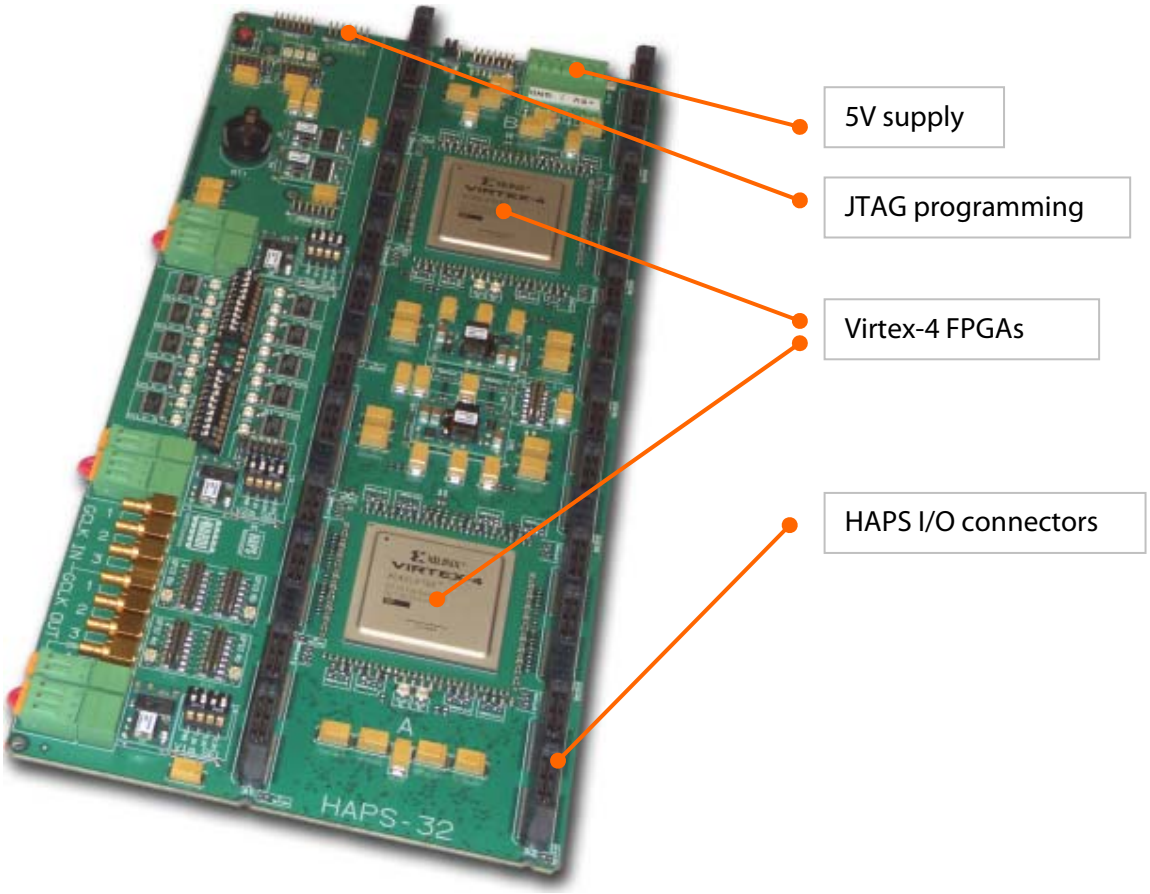

Figure 12: HAPS-32 commercial FPGA board.

The board is equipped with 12 HAPS I/O connectors for flexible system expansion. The DIFFS PCB design is based on these HAPS I/O connectors. Each connector features 119 I/O signals. The connections are 'through board', meaning that a mating connector is available at the top/bottom side of the PCB. This way, boards can be stacked on top of each other to create complex setups or to expand FPGA resources.

One of the I/O connections is used for communication with the host PC. The HAPS FPGA board is configured as PCI target. This means that all transactions between the host PC and the board are initiated by the host PC. It is not possible for the board itself to start a transaction. This implies that the PC has to be aware of what the next action is it should perform. Obviously, interrupts originating from the FPGA-interface and/or the DIFFS have to be polled by the host PC.

The board is programmed using dedicated software from Xilinx. The software communicates with the FPGAs over the JTAG-interface using a specialized USB-to-JTAG cable. Software detects the two FPGAs in a chain and bitfiles are assigned. After downloading the bitfiles to the board, the JTAG interface can be used for debugging purposes.

The FPGA configurations are stored in volatile memory and so the board needs reprogramming after a power down. However, it is possible to add a daughter board to the HAPS from which the FPGA configuration can be loaded from compact flash.

Overview: A schematic representation of the connector usage of the HAPS-32 board is shown in Figure 13. As we can see, this is the same as in Figure 7.

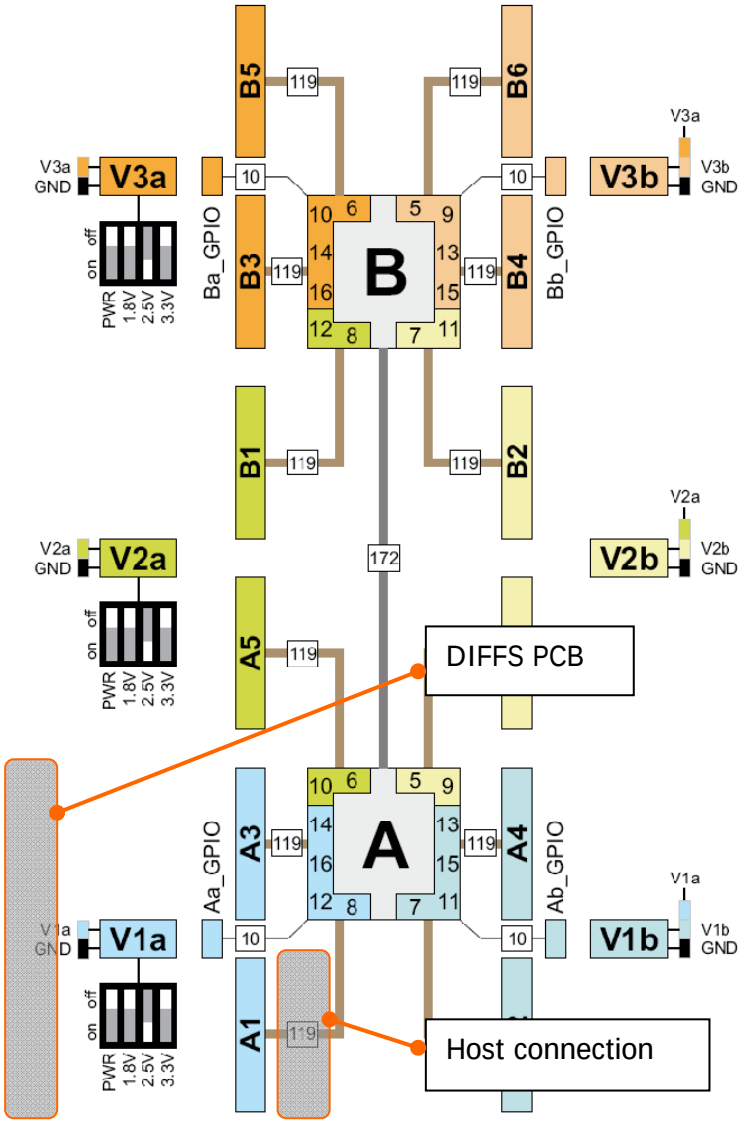

Figure 13: HAPS-32 connector usage.

It is clear that all signals are routed to FPGA A. Consequently, the FPGA-interface is implemented on FPGA A as well. Next to this, all Block RAM resources of FPGA B are used as well, to create a FIFO for storing samples. In Figure 14, a schematic overview is given of how the FPGA-interface is used to connect all hardware components of the design.

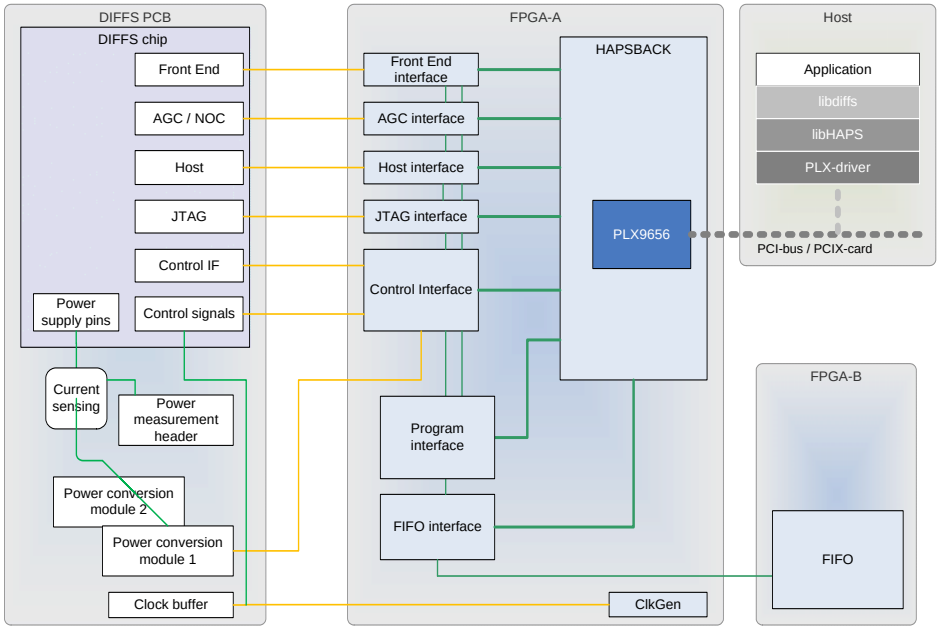

Figure 14: Schematic overview of the FPGA-interface in the DIFFS hardware prototype.

A first component of the FPGA-interface is the ClkGen block, which is responsible for generating the different clock frequencies from the incoming 40 MHz clock. For this, it uses dedicated DCM and BUFG resources found in Xilinx FPGAs. The most important clocks for internal use that are generated are a 40 MHz and an 80 MHz clock as Front End clocks, and a 50 MHz clock as Host interface clock.

The FPGA-interface also houses the PCI communication module called HAPSBACK (HAPS BACKplane) through which it communicates with the host PC. On the host, applications run on top of a set of drivers and abstraction layers. Software can communicate through the PCI card to the setup. The HAPSBACK is discussed in the next section.

Furthermore, there are some other interfaces. Some of them are communicating with the DIFFS, some of them are communicating with the DIFFS PCB, and others are only communicating internally on the FPGA. These interfaces are discussed later in this chapter.

HAPSBACK interface: The HAPSBACK is a wrapper around the local bus signals coming from the PLX PCI9656 chip on the PCIX card. It has 2 types of interfaces towards the internals of the FPGA:

- FIFO interfaces;
- Synchronous interfaces; these are address/data interfaces towards other interfaces on the FPGA. Synchronous means they are working synchronously with respect to the master bridge, but obviously their connection with the other interfaces can be asynchronous.

A block diagram of the HAPSBACK is depicted in Figure 15.

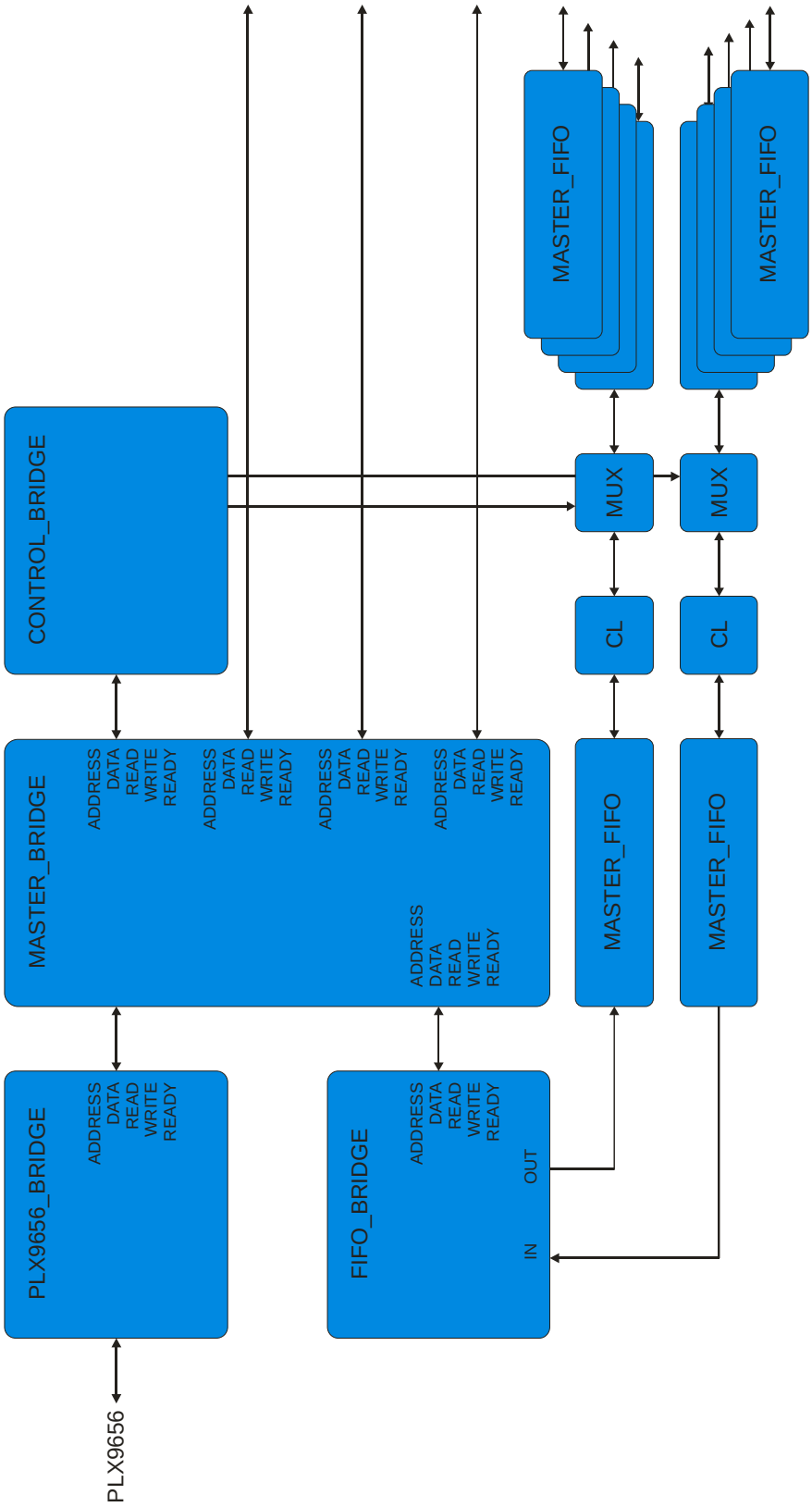

Figure 15: HAPSBACK block diagram.

The HAPSBACK starts with an interface to the PLX PCI 9656 local bus, coming from the PCI card connection on HAPS connector A2 (see Figure 13). It implements a state machine for translating read and write commands on the PLX bus to an simpler address/data interface. A handshake signal (ready) is available for flow control.

The master_bridge block is a mux/demux of the address/data interface from/to multiple targets. In the current configuration, there is 1 control bridge interface and one FIFO bridge interface implemented. Next to these, there is a Program Interface and a Control Interface (not to be confused with control bridge interface). This can easily be extended.

A zoom-in on the HAPSBACK control bridge interface is shown in Figure 16. Only the communication between the control bridge and the master bridge is shown here.

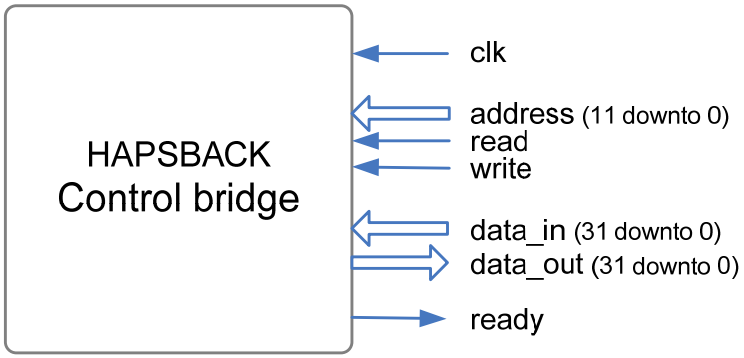

Figure 16: HAPSBACK control bridge.

The control bridge interface has a 12-bit address bus, capable of addressing 4 kByte. The data interface has a split data_in and data_out bus, each 32-bit wide. The clk input allows for asynchronous operation between the preferred clock domain of the IP-block and the (default) 40 MHz clock of the HAPSBACK itself. The read and write signals indicate the type of transaction while the ready signal is used again for flow control. Once the transaction has been registered inside the connected target block, it asserts the ready signal for 1 clock period.

The HAPBACK FIFO bridge is depicted in Figure 17. The topside shows a board-to-host FIFO interface (host read) while below there is a host-to-board FIFO interface (host write). The data interfaces of the FIFO are 32-bit wide. Data flow control is implemented with read/write signals and an empty or full signal respectively.

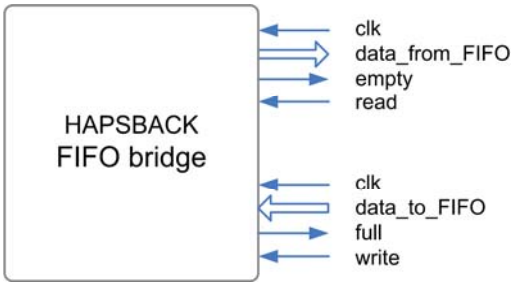

Figure 17: HAPSBACK FIFO bridge.

If there is no data available when the host performs a read, the software function will block until the required data becomes available. The same happens for host write operations to the FIFO interface when there is a full signal.

Unlike the synchronous bridges, there is only one FIFO (bridge) interface. To serve multiple FIFOs, FIFO signals are muxed/demuxed into several FIFO I/O signals. It is up to the software programmer to set the correct mux position before reading or writing data to the FIFO interface (see Figure 15).

To sum up, the purpose of the HAPSBACK could be described as follows: it is responsible for translating and routing a command or request from the host PC to the correct interface. This can be writing or reading data through the FIFO bridge to/from the FIFOs, or performing a write or a read command to one of the other interfaces.

DIFFS Interfaces: The DIFFS interfaces are those interfaces that communicate with the DIFFS chip. These are:

- The Front End interface;
- The Host interface;
- The AGC interface;
- The Control Interface;
- The JTAG interface.

All interfaces will be discussed in this section, except for the JTAG interface, because it is not used so far.

The Front End interface is controlled by an enable signal, which is set by the Program Interface. In the current tests, the interface runs at either 40 MHz or 80 MHz. This clock signal is also sent to the DIFFS Front End as a sample clock for the incoming IQ-samples. These samples are read from an asynchronous FIFO. A schematic representation can be found in Figure 18.

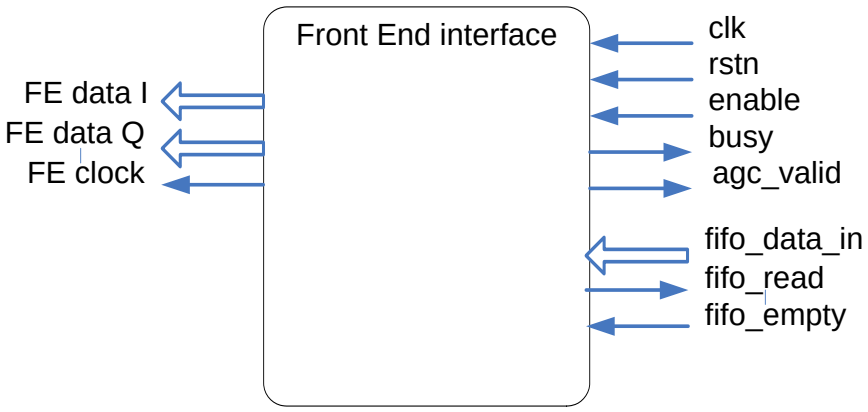

Figure 18: Schematic representation of the Front End interface.

The Host interface is controlled by an enable signal that is set by the Program Interface. The interface runs at 50 MHz. This clock signal is also sent to the DIFFS Host Interface, as a system clock for different DIFFS modules. Whenever there is data available at the Host interface, the *Host_empty* and *Host_alm_emtpy* signals are deasserted. The data is read from the interface by asserting the *Host_read* signal and is written to a FIFO in the HAPSBACK. A schematic representation can be found in Figure 19.

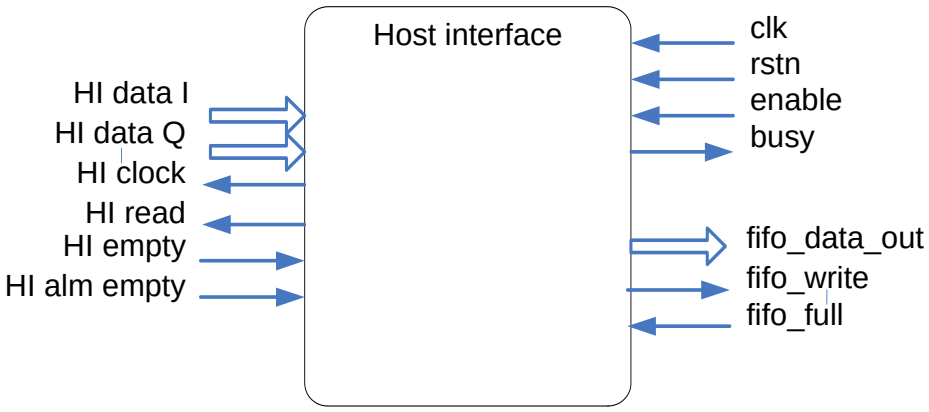

Figure 19: Schematic representation of the Host interface.

The AGC interface runs at 40 MHz. For now, the only function of the interface is to buffer the value which is on the *agc_gain* bus. The interface can be controlled through the HAPSBACK as it is a synchronous interface. A schematic representation can be found in Figure 20.

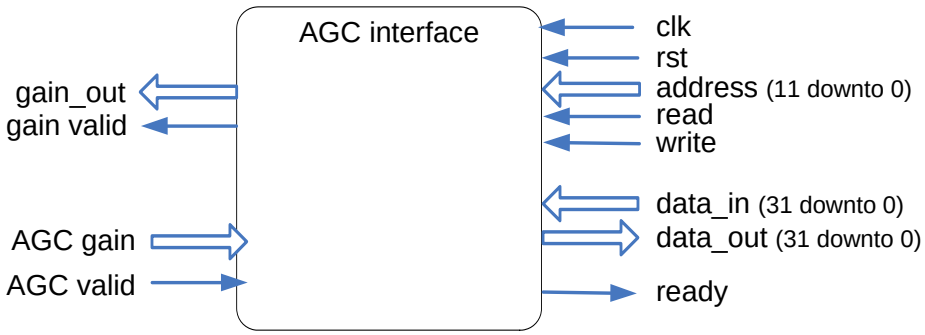

Figure 20: Schematic representation of the AGC interface.

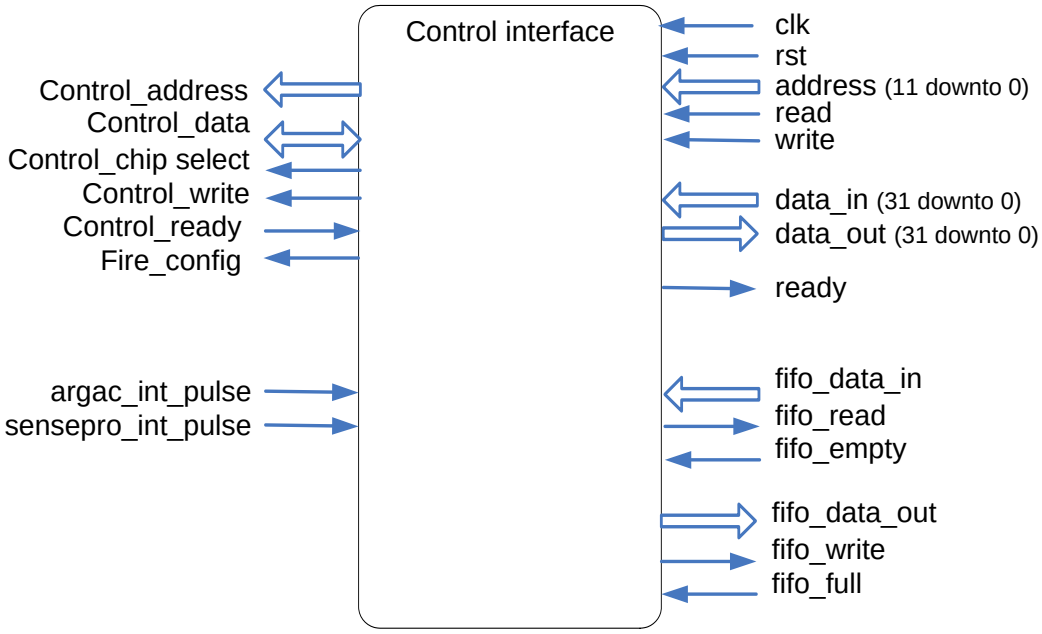

Figure 21: Schematic representation of the Control interface.

The Control Interface (Figure 21) is somewhat special because the control bus of the DIFFS is different to the other interfaces. The data bus is bidirectional and an address and an enable signal is used on the bus. The Control Interface does not have an enable signal; it has to be controlled by the master bridge via the synchronous interface. The Control Interface controls the control bus of the DIFFS. There are 4 possible operations: writing data to one or more registers, via single accesses or a burst write, and reading data from one or more registers, via single accesses or a burst read.

A single write can be performed by performing a write to the Control Interface at address 0x0zz, where 0xzz is the address of the register in the Control interface address map. The 16-bit data word is derived from the LSB's of the 32-bit data word from the synchronous interface. For

example, if we want to write 0x35a2 to address 0x82, the data word 0x000035a2 should be written to address 0x082 at the Control interface.

A single read is performed in a similar way. Obviously, we send a read command to the interface instead of a write command at address 0x0zz, where 0xzz is the address to read from. The resulting data word is read form the data bus and sent back via the synchronous interface.

A burst write will send a large chunk of data to the control bus. This data should be fed to the FIFO before the burst write is executed. The burst write will start at a given 8-bit address; thereafter, this address will automatically increase after every write. Such a burst write to the register at address 0xzz is started by a single write to the Control Interface at address 0x1zz. For example, if the burst write should start at address 0xab, the address should be 0x1ab. The data word is don't care in this situation. The Control Interface will then generate the correct sequence and load the data from FIFO.

A read burst is performed similarly to a write burst. Depending on the address, a burst of 16 (0x4zz), 64 (0x8zz) or 128 (0xczz) data words is read from the Control interface. The address to start the read form is always 0xzz. The data that is read out is sent to the FIFO.

The *fire_config* pin is used to trigger a certain configuration load operation into the DIFFS. Some modules of the DIFFS should be enabled at the same time; this is achieved by starting them when the *fire_config* pin is pulsed high. Pulsing the *fire_config* is done by writing don't care data to address 0x400.

Furthermore, there are two special signals going to the Control interface: *agrac_int_pulse* and *sensepro_int_pulse*. These signals are controlled by the Program interface (see below) and triggered by the interrupt pins. When one of these signals is pulsed, the control interface will start a specific write command. This is implemented here because handling the interrupts through the software is too slow to be able to respond fast enough.

Program interface: the last interface is now discussed: the Program Interface. This is a synchronous interface, meaning that it communicates with the master bridge in a synchronous way. The Program Interface is responsible for controlling various signals in the interface, on the DIFFS PCB or for basic interaction with the host. This control is done through bit-toggling: this means the Program Interface controls the addressed interface by simply toggling one or more bits. The interfaces controlled in this manner are the Host, Front End and Control interface.

The Program Interface controls the Host and Front End interface by setting an enable-bit. A write to the Program Interface at address 0x000 results in setting the 2 enable signals. The data word should be composed as follows: bit 1 and 3 are the enable masks, while bit 0 and 2 are the new enable values. When the enable mask bit is a logical '1' the new enable value will be introduced. Bit 0 and 1 control the Host-interface, bit 2 and 3 control the Front End-interface. The Program

interface also enables interrupt-handling by the Control-interface. A write to the Program Interface at address 0x100 results in setting 2 enable signals. The data word should be composed as follows: bit 1 and 3 are the enable masks, while bit 0 and 2 are the new enable values. If the enable mask bit is a logical '1', then the new enable value will be introduced. Bit 0 and 1 enable or disable AGRAC-interrupt-handling, bit 2 and 3 enable or disable SensePro-interrupt-handling. When AGRAC-interrupt-handling is enabled, the Program interface will send a pulse of 2 clock periods to the Control interface whenever the AGRAC interrupt pin goes high (thus, when a rising edge is detected), so that the Control interface can start its interrupt handling routine.

FPGA B FIFO-chain: The resources in FPGA B are used as a FIFO. The FPGA B receives the same clock as FPGA A, and the clock is handled the same way to ensure that all clocks in the design are phase matched. The FIFO has an input port, consisting of a write signal, a full signal and a 32-bit data bus. Every two written words are mapped into a single 48-bit word, which is then written to the actual FIFO on FPGA B. The mapping is done as shown in Figure 22.

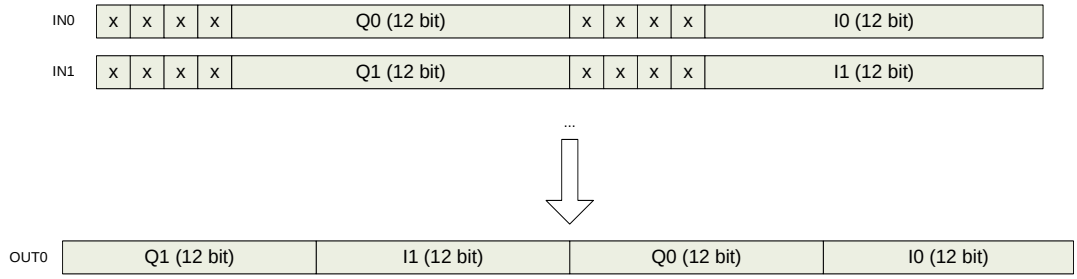

Figure 22: 32-bit to 48-bit mapping on FPGA B.

The output port of the FIFO structure consists of a read signal, an empty signal and a 48-bit data bus. The data words that are read, are sent to an asynchronous FIFO, from which the Front End interface can read. This FIFO should be asynchronous, since the FIFO-structure on FPGA B runs at 40 MHz, and the Front End interface can run at both 40 and 80 MHz.

FPGA utilization: The complete glue logic and interconnect is implemented on the HAPS-32 board. The FPGA utilisation is currently as follows (reported by the implementation tool).

Table 1: FPGA utilization for FPGA A.

Number of BUFGs	8 out of 32	25%
Number of BUFGCTRLs	4 out of 32	12%
Number of DCM_ADVs	1 out of 12	8%
Number of ILOGICs	157 out of 960	16%
Number of External IOBs	278 out of 960	28%
Number of OLOGICs	137 out of 960	14%
Number of RAMB16s	170 out of 336	50%
Number of Slices	9049 out of 89088	10%

Table 2: FPGA utilization for FPGA B.

Number of BUFGs	2 out of 32	6%
Number of BUFGCTRLs	0 out of 32	0%
Number of DCM_ADVs	1 out of 12	8%
Number of ILOGICs	25 out of 960	2%
Number of External IOBs	79 out of 960	8%
Number of OLOGICs	1 out of 960	1%
Number of RAMB16s	336 out of 336	100%
Number of Slices	22472 out of 89088	25%

As can be seen in Table 1 and Table 2, only the RAM-component usage is fairly high for both FGPA's. For FPGA B this is even 100%, which means we implemented indeed the largest possible FIFO on this FPGA. The rather high occupation on FPGA A is because we chose to foresee a large FIFO for FIFO port, to make sure all are large enough. When usage of the FIFOs becomes critical, there is off course some margin in this approach. We could for example carefully calculate what FIFO-size is needed for each FIFO-port, to make sure none is over dimensioned. If this method still does not provide enough RAM-components, the FIFO's should be downscaled, meaning that we will have to fill the FIFO's with multiple accesses while we are reading form the FIFO. This would off course result in some overhead on the control platform.

2.3.4 Component 3: control system on host PC

A standard PC is used to configure and interface with the system. It has a PCI card installed to be able to interface with the FPGA motherboard. The PCI card is connected to the HAPS-32 FPGA board using a high density flat ribbon cable. This way, communication between the host and the FPGA board is possible. The PCI card embeds the PLX PCI9656 chip to translate PCI signals into a simpler local bus format for easier interfacing on the FPGA.

The host PC serves 4 purposes:

- Interface the HAPS-32 board for configuring the FPGAs: The FPGAs on the HAPS board are configured using a dedicated program running on the host and a Xilinx USB II to JTAG cable connection. Generated bitfiles are downloaded through a JTAG interface onto the HAPS boards.
- System control: this part is discussed in section 2.3.4: System control consists of controlling the PCB, configuring the DIFFS and controlling the different data-streams in the system.
- System debugging: As the host PC is used for interfacing with the board, it is also used for debugging. The complete software tool chain is run natively on the host. This means that software can be debugged with the standard GNU/Linux debugging tools.Next to that,

using the Xilinx USB II to JTAG cable, it is possible to debug the implemented hardware as well. With specialized tools (eg. Synplicity identify) signal tracing modules can be installed in the VHDL sources. After synthesis and implementation, logged signals can be downloaded for visualization on the host.

- Demonstration: As the host PC has the capabilities of a normal computer, it can be used for demonstration purposes. Software for displaying images, videos, graphs or graphical user interfaces can be run on the system to demonstrate and support the hardware functionalities.

The control system is designed as a bridge between intuitive control of the prototype and the DIFFS prototype hardware. In an early stage, it is used for testing basic functionality of the FPGA-interface. Next to that it can be used for testing the DIFFS PCB and basic connectivity with the DIFFS. In a later phase, the control system can be used for extensive testing of the DIFFS. When all tests are completed, the system can be used to control any kind of demonstration that is running on the platform.

The only possibility to control the prototype is through the use of scripts. A script is made with all commands subsequently entered. Running the script will then execute all these commands. This mode of control requires a rather high degree of insight in the commands however, but is extremely useful when some commands have to be executed several times, for example in a loop.

Implementation: If we take a closer look to Figure 14, we can see that the host PC is connected to the FPGA-interface through the PCI-bus. The so-called PCIX-card is shown in Figure 23.

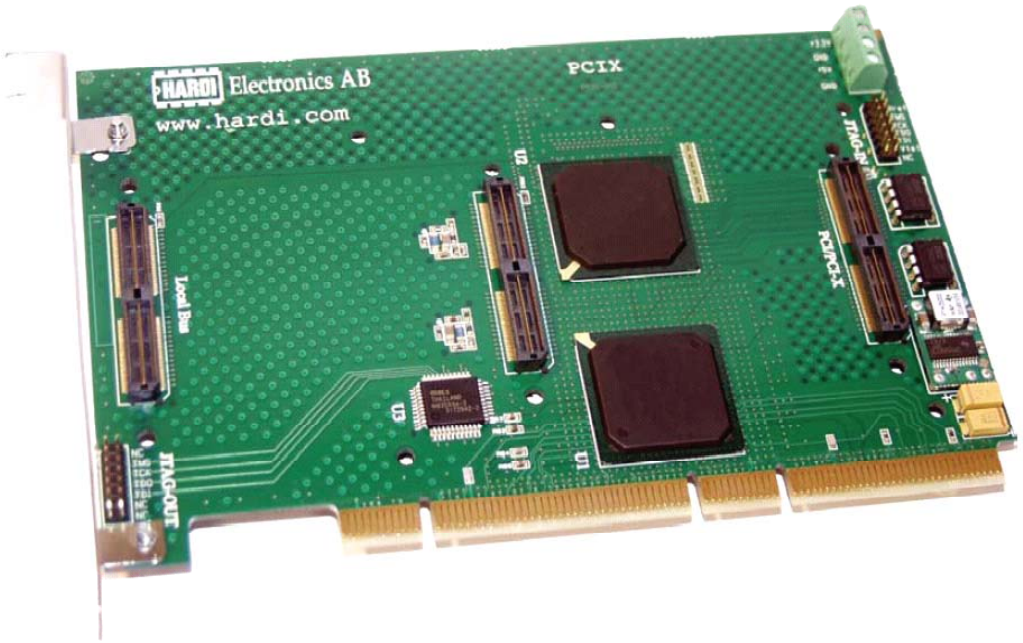

Figure 23: The PCIX-card that connects the host PC and the HAPS-32 board.

When we are booting the kernel, the installed PLX-driver module is automatically loaded. This driver is the lowest level of the software stack, as can be seen in Figure 24. A PLX-driver-SDK is available from the PLX website [4]. This SDK contains the kernel module and a simple API. To be able to use this PLX-driver, the PlxApi.a library must be built.

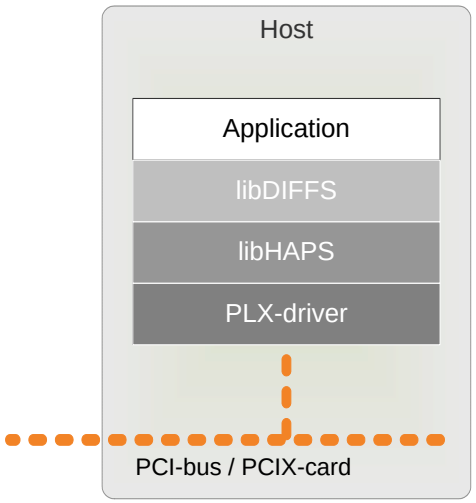

Figure 24: The host PC software stack.

On top of this PLX-driver, a C-based API for HAPSBACK is running. This is a basic, lightweight library for easy access to the HAPSBACK functionality. It hides the lower-level PLX-structures from the user. Only one HAPSBACK is supported and no interrupts are possible. Flow control is done by blocking functions such as DMA.

The whole software control system is based on only a few, basic functions that are included in the libhaps.a library. To be able to use these functions, the library must be build first. There are two types of functions: those for controlling the PLX-device and those for accessing the HAPSBACK. An overview is given in Figure 25. The first three functions control the PLX-device. The others are accessing the HAPSBACK. The basic functions here are the functions *hapsback_read_dma* and *hapsback_write_dma*. Everything else is derived from these functions and offer easy access to the HAPSBACK. The specific DIFFS control software is built on top of this library and is called libdiffs. This library is explained into detail in the next section.

```
int hapsback_init (void);
int hapsback_open (void);
int hapsback_close (void);

unsigned int hapsback_read_single_reg (unsigned int address);
unsigned int hapsback_write_single_reg (unsigned int address, ..);
unsigned int hapsback_read_dma (unsigned int address, ..);
unsigned int hapsback_write_dma (unsigned int address, ..);
unsigned int hapsback_set_fifo_write_mux (unsigned int dst_fifo_id);
unsigned int hapsback_set_fifo_read_mux (unsigned int src_fifo_id);
unsigned int hapsback_get_fifo_write_mux (void);
unsigned int hapsback_get_fifo_read_mux (void);

unsigned int hapsback_read_version (void);
unsigned int hapsback_run_test (void);

.. = parameters removed for readability
```

Figure 25: Overview of the libhaps.a library API functions.

Software: The libdiffs.a library consists of 2 files:

- diffs_haps
- diffs_supp

These can possibly be extended with a file diffs_api if needed. In this library some API's could be introduced, to make the programming and control of the DIFFS more high-level for the user. For now, all functionality is stored in the diffs_supp library.

The library diffs_haps contains a number of basic, low level functions. An overview is given in Figure 26. All these functions access the HAPSBACK. They can be used for testing the FPGA-interface or some other basic testing of the platform.

```
void test_read (void);
void test_write (void);
void test_read_ctrl (void);
void test_write_ctrl (void);
void test_read_loop (uint32_t * data);
void test_write_loop (uint32_t * data);
```

Figure 26: Overview of the functions of the diffs_haps library.

The library diffs_supp defines a number of functions that can be used to interface with the DIFFS. An overview is given in Figure 27.

All these functions are built using the libhaps.a library, or possibly using other functions from the diffs_supp library. Their functionality is most of the time self-explanatory and can be compared to the discussion on the DIFFS interfaces. When the passed argument is a pointer to a file, this file will be used for either dumping the data or fetching the data from. When dumping data to a file, data will be written in 32-bit hex-mode. For example, an entry could be 0x000000ff. When reading from a file, the data in the file is expected to be in 8-bit hex mode. A possible entry could be 0xa2. For the decoder, an extra function is defined that is capable of reading 32-bit hex-data from a file. A possible entry could be 0xdeadbeef. More information about the use of the functions, their arguments and their return values can be found in the source documentation.

```
void ctrl_write_reg (uint8_t address, uint16_t data);
uint32_t ctrl_read_reg (uint8_t address);
void ctrl_start_write_burst (uint8_t address);
void ctrl_start_read_burst (uint8_t read_address, uint8_t count);
void fill_fifo_from_buffer (uint8_t dest, uint32_t * pData, uint16_t count);
uint32_t fill_fifo_from_file (uint8_t dest, FILE * pFile, uint8_t format);
void empty_fifo_to_buffer (uint8_t src, uint32_t * pData, uint16_t count);
void empty_fifo_to_file (uint8_t src, FILE * pFile, uint16_t count);
void set_diffs_reset (boolean on);
void fire_configuration (void);
void set_host_if_enable (boolean on);
void set_front_end_if_enable (boolean on);
void set_all_if_enable (boolean on);
uint8_t read_agc_gain (void);
uint8_t read_intr_status (void);
void set_agrac_intr_handling_enable (boolean on);
void set_fast_FE_clock (boolean on);
```

Figure 27: Overview of the functions of the diffs_supp library.

Next to these library-files, there are 2 independent header files used in the libdiffs library, namely diffs_types.h and diffs_memmap.h. The former contains all types that are used in the library; there are all basic types. More info can be found in the source file. The latter contains the definitions of all addresses used in the DIFFS-interface, so that a certain command or register can easily be accessed. An extraction can be found in Figure 28.

```
#define Ctrl_IF_power_reg 0x01
#define Ctrl_IF_reset_reg 0x02
#define Ctrl_IF_clkstat_reg 0x03
#define Ctrl_IF_diffs_ctrl_reg 0x10
#define Ctrl_IF_CO_IQ_I_reg 0x18
#define Ctrl_IF_CO_IQ_Q_reg 0x19
#define Ctrl_IF_DS_cfg_reg 0x20
#define Ctrl_IF_DS_stat_reg 0x21
#define Ctrl_IF_HI_src_sel_reg 0x30
#define Ctrl_IF_HI_offset_reg 0x31

#define NOC_cmd 0x70
#define NOC_data 0x71
#define NOC_adr 0x72
#define NOC_stat 0x73
#define NOC_ram 0x78

#define HAPS_JTAG_IF_BASE_ADDR 0x80101000
#define HAPS_CTRL_IF_BASE_ADDR 0x80102000
#define HAPS_AGC_IF_BASE_ADDR 0x80103000
#define HAPS_PROG_IF_BASE_ADDR 0x80104000

#define CTRL_write_base_addr (HAPS_CTRL_IF_BASE_ADDR+0x000)
#define CTRL_write_burst_base_addr (HAPS_CTRL_IF_BASE_ADDR+0x004)
#define CTRL_fire_cfg (HAPS_CTRL_IF_BASE_ADDR+0x008)
#define CTRL_read_base_addr (HAPS_CTRL_IF_BASE_ADDR+0x000)
#define CTRL_read_burst_16 (HAPS_CTRL_IF_BASE_ADDR+0x004)

#define PROG_interfaces_en_addr (HAPS_PROG_IF_BASE_ADDR+0x000)
#define PROG_intr_pulse_en_addr (HAPS_PROG_IF_BASE_ADDR+0x100)
#define PROG_diffs_rst_addr (HAPS_PROG_IF_BASE_ADDR+0x200)
#define PROG_frontend_clk_sel_addr (HAPS_PROG_IF_BASE_ADDR+0x600)
#define PROG_rd_intr_stat (HAPS_PROG_IF_BASE_ADDR+0x900)
```

Figure 28: An extraction of the diffs memory map.

As mentioned before, all these functions are developed to be used in a script. These scripts can be of any kind, and here the user is free to develop any script wanted. In the source code, all used scripts can be found, going from *basic_test.c* and *vmem_test.c* to *wlan_sync_test.c* and *dvb_sense_test.c*. An example of how a script could look is shown by the pseudo-code in Figure 29.

```
int main (int argc, char *argv[])
{
  // set global variables
  // declare identifiers
  // initiate platform and DIFFS
  hapsback_open();
  set_diffs_reset(false);
  sleep (1);
  // configure the DIFFS for the wanted mode
  ctrl_write_reg(address, data);
  ...
  // write the firmware to the AGRAC and the SensePro
  fill_fifo_from_file(1, pFile, 1);
  ...
  // write the data to the FIFO and transfer to the DIFFS
  fill_fifo_from_file(2, pFile, 2);
  // check the results
  result = ctrl_read_reg(address);
  if (result = ok) { printf(...); }
  // terminate DIFFS and platform
  set_diffs_reset(true);
  hapsback_close();
  return 0;
}
```

Figure 29: Example of a simplified script in pseudo-code.

2.3.5 Power measurements

The firmware running on the AGRAC and SensePro processors determines the functionality of DIFFS, thereby making DIFFS a fully programmable core. Code has been developed to support the following modes which are relevant for CR operation:

- WLAN synchronization, based on auto-correlation;
- LTE synchronization, based on cross-correlation;
- DVB-T sensing, based on cyclostationary detection;
- LTE sensing, with FFT and carrier leakage removal.

The code was first designed in Matlab, then quantized and finally mapped on the hardware. All modes have been verified on the chip in the lab and will be demonstrated over the air in combination with the Scaldio reconfigurable analog frontend [19], [20] (see section 2.4). Some modes are a lot more demanding in terms of processing power and the challenge is to ensure a power efficient implementation for each mode individually. The leakage is 1 mW and the dynamic power consumption for all mapped modes is presented in Table 3.

TABLE 3: MEASURED POWER CONSUMPTION.

Mode	Power (mW)	Clock (MHz)	Reference power (mW)
WLAN wait	2	40	5 [23]
WLAN sync	4	40	76 **[25]**
LTE sync	22	80	-
LTE sense	20	40	-
DVB-T sense	7	40	-

To the best of our knowledge this is the first flexible architecture that targets important CR features for multiple standards in a single architecture. A comparison to state of the art solutions shows that existing chips do not meet the power/cost constraints for integration in handheld devices [21], [22] or have no specific sensing capabilities [23], [24].

2.4 Phase 3: Integrated Sensing Engine Prototype

Since the analog front-end and digital front-end prototypes prove to be operational regarding the standalone functionality, it is possible to combine them into integrated prototypes. This way, additional functionality can be investigated, such as the control of an analog RF receiver and actual spectrum sensing on real-life signals. Two steps are defined:

- Lab version: this prototype is intended for lab use, initial debugging and code development and therefore offers lots of debugging facilities.
- Portable version: this prototype is intended for easy demonstration of DIFFS in a stand-alone mode without laboratory equipment.

2.4.1 Lab version

The SCALDIO and DIFFS standalone test PCB's (cf. sections 2.2 and 2.3) are connected through the HAPS-32 FPGA prototyping board (Cf. section 2.3.3). Again, the FPGA's will be used as interconnect between the DIFFS, the SCALDIO and the host PC, similarly to the design described in section 2.3.

The system is shown schematically in Figure 30 and physically in Figure 31. Two additional PCB's are needed for this prototype, but are not shown in the schematic. These are a clock PCB to generate the sampling and reference clock for the SCALDIO module, and two power PCB's to generate and distribute the required supply voltages for the SCALDIO IC.

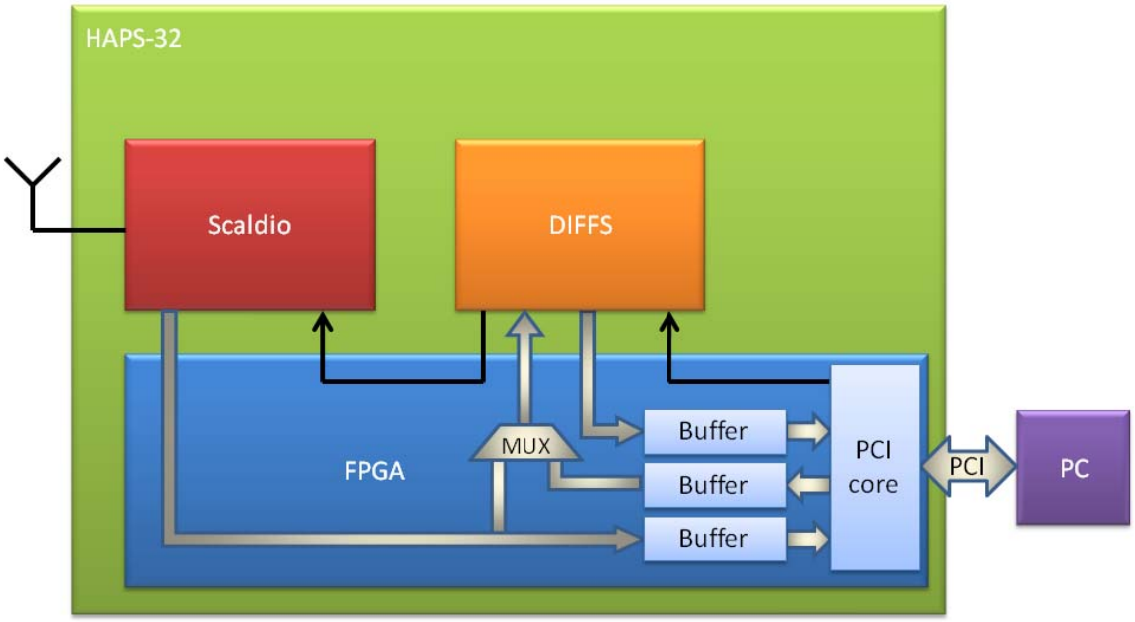

Figure 30: SCALDIO & DIFFS on HAPS-32 board.

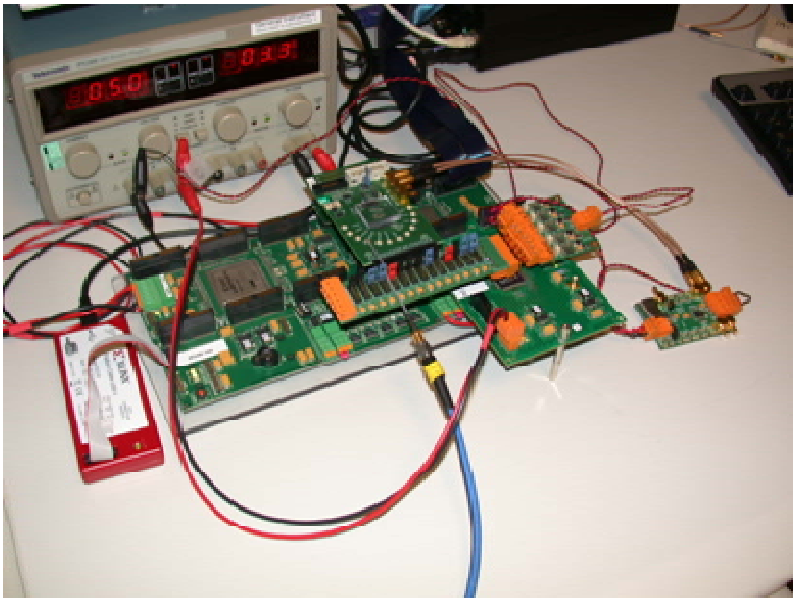

Figure 31: SCALDIO & DIFFS on HAPS-32 board.

The analog front-end is connected to the antenna via a band selection filter to avoid interference from large out-of-band signals. The FPGA on the digital board has two functions. First it acts as glue logic to connect the analog and the digital sensing solution, receiving digital IQ data from and sending control signals to the analog front-end. The second task of the FPGA is providing the interface towards the outside world. For this interface a system of control busses and FIFO's is implemented on the FPGA, connecting to the host PC via a PCI-card. The software running on the host PC starts up the prototype by downloading the firmware into the digital sensing engine. Once this is finished the digital sensing engine will configure the analog front-end to operate at the correct RF frequency, channel bandwidth and gain setting. The digital IQ samples are then processed by the digital sensing engine. Depending on the downloaded firmware different detection algorithms can be evaluated.

The modifications to the setup that are needed in comparison with the approach presented in section 2.3 to integrate the Sensing Engine are the following.

FPGA Design modifications: The modifications to the FPGA-design needed for the integration are not that big. Basically, the FPGA-designs for the standalone setups have to be merged into one design. The design which is used for the standalone DIFFS setup is extended with two components, the RX-block for receiving ADC-samples from the SCALDIO, and a Phase Shift-block to fine tune the ADC-clock used on the platform. The RX-block will demultiplex I- and Q-samples into two different data streams and transform these streams to the correct 12-bit data format needed for the DIFFS. The Phase Shift block allows shifting the phase of the incoming ADC-clock so that the RX-block can sample both I- and Q-values at the correct time. Next to the addition of these blocks, some minor modifications are needed in terms of data routing: now that real SCALDIO samples are used to feed the DIFFS, the Front End interface will use the I- and Q-samples coming from the RX-block instead of generated samples that are fed in through a FIFO. Furthermore, 4 NOC-signals are 'created': a NOC-reset signal is introduced in the Program Interface, and three AGC-signals are used as NOC-TE, NOC-clk and NOC-DI. This last modification allows the SCALDIO to be programmed by DIFFS, who has a NOC-controller on-board.

Software adaptations: the SCALDIO will be programmed by the NOC-controller inside the DIFFS. However, since every SCALDIO has to be programmed in its own specific way, there is no real intelligence in the NOC-controller: the controller will only translate commands from the host into the specific NOC-protocol for SCALDIO configuration. This means that all NOC-commands that were used in the past have to be replaced by control-commands. To implement this in a generic way, files were created for every SCALDIO and for every configuration, so that the software only has to load the correct file and execute the commands that are specified.

The other change to the software could be considered as a more cosmetic procedure. The DIFFS software API that is used for the standalone setup, is extended with more generic functions. These functions will group a number of 'internal' functions to perform a specific operation on DIFFS. This way, the DIFFS API becomes less transparent and more high-level, which will enable control of the DIFFS without the need for a thorough knowledge of the DIFFS protocols etc. Next to the extension of the DIFFS API, a software library was created around it to operate at the level of the

sensing engine. This way, the control of the whole Sensing Engine is enabled, again without the need for a thorough knowledge of the underlying mechanisms of DIFFS, SCALDIO, the FPGA-design or the PCI-interface. An overview of the resulting library can be found in Figure 32. As can be seen, three specific (visible) types are created: a handler *se_t* which contains all info on the active Sensing Engine, a variable *se_mode_t* which indicates in which operating mode the Sensing Engine is configured, and the struct *se_config_s*, which indicates the configuration of the Sensing Engine. Furthermore, there is a set of functions available to open, initialize and close a Sensing Engine, to configure it, to do measurements, etc. The user of the Sensing Engine thus can create his own program using just these functions. More documentation on data formats etc is off course available.

```
typedef struct se_s *se_t;

typedef enum {
        FFT = 0,
        DVB_CYCLOSTAT = 1
} se_mode_t;

struct se_config_s {
        se_mode_t se_mode;
        uint32_t fe_bandwidth;
        int16_t fe_gain;
        uint32_t start_freq;
        uint32_t stop_freq;
        uint16_t fft_points;
        uint16_t dvb_nr_carriers;
};

se_t se_open(void);
int se_init(se_t se_h, struct se_config_s *se_config);
void se_close(se_t se_h);
int se_configure(se_t se_h, struct se_config_s se_config, uint16_t mode);
int se_check_config(se_t se_h, struct se_config_s se_config);
int se_start_measurement(se_t se_h);
int se_stop_measurement(se_t se_h);
int se_location(se_t se_h, float *current_loc);
int se_get_result(se_t se_h, float *destination);
```

Figure 32: Top level software library for the sensing engine.

2.4.2 Portable version

The second, more integrated, version of the prototype will connect the two IC's via a dedicated FPGA which will also handle the connection to a USB controller. The RF module is exactly the same as described above. However, a new PCB has been designed to host the DIFFS chip, the FPGA and the USB controller. The flow of the data on this prototype is fairly similar as on the prototype described above, without the debug facilities. An overview is shown in Figure 33.

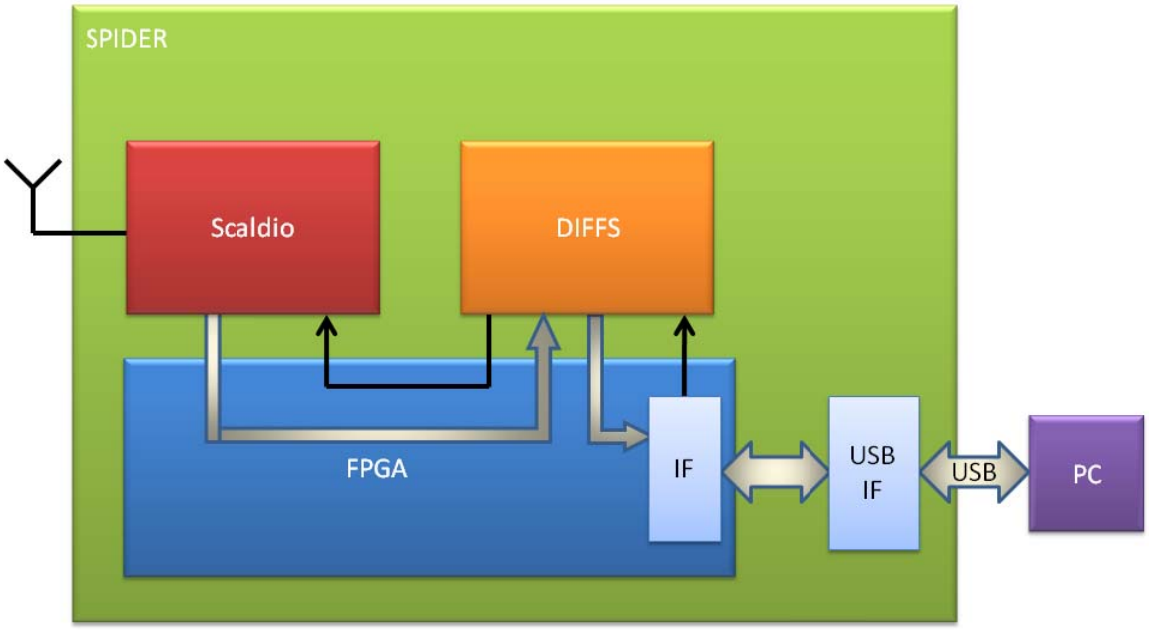

Figure 33: SCALDIO & DIFFS on stand-alone PCB.

The purpose of this prototype is to build a portable setup. The HAPS-32 board is ideal for initial testing and debugging and for building prototypes in a lab-environment, but the downside is the lack of portability and the need for a PC with PCI-card. The PCB that has been designed, called SPIDER (Sensing Platform for Integration and Demonstration of the DIFFS), introduces a more compact, more lightweight prototype, as shown in Figure 34. The advantages are that it is smaller, more portable and robust. It has a USB-interface, so that it can be controlled from a standard laptop. A schematic representation is shown in Figure 35. The SPIDER PCB coupled to the SCALDIO RF module is shown in Figure 36.

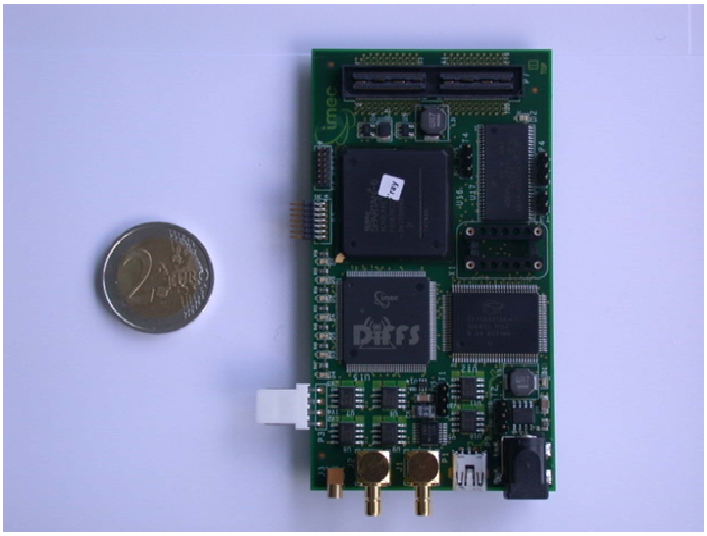

Figure 34: Spider Board.

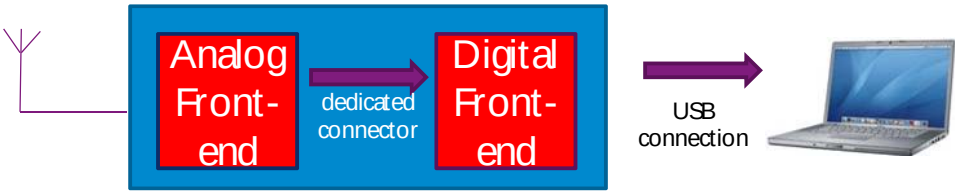

Figure 35: Schematic representation of the Sensing Engine.

Figure 36: Integrated Sensing Engine: SPIDER PCB coupled to SCALDIO RF module.

The portable version of the Sensing Engine is explained in more detail.

SPIDER PCB: The following components are included on the SPIDER PCB, which is depicted in Figure 34:

- the DIFFS chip;
- a Samtec connector to plug on an RF module;
- an Spartan-6 FPGA;
- a USB-interface;
- a memory (SRAM) component to allow intermediate data-storage;
- power modules, to generate the required supply voltages for both the SPIDER PCB as the RF module;
- clock generation, routing and buffering.

The PCB-design was optimized to achieve the following goals:

- No more need for the bulky HAPS-board, nor a PC with a PCI card in it. This functionality has been replaced by the on-board FPGA (which is smaller, but still more than big enough to fulfill all needs), the Samtec-connector and the USB-interface. The level of the signals going to the Samtec-connector is selectable (2.5. V or 3.3 V), depending on the RF module.
- No more need for the SCALDIO power and clocking PCB's. This functionality has been replaced by several voltage regulators to create the needed voltages. Furthermore, there is an oscillator-socket from which the desired frequency can be generated; these clock signals can then be routed to several jacks, of which the level is selectable (2.5. V or 3.3 V), depending on the RF module.
- No more need for the DIFFS PCB: the DIFFS has been mounted directly on the SPIDER PCB.

The programming and usage of the SPIDER PCB is fairly simple, in fact it is almost plug-and-play. Once the SPIDER is connected to a PC or laptop, it can run autonomously: the PCB is bus-powered, which means all power comes from the USB-port. A simple script has to be run to program the firmware of the USB-microcontroller and the bitfile of the FPGA. Once this is done, the user can run its own software executables. How this is implemented is explained in the next subsection.

USB-interface on SPIDER: The USB-microcontroller that is used is Microchip's FX2. This controller determines to a large amount how the USB-interface is implemented off course. The controller is configured to have 4 endpoints: one for data in (from the PC to the microcontroller), one for data out (from the microcontroller to the PC), one for control in, and one for control out. "Control in" packets must be 2 times 16 bit. Data packets can be as long as they want, as long as they are sent in multiples of bytes. These data packets will be sent in chunks of 512 bytes. Using this configuration, we can create a USB-interface on the FPGA which is similar to the PLX-interface on the HAPS. Also in this case, the USB-interface will connect to a bridge which will then act as a

multiplexer to forward the data to the correct block on the FPGA-design. Following the USB-standard, a 16-bit data width has been selected for both commands or addresses and data. This means a control message will consist of a command and a data word. All FIFO's on the FPGA will be 16 bit wide. This has its impact on all interfaces off course, but this impact is limited to representational issues.

Similar to the HAPSBACK interface, a low-level software API is generated which handles the USB-communication at the lowest level, which is called SPIDERback. When initializing, this API will search for USB-devices with the correct firmware and bitfile programmed. If it has found a SPIDER device, it will check if it is not yet being used and if not, it will claim the interface so that it can be used in the user's software program. The way this API is implemented makes it possible to connect multiple SPIDERs to a single PC, so that multiple Sensing Engines can be used at once.

Software adaptations: Because the new low-level API SPIDERback has been implemented, all software layers building on the previous version cannot be used anymore. Therefore, each interface has its own wrapper, which will translate all generic function calls to the specific function calls for the interface that is being used, so that both interfaces can exist together. This implicates that all interface-specific function calls from every software layer has to be replaced by its generic equivalent. For example, everywhere where *hapsback_write_single_reg* has been used, this function call will be replaced by *platform_write_single_reg*. If a setup with a HAPS-board is being used, this function call will be translated into *hapsback_write_single_reg*; if a SPIDER is being used, it will become *spiderback_write_single_reg*. This is done for all low-level function calls. Next to this, the data format has to be taken care of. Since the HAPS is using 32-bit wide words and SPIDER is using 16-bit wide words, casting of arguments in the wrapper files is needed to provide and transfer all data and commands in the right format. Last but not least, the memory mapping is different for both systems. This implicates that specific platform-dependant memory-maps exists, so that can be linked to the correct map when building the software application.

2.4.3 Examples of prototyping results

2.4.3.1 Sensing performance evaluation

The performance of a subset of the algorithms that are mapped onto the digital sensing engine is investigated on the setup. Two algorithms have been verified:

- Cyclostationary detection for detection of DVB-T signals, and
- Resource block allocation detection for LTE signals.

The sensing performance is analyzed in more details in Section 3

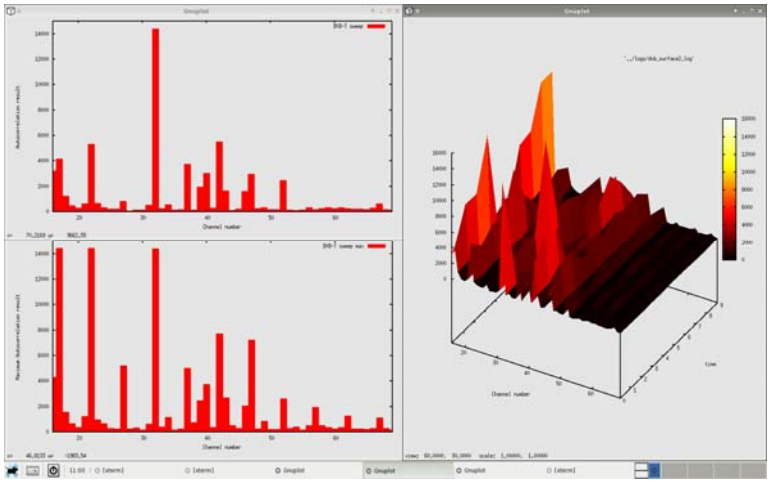

Figure 37 : Prototyping DVB-T Sensing.

2.4.3.2 Wideband spectrum sweeping

By coupling the digital frontend to our wide-band reconfigurable analog frontend [3], a wideband spectrum scanning prototype has been realized. For the first experiments we target fast spectrum scanning. In this scenario the digital frontend is not used for data reception, but solely to quickly scan the frequency spectrum from 500 MHz to 2.5 GHz using a single analog frontend. The spectrum is divided in subbands of 20 MHz. Every band is received consecutively by reprogramming the analog frontend. On every subband a 128-point FFT is performed. By integrating both components in a single platform, it is possible to scan the complete 2 GHz wide band in less than 10 ms with a resolution of ~150 kHz. To our best knowledge this is the highest speed, highest accuracy integrated solution at such a low power. An example output plot is shown in Fig. 38. The activity on the 900 MHz GSM band and the 2.4 GHz ISM band are clearly visible in the spectrum.

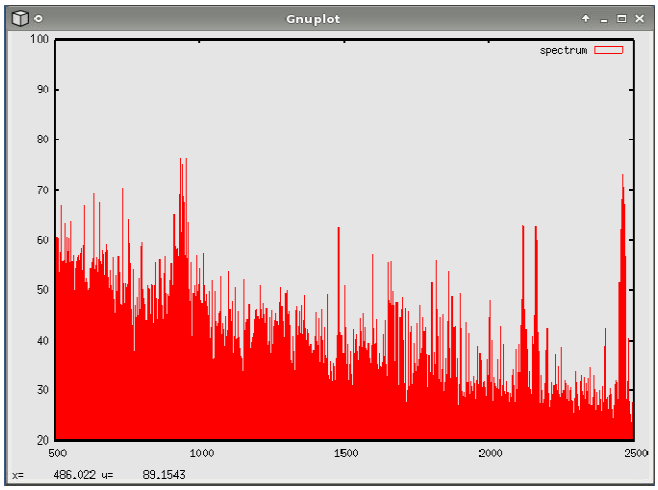

Figure 38: Spectrum sweep screenshot from the prototype platform.

2.4.3.3 Coupling with FARAMIR REM prototype

The custom MCD prototype component comprises two parts, i.e. a sensing engine and software running on a Cappuccino PC (Figure 39). An FPGA connects the different blocks in the sensing engine. On one side, this FPGA is connected to the analog RF board (which, depending on the configuration, can be a WARP radio or a SCALDIO board) whereas on the other to a USB. A pin header connected to the FPGA enables the extension with a GPS module for geo-location purposes described in the requirements above.

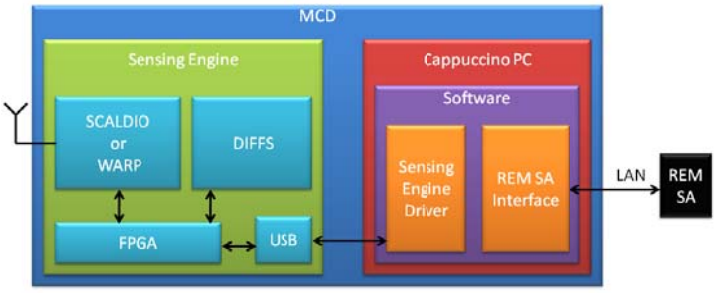

Figure 39: FARAMIR sensing engine integration in REM prototype.

The sensing engine is connected to a small form-factor PC via a USB interface. The PC runs software to implement the translation between the driver for the sensing engine and the interface towards the REM SA. The sensing engine driver provides all the necessary functions to program the sensing engine whilst hiding all the low-level details for the user. The driver provides a limited amount of functions to control the RF frequency, the channel bandwidth and the sensing algorithm of the sensing engine and acquires the result of the sensing operation. The result of the sensing operation depends on the requested measurement and can range from the raw time-domain samples over FFT output vectors to a single power value for a specific frequency band. Further details are available in Deliverable D6.1.

3 Performance Evaluation

3.1 FARAMIR sensing engine

3.1.1 Reference sensing functionality

The performance of a subset of the algorithms that are mapped onto the digital sensing engine is investigated on a setup using the RFIC in combination with Matlab. Two algorithms are verified:

- Cyclostationary detection for detection of DVB-T signals, and
- Resource block allocation detection for LTE signals

The functionality is described in more detail in the following sections.

3.1.1.1 Cyclostationary detection for DVB-T

DVB-T is modulated according to the OFDM technique, using 2048 or 8192 carriers (2k, 8k mode, respectively). Every OFDM block is extended, copying in front of it its own end (cyclic prefix). The width of the cyclic prefix can be 1/32, 1/16, 1/8, or 1/4 of the original block length. This copy can be used as a feature for sensing.

OFDM schemes rely on a cyclic prefix (CP) to combat inter-symbol interference. It is possible to exploit that feature or information when designing sensing algorithms. Indeed, since a CP is an exact copy of the part of the OFDM symbol, coherent detection becomes possible. The CP correlation function can be defined as [5]:

$$r_{cp}[n] = \frac{1}{SL}\sum_{u=0}^{S-1}\sum_{m=0}^{L-1} x[n+m+N+uM]\,x^*[n+m+uM],$$

where *S* is the number of OFDM symbols accumulated for CP correlation, and *x[n]* is the received signal or sample. *L* is the CP length and *N* is the length of an OFDM symbol. *M* is the total number of samples considered per OFDM symbol, i.e., the OFDM symbol length and the CP length: M=N+L. Note that $r_{cp}[n]$ is real when x[n+m+N+uM]=x[n+m+uM].

Since symbol timing information is lacking, it is not possible to compute $r_{cp}[n]$. The only possibility is to compute the value multiple times (M times to be exact) and the autocorrelation value would be the maximum. As a result, the decision statistic for the CP method could be:

$$t_{cp} = \max_{0\le n\le M-1} real(r_{cp}[n]).$$

The drawback of this approach is however that applying the max operation over a set of M noise values has a high probability of giving a relatively large number for the decision statistic, even in case when there is only noise and the correlation is expected to be zero. In fact, in the low SNR regime, the noise variance is very high so the probability that the max over M noise samples is higher than the max of the true autocorrelation function that is also subject to noise.

A better test statistic could then be the average over all the M autocorrelation values. The drawback is that the correlation of the L CP samples is smoothed out over the M samples. L/M can

be in the range 1/4 to 1/32. The advantage is however that the max operation is avoided. The test statistic taken in our measurement is hence:

$$t_{cp} = \sum_{n=1}^{M} real(r_{cp}[n]).$$

Note that we can focus on the real part since in our measurement setup there is no Carrier Frequency Offset (CFO) present since the reference frequency of the chip has been locked to the reference frequency of the test and measurement equipment. Sensing decisions are then taken by comparing the test statistic against a pre-defined threshold that is typically a function of the estimated noise variance σ_o^2. The false alarm probability is only a function of the distribution of the test statistic under hypothesis H_o and the threshold γ. As a result, specifying a target false alarm rate allows determining γ if the distribution of H_o is known. In this measurement, we determine the ROC by varying γ. As a result, it is not needed to compute the noise variance and distribution.

We compare the feature-based sensing schemes with the baseline energy detector test statistic that is defined as

$$t_{ed} = \frac{1}{M} \sum_{i=0}^{M} (x[i])^2,$$

with M the total number of input samples considered.

3.1.1.2 LTE Resource block allocation detection

The LTE sensing functionality aims at determining the LTE OFDMA resource allocation of an LTE channel. In line with the possible channelization for LTE, this scenario focuses on sensing the LTE OFDMA resource allocations in a band of 3, 5, 10 or 20 MHz [6]. Since the LTE OFDMA resource use is very flexible, a way of performing multi-band energy detection is needed to allow determining the exact resource use in such context. For both FDD and TDD, several RF bands have been assigned for LTE. A sensing solution for LTE should allow enough flexibility in terms of bandwidth and carrier re-configurability to allow sensing in each of these bands and channels. For each of the operating bands, a set of different transmission bandwidths has been defined. Each of the bandwidths is subdivided into resource elements which are the smallest resource allocation primitive for LTE. A physical resource block (PRB) is defined as a block of 180 kHz of frequency during 0.5 ms. In terms of OFDM modulation, a resource element corresponds to 12 subcarriers during 7 or 6 OFDM symbol blocks. When detailed information is required about the resource usage of a LTE network, it is required to sense at the granularity of such resource element.

Table 4 shows the number of resource elements for the different LTE bands. Given the large maximum number of possible resources (100 in case of a 20 MHz bandwidth), in order to reduce latency, the sensing of resource elements will be performed in parallel. A windowed FFT based approach has been taken for sensing of resource elements in LTE. No standard sensing requirements have been defined for LTE sensing. Following the requirements on LTE synchronization, a probability of detection P_D of 90% and a probability of false alarm P_{FA} of 10% over the LTE SNR operating conditions has been taken as the target performance requirement for

the sensing algorithms. In the next section, the chosen algorithms for multi-band sensing are discussed.

Table 4: Number of resource blocks.

Transmission Bandwidth (MHz)	3	5	10	20
Number of resource blocks	15	25	50	100
Sampling rate for sensing (MHz)	2.88	5.76	11.52	23.04
FFT size for multiband sensing	16	32	64	128

A multi-band windowed FFT based sensing approach is taken for sensing the resource usage in the flexible OFDMA Downlink LTE system. Windowed FFT analysis suffers from frequency leakage due to the finite time observation of the FFT window [7]. When energy detection is performed directly at the output of the FFT bins, the energy leakage creates a large number of false alarms. In this section a procedure to estimate and remove the leakage is described. The performance gain of the proposed method compared to a non energy detector without leakage removal for a 10 MHz bandwidth with half resource usage is shown in xx. In a windowed FFT analysis the signal before Fourier transformation is expressed as

$$r_m[n] = w[n]x_m[n], \quad (1)$$

where w is a vector containing the window coefficients and x_m is the m vector containing the signal before windowing. By applying the properties of the Fourier transform we have

$$\tilde{r}_m = \tilde{w} \otimes \tilde{x}_m, \quad (2)$$

where $\tilde{r}_m$, $\tilde{w}$ and $\tilde{x}_m$ represent the Fourier transforms of the signal after windowing, the window vector and the original signal respectively and $\otimes$ represents the circulant convolution. From $\tilde{r}_m$, we may get the power spectral density of the received signal as

$$\hat{p} = \frac{1}{M}\sum_{m=1}^{M}\left|\tilde{r}_m\right|^2, \quad (3)$$

with M the number of averaged frequency domain vector samples. The power spectral density estimation on a certain FFT bin i, $\hat{p}[i]$ will be composed by the power of the original signal $p[i]$ weighted by the window gain plus the carrier leakage caused by the windowing processing. According to that, we may express $\hat{p}[i]$ as

$$\hat{p}[i] = l[i,i]p[i] + \sum_{j \neq i} l[j,i]p[j] \quad (4)$$

where l is a matrix containing the carrier leakage coefficients linked to the window w for the different bins as defined in (1). In order to avoid detecting power only created by leakage, the leakage must be estimated and removed from the original power spectral density estimation $\hat{p}[i]$.

This procedure will be performed in an iterative way starting from the strongest bin. The bin leakage removal may be summarized as follows:

1. Selection of $\hat{p}_{\max}$, the strongest bin above threshold.

$$\hat{p}_{\max} = \max_{i \in I, \hat{p}[i] > k[i]} \{\hat{p}[i]\} \quad (5)$$

   where k is a vector containing the thresholds for each bin.
2. Leakage removal. The leakage corresponding to carrier index i, $l[i]$ is removed the power spectral density function estimation

$$\hat{p}[j] = \hat{p}[j] - (\hat{p}_{\max}[i] - \sigma^2_{\tilde{r}}[i]) l[j,i] \quad (6)$$

   with $j \neq i$ and $\sigma^2_{\tilde{r}}$ a vector containing the Gaussian noise variance on each bin.
3. Stop condition. Remove index i from the set I. Stop if $I \cap \hat{p}[i] > k[i] = \varnothing$, else go to step one.

In order to compute the threshold vector k, the variance of the Gaussian noise perturbation at the threshold comparison in step 1 must be obtained. For simplicity, the threshold for the first iteration will be computed. In the case of false alarm, energy will be removed from the Gaussian noise PSD resulting in a lower probability of false alarm among the iterations. By assuming AWGN before channelization we have that

$$\tilde{r} = FWHn, \quad (7)$$

where F is a Fourier matrix, W is a diagonal matrix containing the window coefficients, H is a convolution matrix containing the channelization coefficients and n is a vector containing the zero mean Gaussian distributed noise samples. The variance of $\tilde{r}$ can be determined as

$$\sigma_{\tilde{r}} = D\{E\{FWHnn^H H^H W^H F^H\}\} = \sigma_n^2 D\{FWHH^H W^H F^H\}, \quad (8)$$

where $E\{X\}$ represents the expectation operation and $D\{X\}$ represents the diagonal. From $\sigma_{\tilde{r}}$, we may derive the threshold vector *k*as

$$k = \sigma_{\tilde{r}} \left[1 + \frac{Q^{-1}(P_{FA})}{\sqrt{M}} \right], \quad (9)$$

where $Q(x)$ is the tail probability of a normalized zero-mean Gaussian random variable and P_{FA} is the Probability of False Alarm.

Figure 40 shows improvement in terms of P_{FA} provided by the leakage cancellation scheme described in this article, when applied to a simulated LTE OFDMA block. It is possible to observe how the P_{FA} severally degrades when leakage energy becomes dominant over the noise term when it is not cancelled (red curve). At -4 dB SNR, the proposed leakage cancellation scheme provides an improvement of 30% in terms of P_{FA} maintaining the degradation with the increasing SNR small. In Section V these results will be verified by means of sensing measurements.

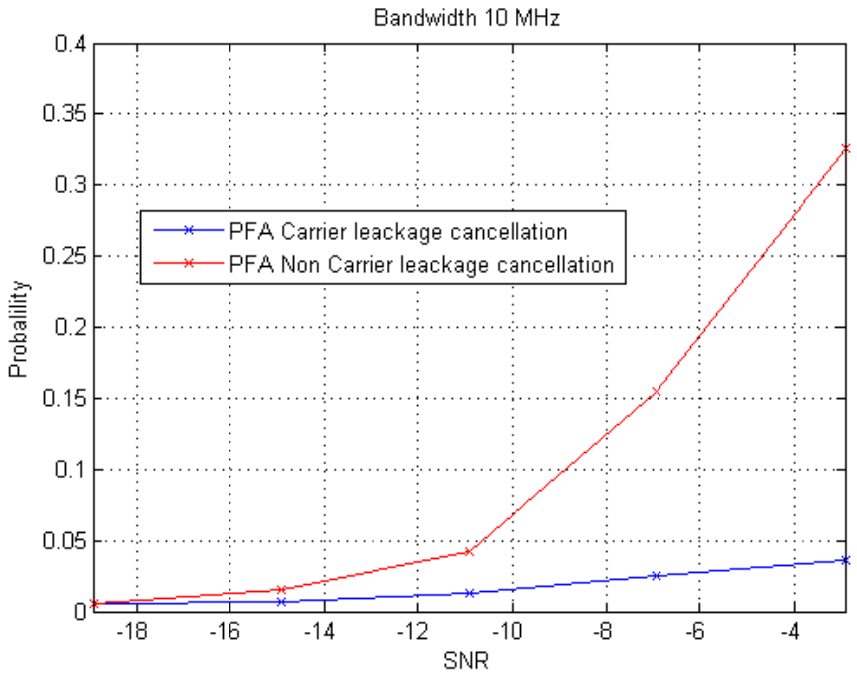

Figure 40: Influence of carrier leakage on the Probability of False Alarm.

3.1.2 Test Setup

The setup is illustrated in Figure 41. The baseband waveform is generated, up-sampled and filtered in Matlab. The waveform is downloaded into a baseband signal generator which produces analog I and Q signals, sampled at 73.14 MHz in case of the DVB-T signal and 40 MHz in case of the LTE signal. A Vector signal generator handles the up-conversion of the analog baseband signal to the desired RF frequency, 480 MHz for the DVB-T signal and 2.6 GHz for the LTE signal, at the wanted power level. Via a power splitter this signal is injected simultaneously into the RFIC and a spectrum analyzer. Both the RFIC and the spectrum analyzer implement the down-conversion from RF to digital baseband signals which are transferred to Matlab for further processing. All measurement equipment is synchronized to the reference clock, which is also the ADC sampling clock, of the RFIC. When the spectrum analyzer is used for down-conversion the captured signal is transferred directly to the host PC. In case the RFIC takes care of the down-conversion the signal is stored in on chip memory of a Xilinx Virtex-IV FPGA. When the memory buffer on the FPGA is filled the signal is transferred to the host PC via a PCI link. The memory can contain 128k complex samples, which are then used for the sensing. The same amount of samples is captured with the spectrum analyzer.

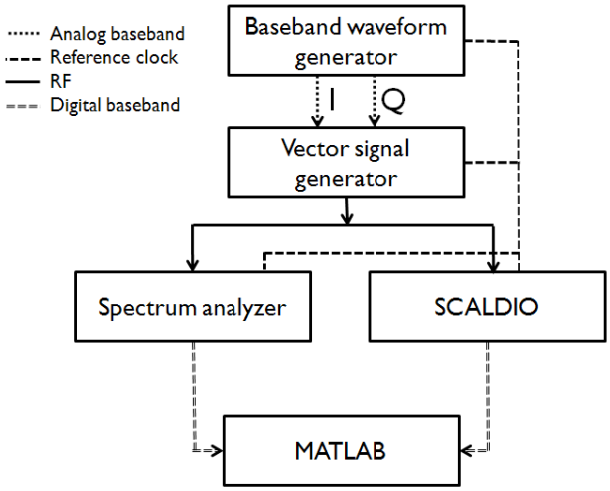

Figure 41: Measurement setup.

3.1.3 Functionality verification

3.1.3.1 Cyclostationary detection for DVB-T verification

The functionality is evaluated on 1000 signal captures for each measured input power level, made with the RFIC and compared to low complexity energy detection. Figure 42 shows the Receiver Operating Curves (ROC) for both the energy detection and cyclostationary detection based on the cyclic prefix for input powers from -116dBm up to – 94dBm.

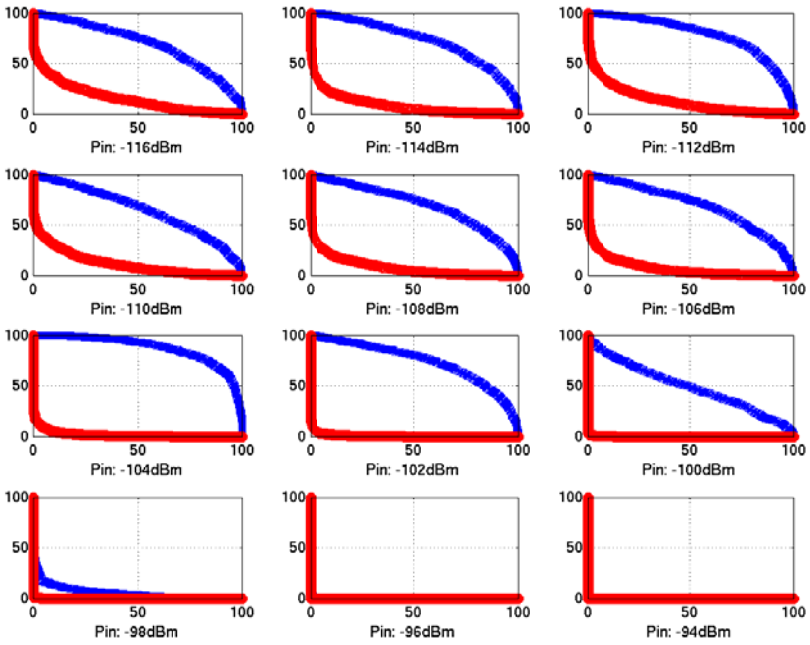

Figure 42: ROC curves - Energy vs. Cyclostationary Detection.

The blue curves show measured performance of the energy detection and the red curves show the performance of the cyclostationary detection. The performance of the energy detection starts to degrade for input powers below -98 dBm. A similar degradation can be observed below -104 dBm for the performance of the cyclostationary detection, hence leading to a performance advantage of 6 dB for the cyclostationary detection. Furthermore the performance degradation presents itself more gradually for the cyclostationary detection. This is shown more clearly in Figure 43.

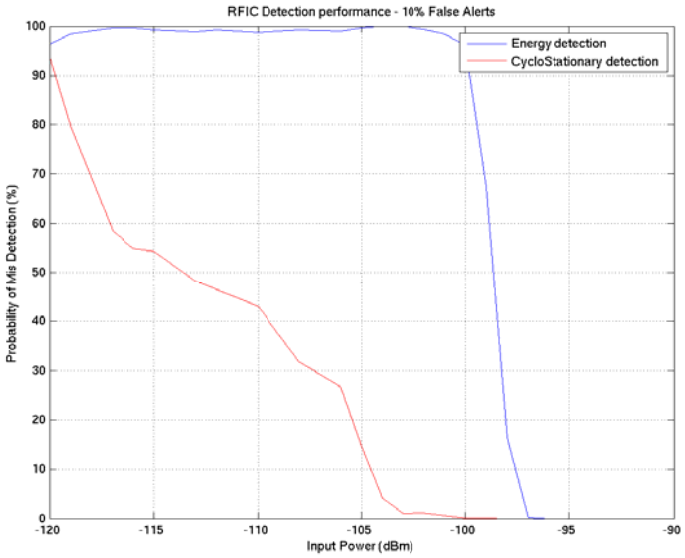

Figure 43: RFIC Detection Performance.

In Figure 43 the performance of both detection algorithms is shown in the case where a probability of false alarms is allowed to be up to 10%. The SNR wall is clearly present for the energy detection scheme around -97 dBm, which is because of the noise uncertainty of the RFIC [8]. The cyclostationary sensing is expected to be less sensitive to this SNR wall, hence the more smooth performance degradation. This result is hence as expected. A sensitivity of -103 dBm is achieved with the RFIC, for the cyclostationary sensing. This is well above the target sensitivity that would be required if the worst-case sensing thresholds that could be expected are targeted. It is however already well accepted in the research community that low-power and low-cost sensing cannot achieve those sensitivity levels. In the next section, we will compare how those results compare to the performance that can be achieved with expensive test equipment.

3.1.3.2 LTE resource block detection verification

Measurement scenario: For the measurement, three contiguous 10 MHz LTE bands are generated at the transmitter side as illustrated in Figure 44. The inter band spacing is 10 MHz. Each of the bands is configured with the same resource allocation, one active resource in every two. This scenario is just for proof-of-concept of the sensing of OFDMA signals. At the receiver side, the baseband analog filters are configured for a 20 MHz cutoff frequency. The ADC sampling frequency is 40 MHz.

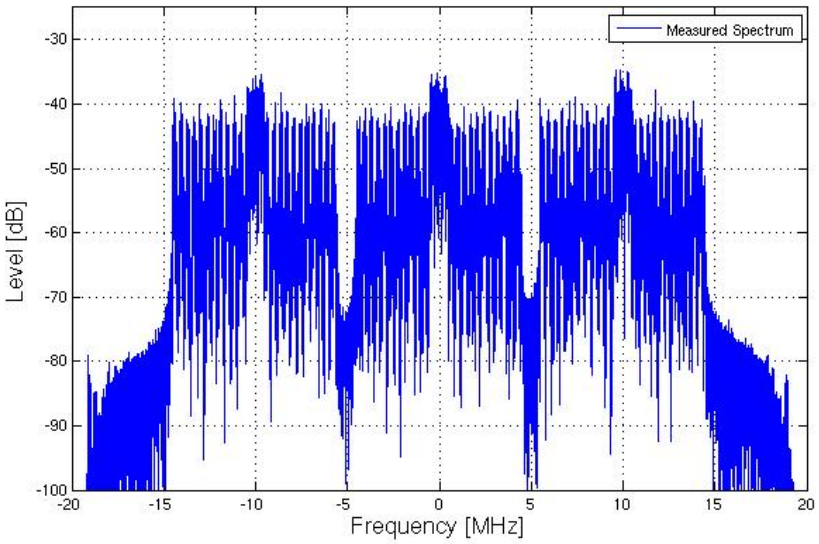

Figure 44: Transmitted signal for the LTE test.

Flexible filter: From the three LTE band transmitted the middle one is first selected with the flexible filter branch. Regarding the digital front end configuration, an order 3 CIC with down-sampling factor 2 is followed by a 21 tap FIR with a final adjacent band rejection of 44 dB. The resulting signal is re-sampled to a final sampling rate of 11.52 MHz, which is the LTE symbol rate for the considered middle band. Together with a spectrum shift of 0 Hz, this configuration results in the selection of the central 10 MHz band of Figure 44.

Leakage removal functionality: To determine the OFDMA Resource Block allocations in that band, a 64 point FFT is required on the 11.52 MHz re-sampled signal. The multiband sensing is performed over a 64 point FFT preceded by a Chebyshev windowing of the 11.52 MHz re-sampled signal. The Chebyshev window is configured for 21 dB of side lobe attenuation. It is then possible to iteratively select the largest peak, determine the power and compute the leakage in the adjacent bins because of the Chebyshev window. The result of this operation is shown in Figure 45. The upper figure gives the received signal after the filtering and 64 point FFT. The red line is the estimated noise floor. Without leakage removal, a lot of the bins look occupied. We know that is however not the case since we know the transmitted signal. After leakage removal, in the lower plot, we see that detection is improved significantly. All empty bins have now a power level below the noise floor. This proves that the carrier leakage removal is needed and working.

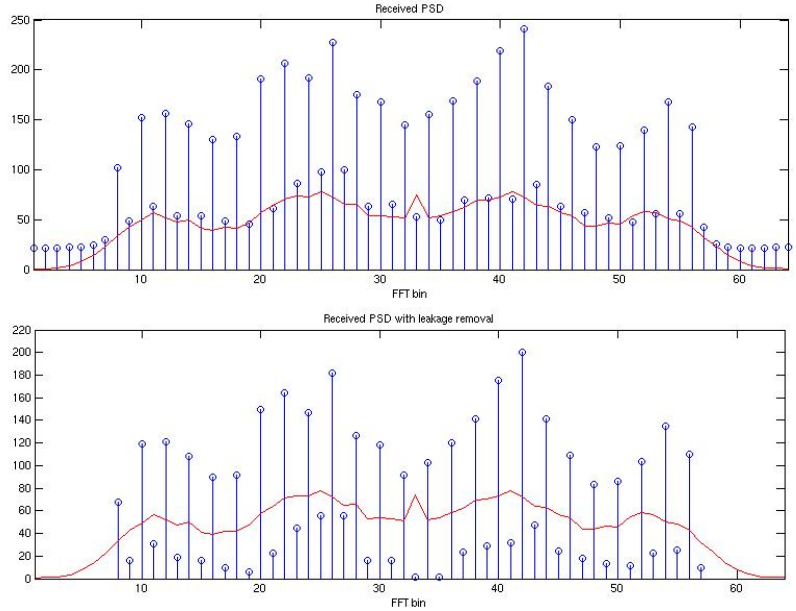

Figure 45: Sensing the middle LTE transmitted band with and without leakage removal.

3.1.4 Comparison with spectrum analyzer

In this section the performance of the RFIC is compared to the performance of a spectrum analyzer. All equipment is synchronized to the reference clock of the RFIC, therefore there is no CFO present in any capture. The RFIC measurement results are based on 1000 captures for each measured input power level, for the spectrum analyzer measurement results 100 captures are used.

Both curves exhibit a similar on/off trend because of the SNR wall, however for the RFIC this behavior occurs at input powers 12 dB higher compared to the spectrum analyzer performance. This is because the noise uncertainty of the test equipment is a lot smaller, giving rise to an improved performance of the energy detector.

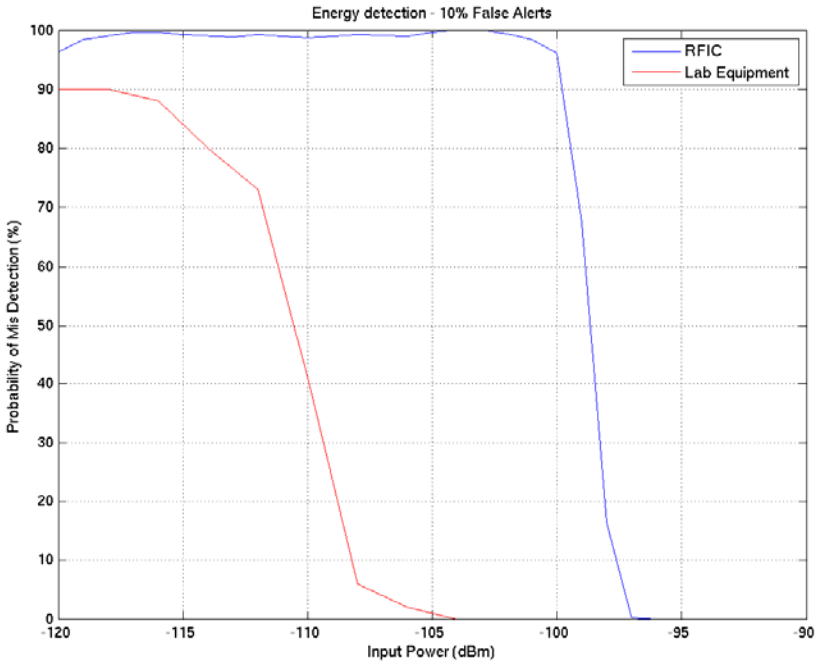

Figure 46: Energy Detection: RFIC Vs Spectrum Analyzer.

This performance gap clearly shows the need for more advanced algorithms such as the cyclostationary detection. Indeed, for the cyclostationary detector based on the cyclic prefix, the performance of the RFIC and the test equipment is comparable (Figure 46). The noise uncertainty is dominating the performance of the sensing less in that case, and the achieved performance is hence a lot closer. This motivates the need for more advanced sensing algorithms. It also shows that with more advanced sensing algorithms, low-power and low-cost sensing equipment can achieve a performance comparable to the test equipment.

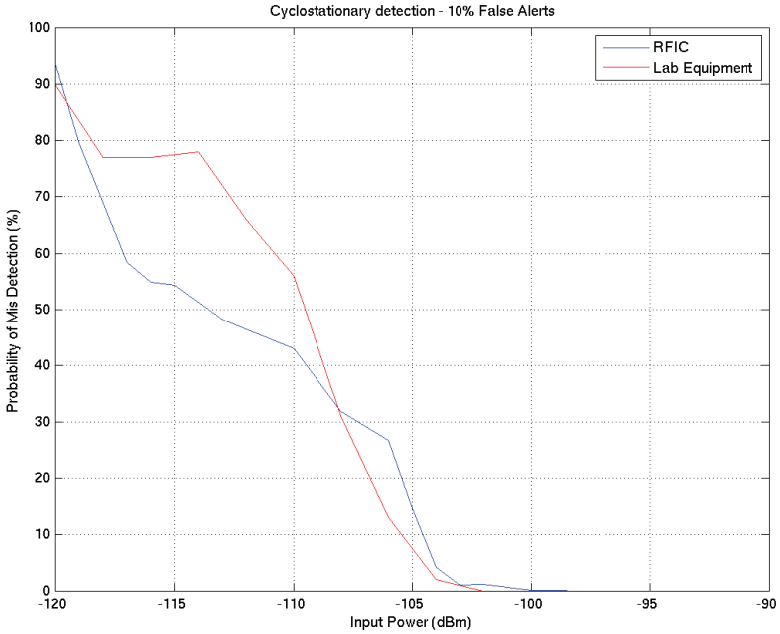

Figure 47: Cyclostationary detection: RFIC vs. Spectrum analyzer.

3.1.5 Sensing with antenna combining selection

A reconfigurable sensing engine equipped with a flexible analog front-end and a digital front-end for sensing has been developed in FARAMIR project. The reconfigurable analog front-end allows scanning the spectrum in a broad range of frequency bands. A single antenna is applied in this sensing engine considering additional signaling and hardware costs occurred by cooperative sensing and single-user multiple-antenna techniques. In order to further enhance the ability of this FARAMIR sensing engine meanwhile handling the trade-off between high sensing performance and low signaling/hardware costs, we propose a scheme [9] that requires no extra signaling and low hardware costs, while still offering substantial diversity gains for enhancing the sensing performance. This is achieved by combining antenna selection with the spectrum sensing mechanism, thus keeping a low implementation complexity as only a subset of RF chains are used in a given time period.

We assume that there are Ω $(1 < \Omega < M)$ RF chains that are connected to the antennas. We equally divide each sensing slot τ into $\frac{M}{\Omega}$ sub-slots, denoted as τ_s (i.e., $\tau = \frac{M}{\Omega}\tau_s$). In each sub-slot, Ω antennas are used to perform spectrum sensing simultaneously. These antennas are exclusively connected to the RF chains, i.e., the antennas that are used for sensing in one sub-slot should not be used in another. Such an antenna selection based spectrum sensing scheme with M antennas and Ω RF chains is illustrated in Fig. 1.

We consider the case when the channel coefficient h_m is unknown. $g_m = \frac{1}{\sqrt{M}}$ is chosen as the weighting factor for each antenna. For the proposed scheme with M antennas and Ω RF chains,

one can confirm that given the target detection probability P_{d_t}, or the target false alarm probability P_{fa_t}, the sensing performance is given by

$$P_{fa} = Q\left(\sqrt{1+\frac{2}{M}\sum_{m=1}^{M}\gamma_m}\,Q^{-1}(P_{d\_t})+\frac{\sqrt{N\Omega}}{M}\sum_{m=1}^{M}\gamma_m\right) \quad (1)$$

$$P_d = Q\left(\frac{1}{\sqrt{1+\frac{2}{M}\sum_{m=1}^{M}\gamma_m}}\left(Q^{-1}(P_{fa\_t})-\frac{\sqrt{N\Omega}}{M}\sum_{m=1}^{M}\gamma_m\right)\right) \quad (2)$$

where $Q(x)=\frac{1}{\sqrt{2\pi}}\int_x^{\infty}e^{-\frac{t^2}{2}}dt$ is Gaussian tail probability, $\gamma_m=|h_m|^2\gamma$ is the average signal-to-noise ratio (SNR) measured from the m^{th} antenna, and $\gamma=\frac{\sigma_s^2}{\sigma_u^2}$.

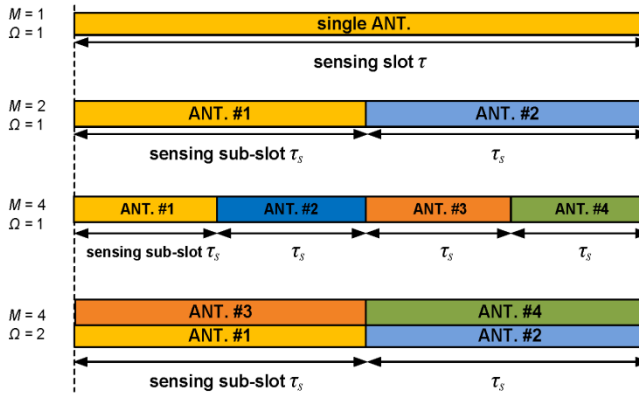

Figure 48: System model of antenna selection based spectrum sensing.

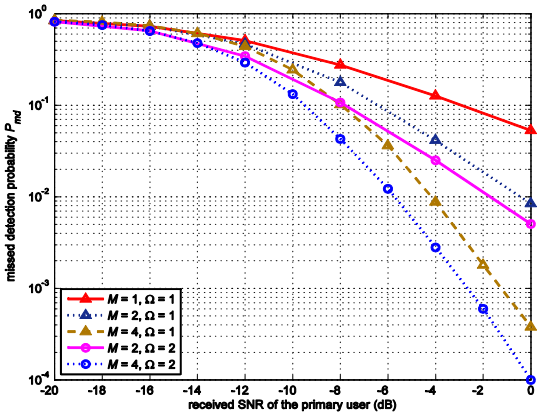

Figure 49: Missed detection probability as a function of received SNR of the primary user (P_(fa_t)=0.1, τ=0.1 ms).

We show the missed detection probability of the proposed antenna selection sensing scheme with different number of antennas and RF chains. It shows that, when the number of RF chains is the same, the performance in terms of missed detection probability is greatly improved when the number of antennas increases. It is also observed that, the proposed antenna selection based sensing scheme using four antennas and one RF chain $(M=4,\Omega=1)$ yields a better sensing performance than the conventional multi-antenna sensing with two antennas and two RF chains $(M=2,\Omega=2)$ when the missed detection probability is smaller than 0.1, and the performance gain compared to conventional multi-antenna sensing scheme becomes more substantial as the missed detection probability becomes smaller. Since a smaller missed detection probability is always preferred in practice, the proposed antenna selection based sensing scheme achieves a better performance with fewer number of RF chains (thus less hardware cost) for practical CR systems. Comparing the performance of the proposed sensing scheme with the same number of

antennas and different RF chains shows that, using a larger number of RF chains can further improved the sensing performance compared to that using only one RF chain. For example, compared to the case where $M = 4$ antennas and one RF chain is used, a performance gain of about 2 dB at the missed detection probability of 0.01 can be achieved when $\Omega = 2$ RF chains are used. In all the curves presented, a diversity order of M can be observed.

3.2 Comparison platforms from prototyping

In this section we provide an overview and comparison statistics on performance of various other sensing engines which are being used in our prototyping efforts in addition to the FARAMIR sensing engine. Most of these are either commercial off-the-shelf spectrum sensors, or software defined radio platforms on which the necessary software components have been developed by the consortium. For a more complete review on other spectrum sensor solutions we refer the reader to the state of the art deliverable D2.1, and for further references on recently proposed and especially in the case of CFRS also commercially available spectrum sensors see [30]-[33].

3.2.1 Sensing engines description

3.2.1.1 USRP2

USRP2 is an open source hardware platform from the USRP product family developed by Ettus Research/National Instruments [10]. The hardware, composed of a motherboard and a daughterboard, enables flexible tuning of the Tx/Rx characteristics of the radio. Different daughterboards allow broad selection of frequency ranges. It supports fast analog-to-digital and digital-to-analog switching, thus allowing inspection of wider spectrum bands and higher dynamic range. USRP2 and GNU Radio [11], an open source software development toolkit, form a Software Defined Radio (SDR) platform where the GNU Radio provides the basic signal processing blocks. Figure 50 depicts the USRP2 device.

Figure 50: Universal Software Radio Peripheral 2 (USRP2).

Hardware features: The USRP2 hardware is composed of two main hardware boards, i.e. a *motherboard*, performing the high sample rate operations and communicating with the host PC, and a *daughterboard*, representing the RF part. Different daughterboards allow selection of broad frequency ranges. The main characteristics of the motherboard are:

- *Spartan 3 Field Programmable Gate Array (FPGA)* with *50 MHz 32-bit RISC microprocessor* performing the high-rate operations of digital up and down conversion (the lower sample rate operations are performed on the host PC side);
- *1 MB high-speed on-board SRAM memory*, while the FPGA and firmware codes are read from an SD card;
- Two *ADC converters*, for the real and imaginary part of the signal, perform sampling with rate of 100 Msamples/s with data resolution of 14 bits;
- Two *DAC converters*, for the real and imaginary part of the signal, perform sampling with rate of 100 Msamples/s and data resolution of 16 bits;
- *Decimation and interpolation factors* in the range 4-512, resulting in RF bandwidths of ~195 kHz up to 25 MHz;
- *Gigabit Ethernet interface* for connection to the host computer supporting a maximum transfer rate of 125 MB/s;
- An *expansion interface* allowing connection of multiple USRP2 devices into fully coherent multiple antenna systems for MIMO;
- Support for wide variety of RF daughterboards (e.g. WBX is a transceiver daughterboard in the range 50 MHz to 2.2 GHz, RFX400 is a transceiver daughterboard in the range of 400-500 MHz, RFX2400 is a transceiver daughterboard in the range of 2.3-2.9 GHz);
- Host-PC dependent framework (low rate operations are performed on the host PC side).

Software features: USRP2 hardware can be programmed using the following Integrated Development Environment (IDE) solutions:

- GNU Radio - runs on Linux OS;
- Matlab Simulink, LabView - run on Windows OS;
- Universal Hardware Driver (UHD).

GNU Radio is an open source software toolkit for developing software defined radio applications. Generally, GNU Radio turns radio hardware problems into software problems. It enables definition of transmitting waveforms and demodulation of received waveforms in software processing manner instead of processing done in analog circuitry or analog circuitry combined with digital chips. The basic principle functions in a two-tier architecture with creation of signal processing blocks in the first tier and their connection in a graph-flow manner in the second tier. More generally, GNU Radio encompasses the following:

- An API for creating signal blocks (C++/Python);

- A runtime environment for signal processing;
- A library of signal processing blocks;
- User scripts and applications;
- Hardware drivers (USRP/USRP2/VRT);
- An application for creating flow graphs (GRC).

Figure 51 depicts the relation between GNU Radio and USRP2. As a software defined radio tool, it should be placed as near as possible to the antenna.

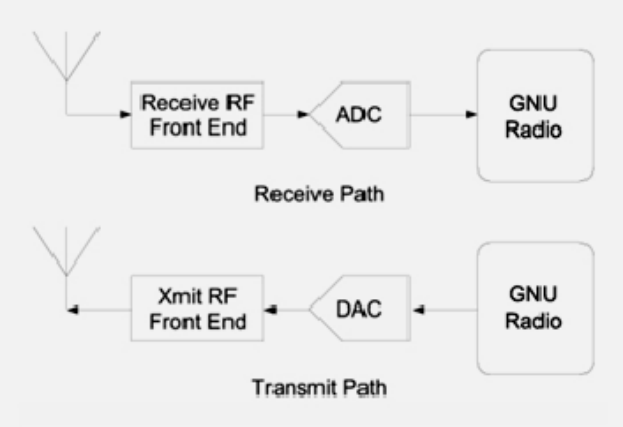

Figure 51: GNU Radio software development tool implementation.

Spectrum sensing features: The USRP2 hardware can be used as a spectrum sensing device. The main features of the hardware in terms of sensing capabilities (mainly related to energy detection) are:

- The USRP2 performs IQ sampling. It has two input channels, therefore, the received power is represented as 10 times logarithm of the summed magnitude squared I and Q components;
- The resolution bandwidth of the USRP2 depends on the selected decimation rate. So, the maximum decimation rate of 512 results in a resolution bandwidth of 195 kHz, while the minimum decimation rate of 4 results in a resolution bandwidth of 25 MHz;
- USRP2 can perform Fast Fourier Transform. Using this feature, higher and broader set of frequency resolutions can be achieved;
- The frequency switching delay is below 200 μs for the transceiver daughterboards;

- The IQ sampling time depends on the selected decimation rate, e.g. resolution bandwidth of 20 MHz RBW results in 50 ns inter-sample delay.

Three types of measurement modes can be distinguished using the different set of settings in the GNU radio and the USRP2:

- *real-time* measurement mode – offering real-time analysis of spectrum bands, with FFT as an option;

- *sweeping* measurement mode – offering spectrum band analysis in sweeping manner;

- *hybrid* measurement mode – offering analysis of spectrum bands in sweeping manner combined with FFT analysis of separate spectrum blocks.

3.2.1.2 Texas Instrument eZ450 CC2500

The eZ430-RF2500 is small and portable wireless development tool by Texas Instruments. TI eZ430 development tool includes all the hardware and software required to develop an entire wireless project in a single USB-based device. The tool includes a USB powered emulator to program and debug applications. Projects are easy to develop and can be instantly deployed. TI eZ430 is not originally a spectrum sensing device and needs custom made software to perform sensing. Figure 52 depicts a TI eZ430-RF2500 device.

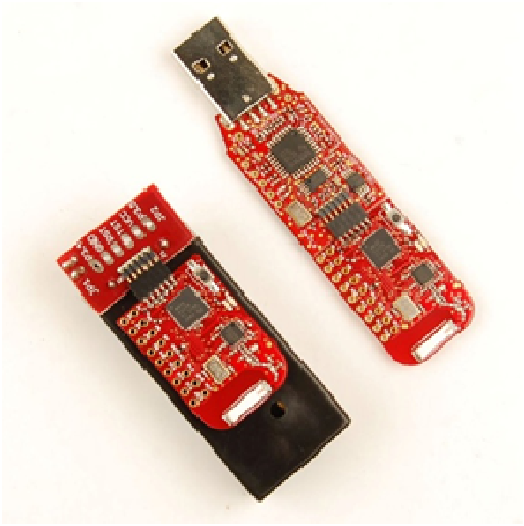

Figure 52: Texas Instruments eZ430-RF2500.

Hardware features: TI eZ430 hardware is based on 16 MHz 16 bit RISC processor. It has small Random Access Memory (RAM) of 1 KB and 32 KB of Read-only Memory (ROM). This device is based on the TI MSP430F2274 microcontroller and CC2500 wireless transceiver in 2.4 GHz band (2400 – 2485 MHz). The microcontroller can perform 16 million instructions per second (MIPS), has two

built-in operational amplifiers and a 200 ksps analog-to-digital converter. It can support five low-power modes using only 700 nA in standby state. TI eZ430 can be powered by host computer using USB connector or by AAA batteries with 1200 mAh. The device has a programmable data rate up to 500 kbps

Software features: The eZ430-RF2500 uses the IAR Embedded Workbench IDE or Code Composer Essentials (CCE) to write, debug and download an application on the TI eZ430. The IDE has been developed with a Graphical User Interface (GUI) and the development and compiling of the applications is performed using C programming language. The eZ430-RF2500 also uses an additional TI developed application called SmartRF studio. It can be used for configuring the RF front end of the eZ430-RF2500 in terms: filter band, center frequency, receive and transmit gain etc.

Spectrum sensing features: The eZ430 can be used as a spectrum sensing device with appropriate custom made modifications. According to TI eZ430 spectrum measurement properties, this device belongs to the low-end group of devices. It covers the full 2.4GHz ISM band with a resolution bandwidth of maximum of 812 kHz. The TI eZ430 has a noise floor of -83 dBm using a resolution bandwidth of 812 kHz and -104 dBm using a resolution bandwidth of 203 kHz. The modest hardware characteristics of the device lead to a non-linear input/output characteristic. Additional features in terms of sensing capabilities, particularly energy detection, are the following:

- RSSI sampling based sampling. I/Q sampling is not feasible;
- Different sets of frequency steps. The maximal value is 400 kHz;
- Sampling rate of 310μs satisfying good sensing agility of the device;
- Switching delay between frequencies of 809 μs.

3.2.1.3 Sun SPOT CC2420

Sun SPOT (Sun Small Programmable Object Technology) [16] represents a small Java-based wireless device developed at Oracle Labs [17]. The device is built upon the IEEE 802.15.4 standard and is designed to perform as wireless sensor network node. The SPOTs sensor board has inbuilt temperature, light and acceleration sensors, 8 three color LEDs, two momentary switches and input/output pins where additional sensors or actuators can be attached. Spectrum sensing is not an original inbuilt feature of the device. This capability is custom made, i.e. the Sun SPOT can perform spectrum sensing based on the *Received Signal Strength Indicator (RSSI)* indicator in each received information packet. As a sensing device, it belongs to the group of low-end, low quality devices. Figure 53 depicts a Sun SPOT device.

Figure 53: Sun Small Programmable Object Technology (Sun SPOT).

Hardware features: Sun SPOTs hardware is designed to enable independent function of the node in a wireless sensor network. These devices can function either connected to a host computer or powered by a battery. Sun SPOTs are equipped with rechargeable lithium-ion batteries and consume small amount of energy in the power saving operation mode. Appropriate software provides the automatic battery management. Sun SPOTs are equipped with 180 MHz 32 bit processor, 512K RAM and 4MB Flash memory. The Sun SPOTs' in-built radio is CC2420 transceiver which is IEEE 802.15.4 compliant operating in 2.4 GHz ISM band.

Software features: Sun SPOTs are programmable in Java. NetBeans or any other standard Java integrated development environments are used to create SunSPOT MIDlet applications. Sun SPOT Manager Tool (SPOTWorld), supplemented with NetBeans, is a tool for programming, configuring, managing and monitoring Sun SPOT devices. The SPOTWorld allows Sun SPOT developers to program and manage hundreds of devices spread out over a large area. The SPOTWorld also offers Solarium emulator for experimental purposes with virtual SPOTs.

Sun SPOT applications are built on Squawk Java Virtual Machine, intended to work with small, resource constrained devices. The Squawk enables programming in Java which gives portability, ease of debugging and good maintainability. The management and deployment of the MIDlet applications is handled by "ant" scripts, which can be called from the development environment, from the command line, or the Solarium.

Spectrum sensing features: Sun SPOTs operate in a multipoint-to-point communication manner. Remote devices send their RSSI readings to a base station connected to a host computer. To enable this, additional user code is required in the Sun SPOTs' library suite. The additional code enables the Sun SPOTs to receive the RSSI directly from the radio chip and to set the operating frequency at any integer value between 2253MHz and 2740MHz in 1MHz steps.

3.2.1.4 Test Setup

This section provides the information on the scenario employed to test the performances of the inspected MCDs. The MCDs of interest in the performance testings are the USRP2 device, TI eZ430 RF2500 and the Sun SPOT based spectrum sensors.

The Anritsu MS2690A signal analyzer/generator [18] is used to generate a reference signal. The generated signal is a constant BPSK signal with a bandwidth of 500 KHz at a central frequency of 2.401 MHz in order to avoid the ISM band activity as much as possible. The tested range of the transmit power is between -100 and -80 dBm for the reference signal with steps of 2 dBm.

The USRP2 device in the test setup is connected directly to the signal generator via a cable. The measured cable losses are around 2 dB (taken into account in the performance evaluation). The USRP2 used the RFX2400 daughterboard for the ISM 2.4 GHz band. The resolution bandwidth was set to 2 MHz and the receiver gain to 40 dB, which is a most commonly used configuration (due to the most satisfying trade-off between the sensitivity and the linearity of input-output power characteristics). The statistics for each of the input power test-cases were calculated based on one million samples. Furthermore, the measurements on the noise floor were completed in parallel to eliminate the effect of noise uncertainty.

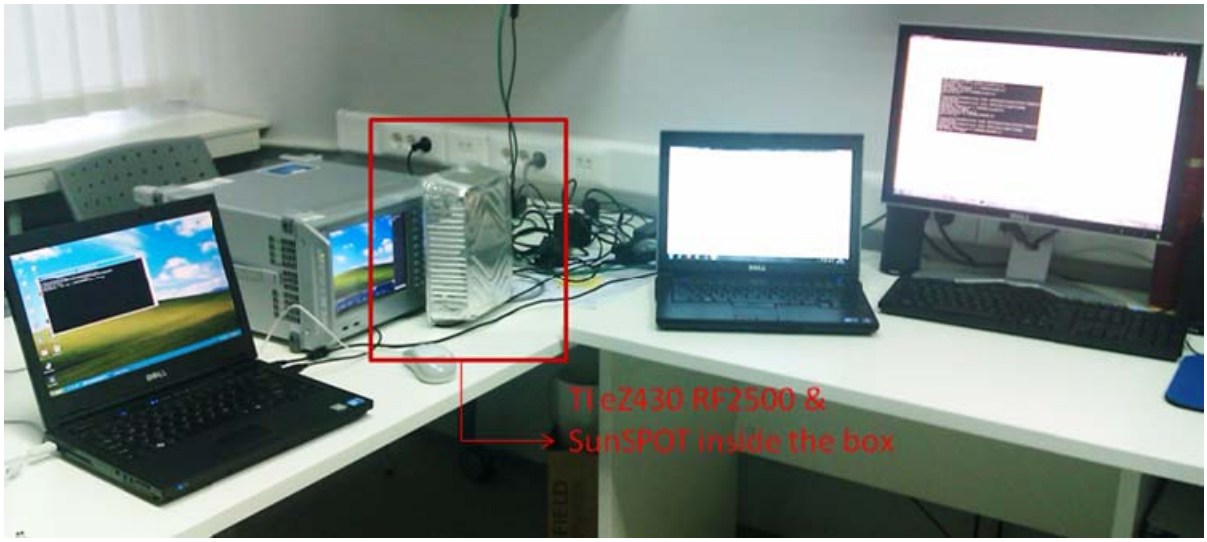

Figure 54: TI eZ430 RF2500 and Sun SPOT testing scenario.

The Sun SPOT and the TI eZ430 RF2500 spectrum sensors cannot be connected to the signal generator directly via a cable since they have on-board antennas. Therefore the measurements performed with these devices were conducted in an aluminum case to improvise a faraday cage. The test setup for these devices is depicted on Figure 54. A VERT2450 antenna was used by the signal generator to provide the reference BPSK signal. The measured antennas' and propagation losses were 14 and 16 dB for the Sun SPOT and TI eZ430 RF2500 device, again taken into account for the performances evaluation. The Sun SPOT device used the single possible 2 MHz bandwidth and the TI eZ430 RF2500 used the highest possible 812.5 KHz bandwidth for the measurements. The lower receiving bandwidth of the TI based MCD device is giving the referred an advantage to

the other two devices in theory with 3.9 dB. The test statistics for the TI eZ430 RF2500 MCD is 100000 samples, while for the Sun SPOT MCD 500000 samples. Noise measurements with the same amount of statistics were also performed in parallel to account for the noise uncertainty problem.

The next sections provide the gathered results and the performances comparisons between the inspected MCDs.

3.2.2 Sensing Performance

3.2.2.1 USRP2

This subsection gives an insight of the spectrum sensing performance of the USRP2, focusing on:

- Energy detector (elaborates the energy detection capabilities of the MCD)
- Energy detection performances (elaborates the energy detection performances of the MCD)

Energy detector: The classical energy detector option of the USRP2 based sniffer implementation is consisted of the minimum hold, maximum hold and the mean hold detector types. This subsection presents applications of the USRP2 based energy detector, that focus on frequency spectrum measurements. In the USRP2 sniffer implementation, it can be included in medium and long term spectrum measurement campaigns. Wideband measurements can mainly employ the sweeping and hybrid mode of operation of the sniffer, while band–specific measurements can be performed with the real-time measurement mode of the sniffer (if the bandwidth of interest does not surpass the USRP2 hardware limitations, i.e. the maximal bandwidth of 25 MHz).

Figure 55 depicts the general USRP2s energy detection architecture[12]. It contains six GNU Radio and one custom made processing blocks:

1. ***usrp2_source_32fc:*** creates the USRP2 source and controls the hardware (sets up sampling rate and tuning frequencies);
2. ***gr_stream_to_vector:*** converts a stream of complex samples to vector of complex samples;
3. ***gr_fft_vcc:*** calculates a FFT transform on input complex samples;
4. ***gr_complex_to_mag_squared:*** calculates squared magnitude (power) on complex samples;
5. ***gr_energy_detector_f:*** custom made processing block - selects between different detector types and sensing modes, initiates frequency switching;
6. ***gr_complex_to_mag:*** calculates magnitude (amplitude) on complex samples;
7. ***gr_fft_vfc:*** calculates a FFT transform on input real samples.

The processing blocks are written in C++ programming language. Processing block no. 5 is customized to add different detector types to GNU Radio. It comprises the mean-hold, the max-hold, the min-hold, the FFT Averaging Ratio (FAR) detector [13] and the Higher Order Statistics (HOS) detector [14].

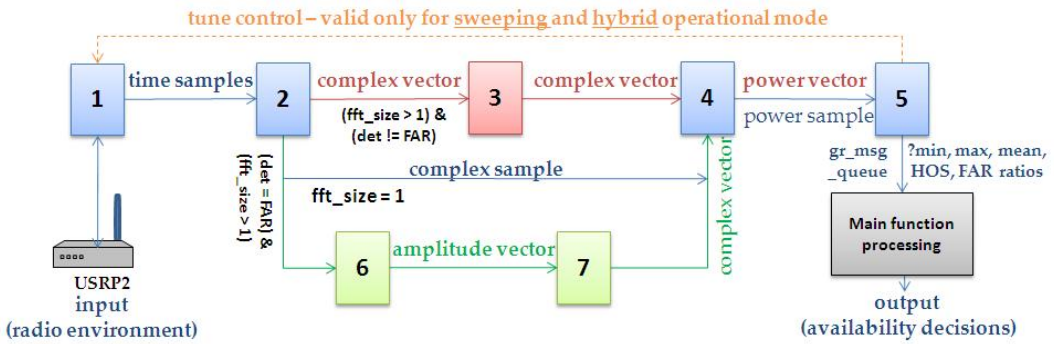

Figure 55: USRP2 based sensing application architecture.

It must be stresed that different USRP2 daughterboards have different noise figure values (e.g. RFX2400 and WBX daughterboards have noise figure in the range of 8-10 dB and 5-7 dB, respectively). Given a background noise level of -174dBm, the USRP2's noise floor is -164 dBm/Hz and -167 dBm/Hz for the respective daughterboards. These thresholds represent the lowest detectable signal level i.e. the sensitivity of the USRP2.

Figure 56 depicts the calibrating curve for the RFX2400 USRP2 daughterboard. The curve in this example can be closely described with the following input/output function:

$$g(x) = \frac{p_1 x^2 + p_2 x + p_3}{x^2 + q_1 x + q_2} \quad (1)$$

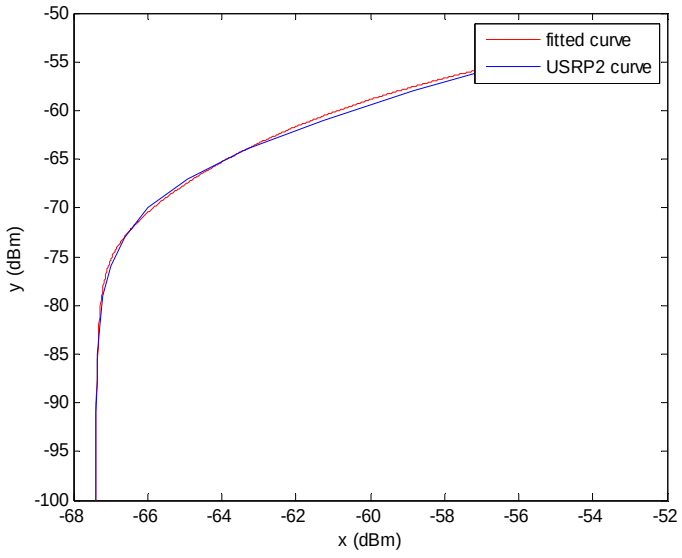

Figure 56: Calibrating curve (fitted one) for RFX 2400 daughterboard on 2412 MHz central frequency, 2 MHz resolution bandwidth (calibration performed with Anritsu MS2690A spectrum analyzer).

Section 3.3.1 also shows the usage of the USRP2 sniffer implementation for the 2.4 GHz ISM band inspection. The measurement setup includes the USRP2 hardware comprising the RFX2400 daughterboards. The focus on the campaign was on the definition of the measurement methodology for the referred frequency band. The USRP2 based sniffer implementation is used mainly in the sweeping and hybrid operational mode with maximum hold energy detection type. The results prove that this USRP2 sniffer implementation offers sensing and detection performances comparable to high end-devices performances. Moreover, it has been verified that the hybrid mode of operation of the sniffer offers significantly better performances than the classical sweeping operational mode [15].

On the other hand wideband measurements have been also performed with the energy detector of the USRP2 based spectrum sniffer implementation using WBX daughterboards [10]. The USRP2 sniffer implementation is used in hybrid operational mode with mean hold detection for these measurements employing FFT with size 16 and receiving bandwidth of 1 MHz, resulting in resolution bandwidth of 62.5 KHz. Figure 57 plots the duty cycle results for the spectrum usage gathered in this campaign.

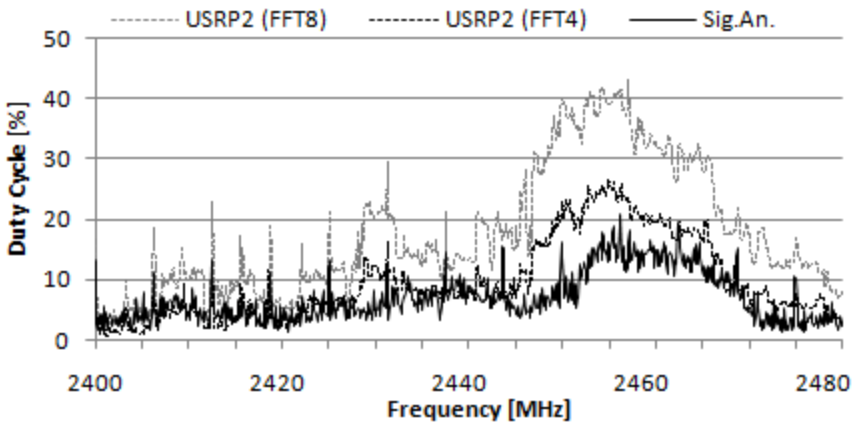

Figure 57: Wideband duty cycle results in working day busy hour gathered using the USPR2 sniffer in hybrid mode.

HOS detector: This part presents the USRP2 sniffer implementation in sweeping measurement mode with HOS (Higher Order Statistics) detection type. The following subsection will elaborate on two HOS based applications:

- Channel selection application;
- Cooperative channel selection application.

The HOS detector of the USRP2 sniffer implementation can be useful in cognitive radios in order to serve the channel selection process. Namely, the consideration of the HOS values in the channel selection improves the decisions for the best channel. The impact of the HOS detection has been

tested on USRP2 devices in a demo based on real environment conditions. The demo is focused on the 2.4GHz ISM band and Figure 58 plots the environment conditions during the tests.

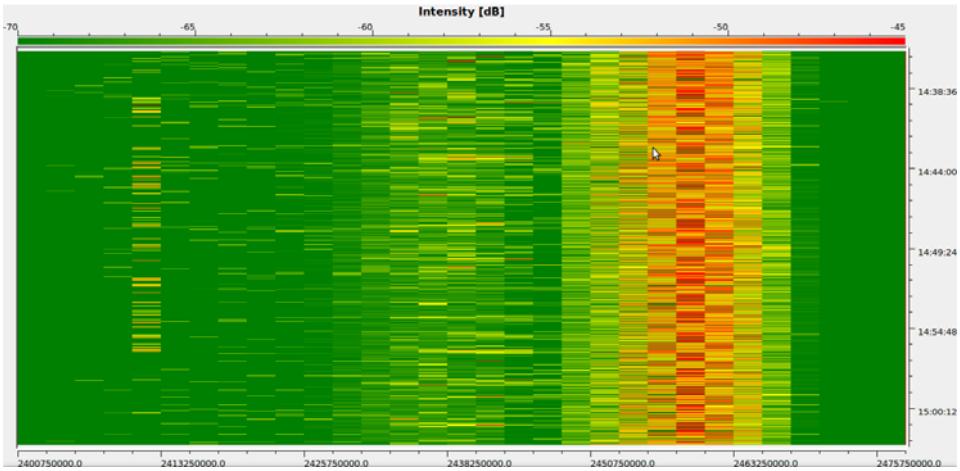

Figure 58: 2.4 GHz ISM spectrum conditions during USRP2 HOS detector.

The channel classification criteria is based on average, skewness and kurtosis values of the received power, each of the referred considered with utility factors a, b and c, respectively. The target is to select channel with statistics closest to the system noise. Figure 59 plots the dependence of the channel selection probability and the utility factors settings. As can be noticed (considering the environment conditions), the worse results are gained when the channel selection depends mostly on the average received power. Predefined frequencies are chosen due to the USRP2 hardware non-idealities – it has different noise power levels and different variations at different frequencies. The decisions on the best channel are more confident when the channel decision depends more on the skewness and kurtosis of the received power. The best performances in terms of channel selection are gained when the decision depends equally on the skewness and kurtosis of the received power, i.e. b=c=0.5. This results reflects the actual environment conditions (Figure 58).

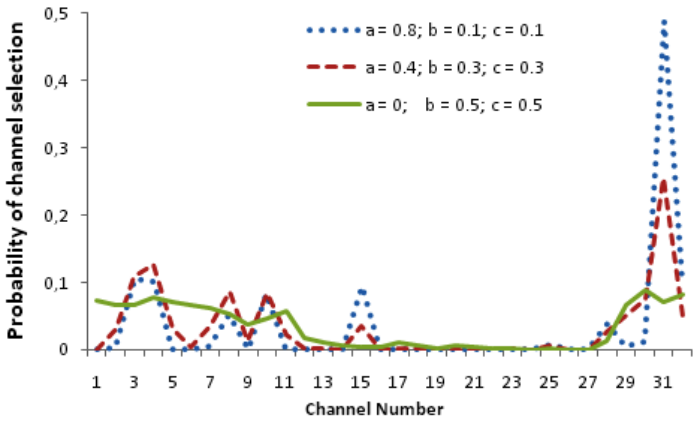

Figure 59: 2.4 GHz ISM spectrum conditions during USRP2 HOS detector.

The dependence of the channel selection with the sensing time duration is presented in Table 5. The channel selection decision improves as the sensing period is higher and sensing period of 3.2 s yields a free channel selection probability of 1 (thus resulting in successful avoidance of busy channels)

Table 5: HOS detection: channel selection probability.

Sensing period	Busy channel selection probability	Free channel selection probability
800 ms	0.226037	0.773963
1600 ms	0.118825	0.881175
3200 ms	0	1

The HOS detection option of the USRP2 based sniffer implementation has also been included in a cooperative demo comprising USRP2 based cognitive nodes. The cooperative nodes exchange their mean power, skewness and kurtosis values in distributed fashion according to the RAC2E [13] rendezvous protocol for cognitive ad-hoc networks. After a simple fusion (with averaging) of the gathered data, a source USRP2 based cognitive radio finds the best channel available and starts the communication with a destination USRP2 node. The targeted band in the cooperative demo is the 2.4 GHz ISM band and the environment conditions are the same as in the demo presented in the previous part (the tests were run simultaneously). The demo aims to prove how the cooperative exchange and fusion of the HOS data among the cooperative nodes improves the best channel decision. The results (Table 6) prove that, as the number of cooperative nodes increases, the channel selection process is more reliable (free channels are most probably to be chosen, whereas busy channels are effectively avoided). As can be noticed, with sensing period of 1600 ms and three cooperative nodes, the probability of selection of a busy channel is 1.5%, and the probability of selection of a free channel is 98.5%.

Table 6: Cooperative HOS detection: channel selection probability.

Sensing period	Cooperative nodes	Busy channel selection probability	Free channel selection probability
800 ms	1	0.226037	0.773963
	2	0.138968	0.861032
	3	0.071736	0.928264
1600 ms	1	0.118825	0.881175
	2	0.029412	0.970588
	3	0.014706	0.985294

Energy detection performance: This subsection presents the performances of the USRP2 based MCD device for the test scenario explained above. The metrics of interest are the probability of detection for different probabilities of false alarm, i.e. the ROC curves, as well as the dependence of the probability of detection on the RF input power. These curves are empirically calculated considering the noise and the signal statistics gathered in parallel to the measurements. The metrics are calculated for a case of averaging 100 samples in mW domain, as gathered USRP2 sensing application.

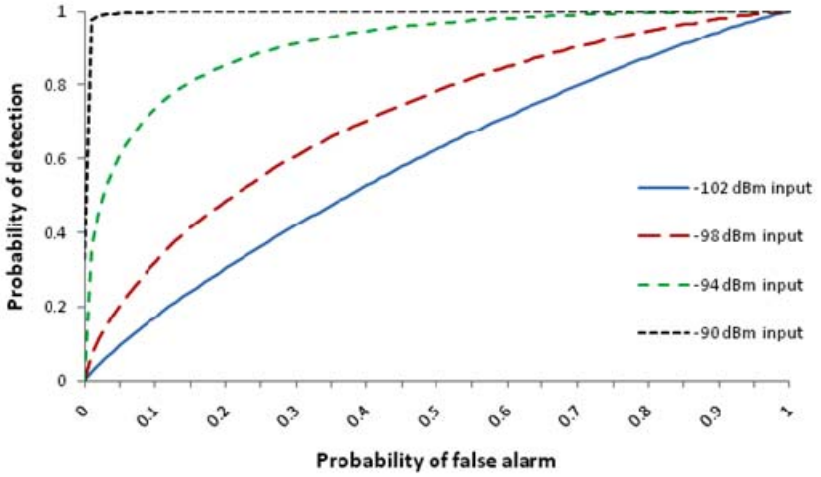

Figure 60: USRP2 ROC curves for input RF power of -102, -98, -94 and -90 dBm, resolution bandwidth of 2 MHz and 40 dB of receiving gain.

Figure 60 presents the ROC curves for the USRP2, representing the detection capabilities of the referred. As can be noticed, input signals of -94 dBm and above can be reliably detected with the USRP2 device. It should be noted that these curves are valid for the specific configuration of the USRP2 – the RFX2400 daughterboard on center frequency of 2.401 GHz, resolution bandwidth of 2 MHz and receiver gain 40 dB. When used with the higher gain settings the USRP2 device would surely result in better sensitivity.

3.2.2.2 T.I. eZ450 RF2500

This subsection gives an insight on the spectrum sensing performance of the T.I. eZ450, focusing on:

- Energy detector (elaborates the energy detection capabilities of the MCD)
- Energy detection performances (elaborates the energy detection performances of the MCD)

Energy detector: Figure 61 depicts an implementation example of TI RF2500. The TI_sense represents a custom developed C-language based sensing application ported on the device.

Application porting and RF part configuration are conducted by a Windows-based application called SmartRF Studio. The Graphical User Interface (GUI) contains host applications for representation of the measured data.

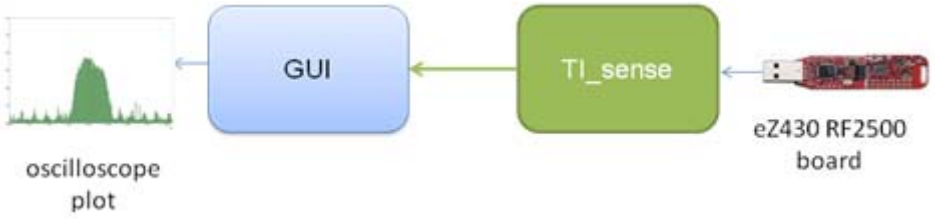

Figure 61: Implementation example of TI RF2500.

Figure 62 represents the relation between the input and the output signal power of the TI RF2500 device for different symbol rates. The figure shows slight non-linearity in the case of high (-20 to 0 dBm) and low (-120 to -95 dBm) signal received power.

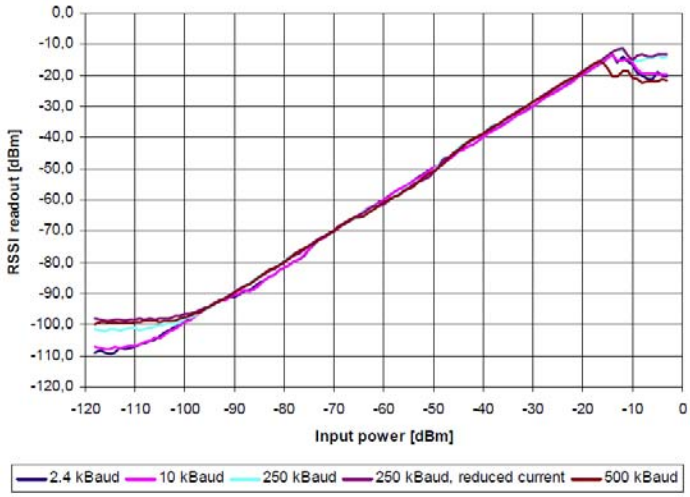

Figure 62: TI RF2500 input/output characteristic.

Due to the lack of processing power, the TI is capable to perform only energy detection measurements. Namely the TI RF2500 is less sensitive than the USRP2. Its noise floor is -83 dBm using a resolution bandwidth of 812 kHz and -104 dBm using a resolution bandwidth of 203 kHz. Figure 63 represents a waterfall plot, as an energy detector application, during eight hour measurement period for a calibrated TI RF2500 device.

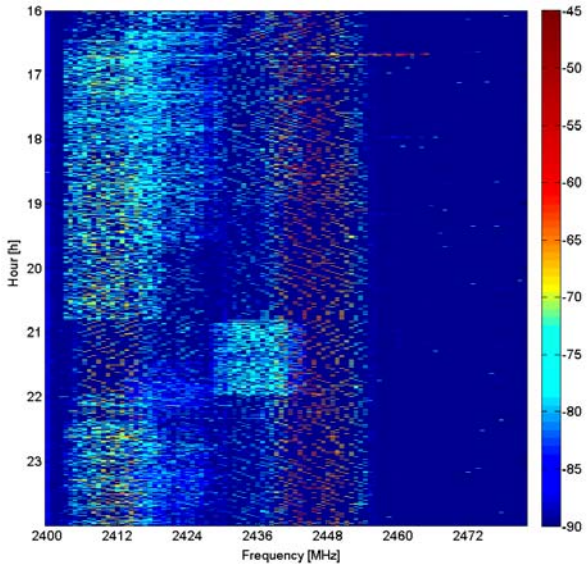

Figure 63: Waterfall plot of spectrum occupancy measured with TI RF2500.

Energy detection performance: This subsection focuses on the detection capabilities of the TI eZ430 RF2500 based MCD device for the test scenario explained above. The metrics of interest are the ROC curves and the probability of detection dependence on the input power. However, several cases of samples averaging are tested also. Namely, the TI eZ430 RF2500 MCD device has 8-bit register for the RSSI value in dBm scale. Therefore, if averaging is not performed it would result in stepped ROC curves not reflecting the actual sensing performances of the device. Four different averaging cases are inspected for the TI eZ430 RF2500 MCD device, i.e. averaging of 10 and 100 samples in mW domain (converted from the received dBm RSSI values) and averaging of 10 and 100 samples in dBm domain (as originally sampled). The ROC curves are calculated empirically from the noise and signal samples.

Figure 64 presents the dependence of the probability of detection on the input RF signal power. As can be seen, the sample detection case can detect signals up to -93 dBm power for the P_{fa} = 1% threshold. However, the case of dBm values averaging significantly increases the detection performances of the TI eZ430 RF2500 MCD device. The case of 100 samples averaging in dBm domain can detect input signals of -110 dBm. This proves as the most sensitive mode of operation of this MCD device.

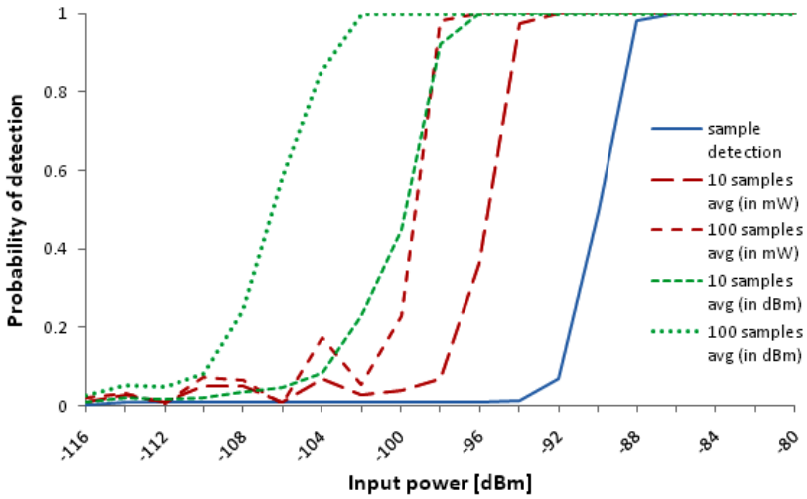

Figure 64: TI eZ430 RF2500 probability of detection dependence on the input signal power for Pfa = 1%, for sample detection and four different averaging cases.

Figure 65 presents the ROC curves of the TI eZ430 RF2500 MCD device for four different cases of input power. It is evident again that the TI eZ430 RF2500 device can detect signals below -110 dBm and reliably detect signals up to power of -110 dBm. The results refer to the most sensitive settings as explained before, the 100 samples averaging in dBm domain.

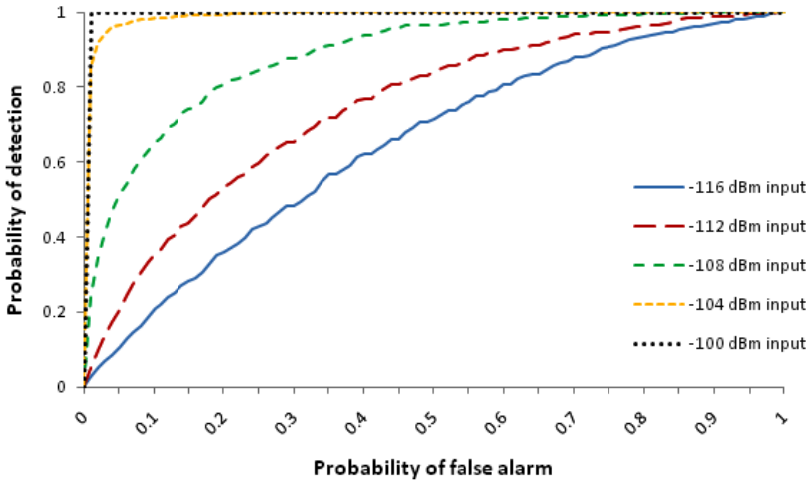

Figure 65: TI eZ430 RF2500 ROC curves for five different input power levels, and sample averaging of 100 RSSI values in dBm domain.

The performance evaluation of the TI eZ430 RF2500 proves that it can provide very satisfying results if appropriately used, even though it is a low cost MCD solution. This device, although sensitive to low input power levels, has a higher variability of the noise samples and, therefore, the averaging case proves to be significantly better than the simple sample detection. It should be also noted that this device also performed the measurements with relatively low resolution bandwidth, resulting in lower noise floor. Furthermore, the device's registers have been set up to give the highest possible sensitivity.

3.2.2.3 Sun SPOT CC2420

This subsection gives an insight in the spectrum sensing performance of the Sun SPOT CC2420, focusing on:

- Energy detector (elaborates the energy detection capabilities of the MCD)
- Energy detection performances (elaborates the energy detection performances of the MCD)

Energy detector: Figure 66 depicts a possible implementation of the Sun SPOT CC2420 devices. SS_sense is a custom developed sensing application deployed at each device. The GUI visualizes a set of host applications, e. g. an oscilloscope plot of signal power.

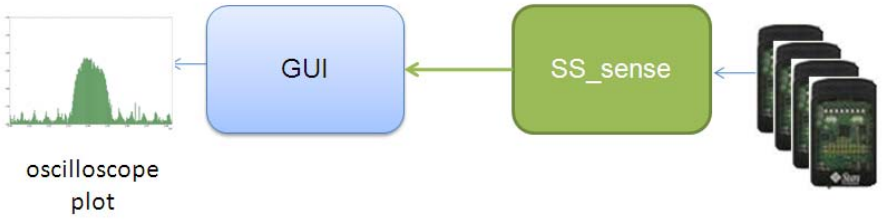

Figure 66: Implementation example of Sun SPOT devices

Sun SPOTs have a noise floor of -100 dBm. The radio chip (i.e. CC2420) has RSSI offset in the range of ±6dBm and almost linear RSSI characteristic with deviations of ±3 dBm. Figure 67 depicts its input/output characteristic.

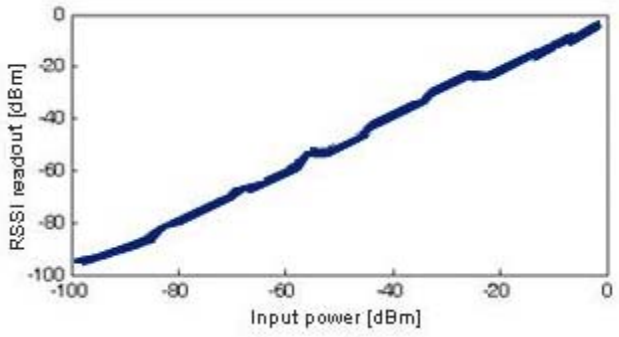

Figure 67: Input/output characteristic of Sun SPOT devices.

The Sun SPOTs possess modest measurement performances due to the hardware fixed 2MHz resolution bandwidth and 1MHz inter-frequency step. On the other hand, Sun SPOTs have solid processing power (32-bit 180MHz processor, 512KB random access memory, 4MB flash memory) enabling them to take about 5500 RSSI samples per second. This makes the Sun SPOT a modest energy detection based MCD. Figure 68 presents a plot of received signal power in the 2.4GHz ISM band for a single Sun SPOT.

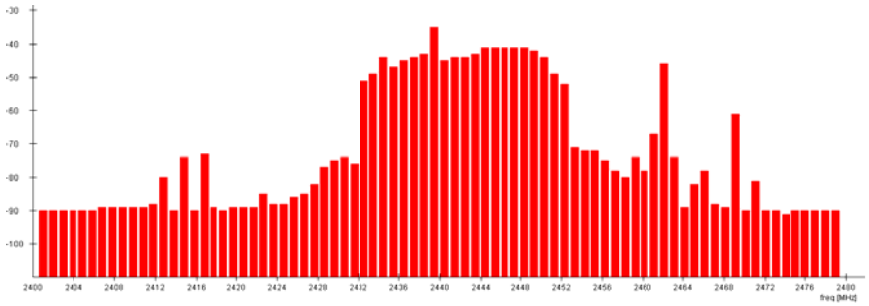

Figure 68: Representation of received signal power in ISM 2400 band by a single Sun SPOT device.

Energy detection performance: The metrics of interest for the performance evaluation of the Sun SPOTs are the same as the ones for the previously elaborated MCDs, also gained empirically from the noise and the signal statistics. As in the case of TI eZ430 RF2500, the averaging cases were also tested for the Sun SPOT device. However, these cases do not give significant performance increase as seen on Figure 69, which depicts the dependence of the probability of detection on the input power for a P_{fa} = 1% threshold. The figure shows that the averaging case offers the best detection performances. The best averaging case is 100 samples in dBm domain, but the detection performance increase is not so significant as in the case of TI eZ430 RF2500. Under this best case the SunSPOTs can detect signals up to -93 dBm for a P_{fa} = 1%.

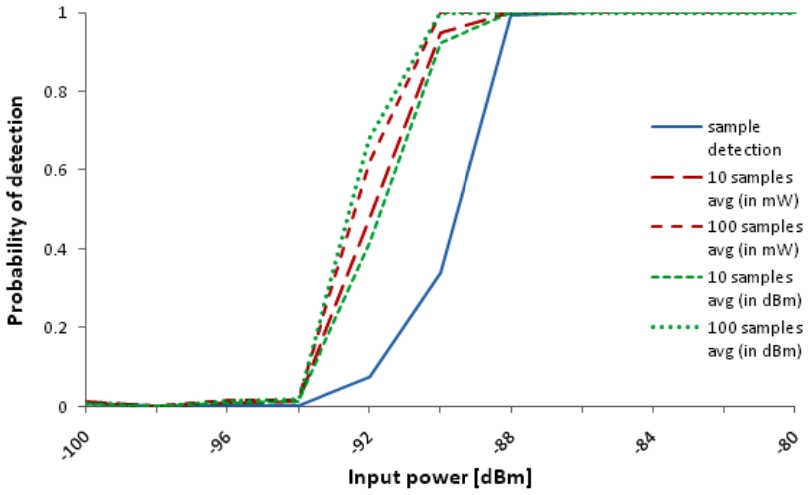

Figure 69: SunSPOT probability of detection dependence on the input signal power for Pfa = 1%, for sample detection and four different averaging cases.

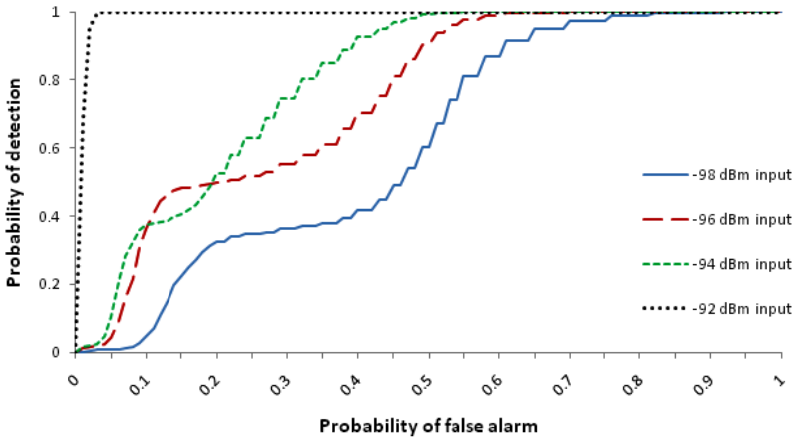

Figure 70: Sun SPOT ROC curves for five different input power levels, and sample averaging of 100 RSSI values in dBm domain.

Figure 70 depicts that signals of -92 dBm and below are reliably detected with the Sun SPOT. The figure also proves that the Sun SPOT has a noise uncertainty problem, resulting in inconsistent ROCs for different input signal powers (the ROCs have slopes in the increase of the probability of detection). The Sun SPOT has less variable noise samples and therefore the averaging case does not offer such a high gain as in the previous MCD case.

3.2.2.4 MCDs comparison

To summarize, Figure 71 plots the ROCs for all inspected devices for an input power of -96 dBm. The low cost devices use their optimized sensing cases, i.e. the Sun SPOT and the TI eZ430 RF2500 perform 100 samples averaging in dBm domain. The optimal detection performances are offered by the TI eZ430 RF2500 spectrum sensor, once again proving that the referred has a high sensitivity when used in a proper manner. However, it should be noted that the USRP2 did not use the highest gain setting, while the other two devices were optimized for sensitivity. Moreover, the TI eZ430 RF2500 used lower bandwidth than the other two. When the USRP2 uses the highest gain setting it would most probably provide the best detection performances. It should be also mentioned that the test scenario employed a constant power signal, which cannot be a usual case in reality.

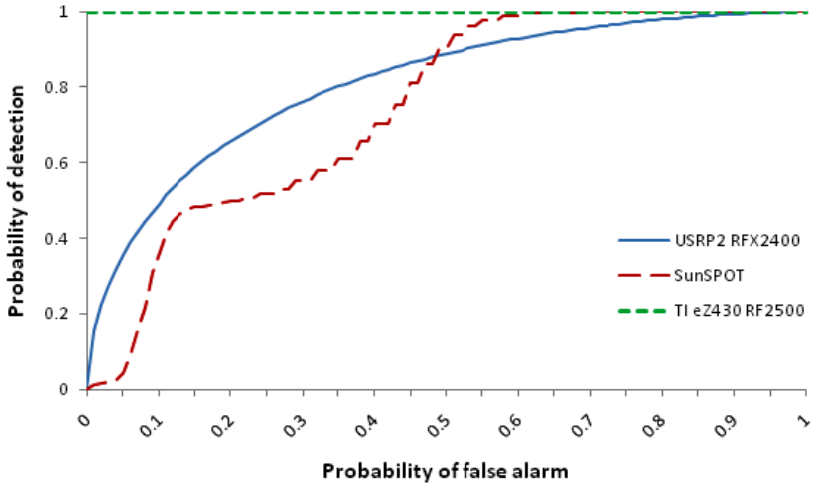

Figure 71: ROC comparison for the different MCDs and input power of -96 dBm. Sun SPOT and TI eZ430 RF2500 perform 100 samples averaging in dBm domain, while USRP2 performs 100 samples averaging in mW domain.

3.2.3 Comparison with spectrum analyzer

3.2.3.1 USRP2 vs. Anritsu MS2690A

The comparison of the results between the Anritsu MS2690A and the USRP2 are based on duty cycle measurements. Figure 72 [15] shows that the USRP2 based sniffer provides duty cycle curves comparable to the spectrum analyzer ones. The used resolution bandwidth (RBW) of the USRP2 is 400 kHz, while the spectrum analyzer has a resolution bandwidth of 300 kHz. Both devices have the same number of measurement points, i.e. 200. The settings are different for both of them as they operate with discrete values for the bandwidth sizes and an existing match for lower bandwidth sizes cannot be clearly identified.

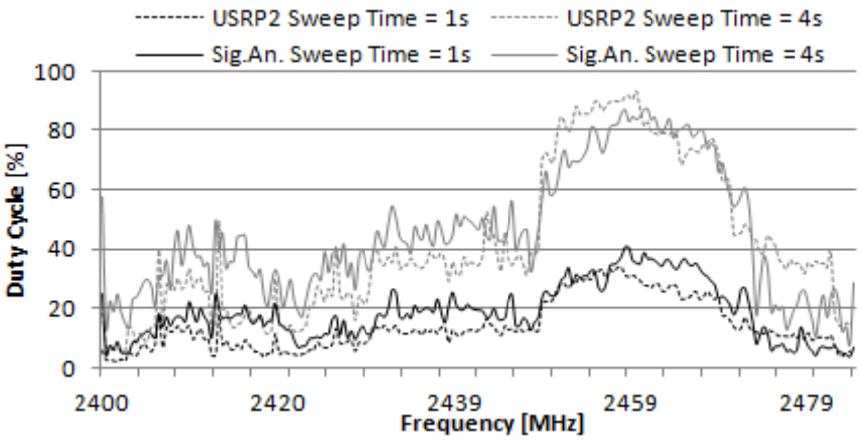

Figure 72: Comparison between USRP2 hardware (RBW = 400KHz) and spectrum analyzer (RBW = 300 KHz)

Figure 72 shows that the USRP2 offers acceptable results for the duty cycle compared to the precision of the spectrum analyzer. However, in the case of lower sweep time (i.e. 1 s) the USRP2 is outperformed since it needs half of the sweep time (0.5 s full, 2 ms actual tune delay) for switching between frequency points and the other half for the actual measurements. This may result in an inability for the USRP2 to handle very low sweep times when using low resolution bandwidths.

This drawback can be overcome with the FFT option of the USRP2 increasing the resolution of the USRP2 device. The performances of the USRP2 using this feature compared to the spectrum analyzer in terms of duty cycle are depicted at Figure 57. The figure shows three curves. The lowest one is spectrum analyzer duty cycle for resolution bandwidth of 100 kHz and sweep time 1s, whereas the other two are the USRP2 based duty cycle curves for resolution bandwidth of 400 kHz and FFT of size 4 and resolution bandwidth of 800 kHz and FFT of size 8, respectively. Both USRP2 settings offer effective resolution bandwidths of 100 kHz and sweep times of 1s. Figure 57 shows that both USRP2 FFT cases outperform the spectrum analyzer in terms of the duty cycle measurement. The reason lies in the lower need of number of points of the USRP2 to cover the full span of 80 MHz. Therefore, the USRP2 spends more time on each point catching more of the transmission. The best duty cycle results are therefore gathered in the case of FFT size 8.

4 Conclusions

In order to increase the radio environmental and spectral awareness of future wireless systems, radio platforms should ideally enable flexible spectrum sensing according to different scenarios, bands and algorithms, within the different nodes of the radio network. With this target in mind, this deliverable presents the architecture and implementation of a reconfigurable engine for multi-purpose spectrum sensing within the power and cost constraints of mobile devices.

The proposed FARAMIR sensing engine builds on a flexible analog front-end and a digital front-end for sensing. The prototyping steps are described in three main phases,

- Phase 1: Standalone analog front-end prototype
- Phase 2: Standalone digital front-end prototype
- Phase 3: Integrated sensing engine prototype

The resulting integrated sensing engine prototype is functionality validated

- By implementing important sensing algorithms (feature and energy-based detection)
- By performing a wideband spectrum sweeping from 500 MHz to 2.5 GHz
- By integrating it in the FARAMIR REM prototype (Deliverable D6.1)

In its current state the prototype has already passed core implementation targets, and laboratory measurements show that it reaches the required sensitivity and sampling rate. The corresponding power measurements also confirm fitness of the proposed architecture and implementation for the energy budget of battery-operated mobile devices. Sensing performance of the proposed spectrum engine prototype is also evaluated in more details and compared with a high-end spectrum analyzer. State-of-the-art measurement-capable devices are finally reviewed and compared, showing performance-cost trade-off. The consortium plans to enter to the next phase on the testing that includes more careful testing of the sensing engine in the more real-life situation for spectrum monitoring in conjunction with field testing of spectrum sensing solutions. In particular we will compare the results on different spectrum use measurements obtained using the FARAMIR spectrum sensing engine against spectrum analyzer based mobile measurement platforms (“Blue Boxes”) of RWTH, and the SDR platform based measurement setups at UKIM.

These results also demonstrate the feasibility of distributed sensing and spectrum mapping approaches based on spectrum inputs collected from various radio nodes, which is a key target in FARAMIR. Measurements also indicate that such distributed sensing techniques will be play an important role in the effort to reach the sensitivity levels envisioned in future wireless systems relying on dynamic spectrum access strategies.

References

[1] Pollin, S et al.; , "Digital and Analog Solution for Low-Power Multi-Band Sensing," New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on , vol., no., pp.1-2, 6-9 April 2010 doi: 10.1109/DYSPAN.2010.5457923

[2] M. Ingels et al., A 5mm2 40nm LP CMOS 0.1-to-3GHz Multistandard

[3] Transceiver, ISSCC 2010.

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

[5] Hou-Shin Chen; Wen Gao; Daut, D.G., "Spectrum Sensing for OFDM Systems Employing Pilot Tones and Application to DVB-T OFDM," Communications, 2008. ICC '08. IEEE International Conference on , vol., no., pp.3421-3426, 19-23 May 2008

[6] 3GPP TS 36.101 V9.0.0. User Equipment (UE) radio transmission and reception.

[7] F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform".

[8] A. Sahai, N. Hoven, and R. Tandra, "Some fundamental limits on cognitive radio," in Fourty-Second Allerton Conference on Communication, Control and Computing, Sept. 2004.

[9] S. L. Wang, Y. Wang, J. Coon, A. Doufexi, "Antenna selection based spectrum sensing for cognitive radio," accept to be published in IEEE Symposium on *Personal, Indoor and Mobile Radio Communications (*PIMRC) 2011

[10] Universal Software Radio Peripheral 2 (USRP2). Information available at: www.ettus.com.

[11] GNU Radio software development toolkit. Information available at: http://gnuradio.org/redmine/wiki/gnuradio.

[12] D. Denkovski, V. Atanasovski and L. Gavrilovska, "Efficient Mid-end Spectrum Sensing Implementation for Cognitive Radio Applications based on USRP2 Devices," COCORA 2011, Budapest, Hungary, April 2011.

[13] Z. Chen, N. Guo and R.C. Qui "Demonstration on Real time Spectrum Sensing for Cognitive Radio", IEEE Communications Letters, 14(10), 2010.

[14] "Measures of Skewness and Kurtosis", Engineering Statistics Handbook, at http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm.

[15] D. Denkovski, M. Pavloski, V. Atanasovski, L. Gavrilovska, "Parameter settings for 2.4GHz ISM spectrum measurements," 3rd International Workshop on Cognitive Radio and Advanced Spectrum Management (CogArt 2010) Rome, Italy, November 2010

[16] Sun™ SPOT Developers Guide., August 2008. Information available at: http://www.sunspotworld.com/Tutorial/index.html.

[17] Oracle Labs Research and Development Center. Information available at: http://labs.oracle.com.

[18] Anritsu MS2690A signal analyzer. Information available at: http://www.anritsu.com/en-gb/products-solutions/products/ms2690a.aspx

[19] M. Buddhikot, "Understanding Dynamic Spectrum Access: Models, Taxonomy and Challenges", Proceedings of IEEE DySPAN 2007, Dublin, Ireland, April 17-21, 2007.

[20] S. Pollin et al., "Digital and analog solution for low-power multi-band sensing", in DySPAN, 2010.

[21] M. Ingels et al., "A 5mm2 40nm LP CMOS 0.1-to-3GHz Multistandard Transceiver", in ISSCC 2010 .

[22] A. Tkachenko et al., "Cyclostationary feature detector experiments using reconfigurable BEE2", in DySPAN, 2007.

[23] J. Lotze et al., "Spectrum sensing on LTE femtocells for GSM spectrum re-farming using Xilinx FPGAs", in SDRforum, 2009.

[24] V. Derudder et al., “A 200Mbps+ 2.1nJ/b digital baseband multi processor system-on-chip for SDRs”, in VLSI Symposium, 2009.
[25] I. Diaz et al., “Single-bit based architecture for OFDM acquisition for multiple standards”, in NORCHIP, 2009.
[26] Hsin-Lei Lin et al., “Implementation of synchronization for 2x2 MIMO WLAN systems”, in ICCE, 2006.
[27] V. Atanasovski et al., "Constructing Radio Environment Maps with Heterogeneous Spectrum Sensors", demonstration paper in IEEE DySPAN 2011, Aachen, Germany, May 2011.
[28] S. Pollin et al., “An Integrated Reconfigurable Engine for Multi-purpose sensing up to 6 GHz”, demonstration paper in IEEE DySPAN 2011, Aachen, Germany, May 2011.
[29] T. Song, J. Park, S. Lee, J. Choi, K. Kim, C. Lee, K. Lim, J. Laskar, "A 122-mW Low-Power Multiresolution Spectrum-Sensing IC With Self-Deactivated Partial Swing Techniques," Circuits and Systems II: Express Briefs, IEEE Transactions on , vol.57, no.3, pp.188-192, March 2010.
[30] T. Mukherjee, G. Fedder, H. Akyol, U. Arslan, J. Brotz, F. Chen, A. Jajoo, C. Lo, A. Oz, D. Ramachandran and V. K. Saraf, “Reconfigurable MEMS-enabled RF Circuits for Spectrum Sensing”, in Government Microcircuit Applications and Critical Technology Conference 2005 (GOMACTech), April 4-7, 2005, Las Vegas, NV.
[31] T.-H. Yu, O. Sekkat, S. Rodriguez-Parera, D. Markovic, D. Cabric, "A Wideband Spectrum-Sensing Processor With Adaptive Detection Threshold and Sensing Time," Circuits and Systems I: Regular Papers, IEEE Transactions on , no.99.
[32] M. McHenry, K. Steadman, A. E. Leu, E. Melick, “XG DSA Radio System”, in Proc. of DySPAN 2008.
[33] CRFS homepage: http://www.crfs.co.uk/