# Cognitive Radio Programming: Existing Solutions and Open Issues

**Mickaël Dardaillon, Kevin Marquet, Tanguy Risset**
*Université de Lyon, Inria,*
*INSA-Lyon, CITI-Inria, F-69621, Villeurbanne, France*
**Jérôme Martin**
*CEA, LETI, Minatec Campus*
*F-38054, Grenoble, France*
**Henri-Pierre Charles**
*CEA, LIST, Minatec Campus*
*F-38054, Grenoble, France*

## ABSTRACT
Based on our analysis, the success of cognitive radio heavily depends on Software Defined Radio (SDR). The cost, performance and power consumption of SDR hardware platforms will enable (or forbid) smart radio applications and cognitive radio networks. SDR has evolved rapidly and is now reaching market maturity, but many issues have yet to be studied. In this chapter, we highlight how hardware architectures fulfill the constraints imposed by recent radio protocols and we present current architectures and solutions for programming SDR. We also list the challenges to overcome in order to program future cognitive radio systems.

## 1 INTRODUCTION
Until now, radio technologies have been developed in a static paradigm: protocols, radio resources allocation and access network architectures were defined beforehand, allowing implementation of effective yet non-adaptable radio systems. Nowadays, the saturation of radio frequency bands calls for a new era of radio ssytems that will be characterized by self-adaptive mechanisms. These mechanisms will rely on *software radio technologies*.
J. Mitola has coined the concept of software radio in his seminal work during the early 90's (Mitola, 1992). While implementing the whole radio in software is still a utopia, many architectures now hitting the market include some degree of programmability.
With the emergence of SDR, many questions related to the software layer of a software radio machine arise. How will this kind of platform be programmed? How can we write radio transceiver programs that are portable from one terminal to another? To answer these questions, programmers have to know how the architectural characteristics of SDR systems can be abstracted to provide portable code. Unfortunately, there is no agreement on the hardware architecture embedded in a cellular phone with SDR facilities. Various technologies are used: Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP), General Purpose Processor (GPP), etc. These technologies are often mixed together and sometimes the term configurable is more adequate than programmable for them.
Studying architectures, programming environments and programming models for emerging SDR systems simultaneously is of crucial importance because of the need to define the hardware abstraction layer of SDR systems: the *radio hardware abstraction layer* (R-HAL).

In this chapter, we provide an up-to-date review of existing SDR hardware platforms, classifying them into five categories. Programming models and programming tools used in these platforms are not yet mature, most of these platforms being currently programmed using *ad-hoc* techniques. No common language, format or API has yet emerged; hence it is impossible to compare precisely the performance of the different approaches. All performance results presented here are taken from the existing works.
We illustrate, with LTE as an example, the problems of modern digital physical layer protocols: fast terminal reconfiguration, data-dependent data flow. We also give an insight on what should be used as programming model for the programming of SDR platforms.
The rest of the chapter is organized as follows: we first provide a brief summary of radio, SDR and cognitive radio technologies, we also present an example extracted from the LTE protocol that illustrates the difficulties of SDR programming. Next, we present a survey of hardware SDR platforms, categorize and provide synthetic performance comparisons between them including power consumption when available. We then focus on programming environments for SDR and more precisely on the problem of defining a language for describing waveforms, i.e. the radio's physical layer. We finally review the remaining most important open problem: defining a programming model for SDR and a hardware abstraction layer for SDR.

# 2 COGNITIVE RADIO

## 2.1 Software Defined Radio

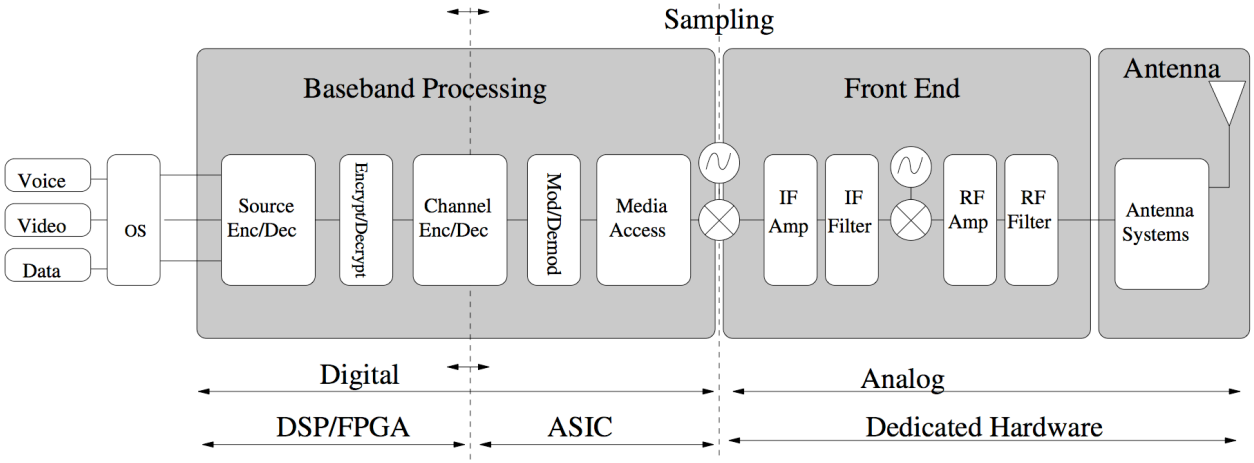


*Figure 1. Radio block diagram, highlighting separation between digital and analog parts, as well as programmable, configurable and dedicated hardware parts.*

The different components of a radio system are illustrated in Figure 1. Clearly, all of the digital components may not be programmable, but the larger the programmable part (DSP/FPGA part on Figure 1), the more *software* the radio is. Dedicated circuits are usually needed, for which the term *configurable* is more adapted than *programmable*. In a typical SDR, the analog part is limited to two frequency translations: from radio frequency to intermediate band, and from intermediate band to baseband. The baseband is sampled, and all the signal processing is done digitally.
To encourage a common meaning for the term "SDR", the SDR Forum (recently renamed Wireless Innovation Forum) proposes to classify it into five tiers. Tier 0 corresponds to hardware radio; Tier 1 corresponds to software controlled radio with control functions implemented in software; Tier 2 corresponds to software defined radio with digital baseband processing implemented in software; Tier 3 is the ideal software radio with sampling at the antenna to process radio frequency signal in software; Tier 4 corresponds to ultimate software radio and extend these capacities with fast transition (millisecond) from one protocol to another. Tier 3 is the most popular definition of SDR: the radio includes software control of modulation, bandwidth, frequency range and frequency bands. Tier 3 and 4 are not realistic today.

Building an SDR implies on choosing a computing platform for the digital part, a sampling frequency and a radio front-end. In addition to the careful choice of a computing platform, the designer must make a trade-off between sampling frequency and computing complexity. For instance, sampling a signal at 4.9 GHz (hence with a sample rate greater or equal to 10 GHz) is not available today with reasonable power consumption. Even after the ADC evolve to low power, a high bandwidth ADC would produce a very high sample rate; therefore the front-end characteristics (bandwidth, ADC resolution, etc.) constrain the digital part in terms of computing power. In this chapter, we focus on the digital part represented on the left side of Figure 1, assuming an adequate front-end is available for the platform.
The hardware platforms we review in the following are considered from a programmer's point of view. They target the implementation of wireless communication protocol stacks from application down to physical layer (including baseband processing and intermediate frequency conversion), for emission (TX) and/or reception (RX).

## 2.2 Cognitive Radio

A *Cognitive Radio* is a wireless communication system that can sense the air medium, and decide to configure itself in a given mode. Tier 2 SDR platforms are natural candidates for cognitive radio implementation but cognitive radios do not have to be SDR.
The main feature enabled by spectrum sensing ability is called *dynamic spectrum management*: the system is able to configure radio-system parameters in an autonomous manner. These radio-system parameters include transmission power, frequency band, modulation, channel and source coding, but might as well include higher level parameters such as waveform (physical layer protocol), MAC protocol, routing protocol and other networking characteristics. In that case, the term "autonomous" means "without human decision", i.e. automatic. However in many cases the decision cannot be taken independently of neighbouring communicating devices implied in the communication. This leads to the scientific field of *distributed algorithms for radio resource allocation*.
Distributed algorithms are used when the decision of choosing a coding scheme or a frequency band has to be shared by many radio terminals. This perspective opens many new research problems and many new applications at the same time. For instance, distributed algorithms can be used to optimize interference cancellation globally, hence optimizing power consumption. Another example is the use of *relay*, i.e. transmission of packets from neighbour to neighbour according to routing decisions done at the physical layer, as opposed to routing decisions taken at a higher level in the protocol stack. Relay can be used for reducing transmission power or to improve quality of transmission using network coding techniques.
From the research point of view, distributed algorithms open new fields: complexity and optimality of distributed solution to dynamic spectrum management. In point to point communication, OFDM techniques are approaching theoretical optimal bound for spectrum efficiency, that is the information rate that can be transmitted over a given bandwidth. But if cooperation between terminals is allowed, theoretical bounds are much more difficult to compute and technologically there is place for large improvements in communication systems that use cognition, cooperation and distributed decision algorithms. These problems are tackled in a new scientific field named *network information theory*.

## 2.3 Connection between SDR and cognitive radio

A cognitive radio is not necessarily based on SDR architecture. As an example, today's mobile phones are able to switch autonomously between local Wifi connection and 3G to communicate. However this switch is not instantaneous, it needs to go through the entire software stack up to the operating system level. Also, today's mobile phone include a dedicated chip for each wireless protocol even if they share the same frequency band such as Wifi and Bluetooth for instance. SDR technology enables both evolutions: a dynamic switching between two protocols and a common hardware for all wireless protocol.
One could possibly imagine that cognitive radio will be implemented in pure software in a far future, providing the highest flexibility. Unfortunately, this is not possible today, and will not be in the near future because the complexity of the communication protocols increases faster than Moore's law (UMTS

Forum, 2011). Even if SDR is not *necessary* for cognitive radio technology, it will become *mandatory* because of the rapid appearance of new protocols and because of the constant need to lower the hardware platform prices. We give here the three main reasons that make cognitive radio highly dependent on SDR hardware technology.

- Fast reconfigurability. When dynamic spectrum sensing is used, in the primary-secondary user scenario for instance, the terminal should be able to switch very quickly (in a few micro-seconds) from one protocol to another, otherwise communication packets will be lost, leading to costly retransmission, impossible in many streaming applications. This fast reconfigurability can only be achieved if the hardware has been dedicated to that; it is not possible to do it in a pure software way.
- High performance and power consumption. Even if not all protocols require high performance, some of them do, mainly those relying on MIMO OFDM protocols. They need several GFlops operations, which cannot be achieved by pure software except at a power consumption of hundreds of watts using grid computing. Hence dedicated hardware is needed to have both high performance and low power consumption for future wireless terminals.
- Unitary cost and adaptability. Having one chip per protocol does not scale, the price increases with the number of chips needed and new phones need to be compatible with old protocols. Moreover, cognitive radio terminals need to be adaptable to new arriving protocols. For instance, in the white space scenario, new protocols might appear very frequently and the mobile will configure itself to connect to the emerging protocols. This is only possible using SDR technology.

Given the above reasons, it is clear that SDR technology will be necessary to realize cognitive radio application. However, the SDR technology is not stable yet, it is still in a very active development phase, from the hardware point of view as well as from the software point of view. In this chapter, we study how current SDR technology, hardware and software, enable performance, fast reconfiguration and low power consumption for wireless mobile devices.

## 2.4 Example of of high performance and fast reconfiguration needs

In this section we will show why recent radio communication protocols require a specific attention from a programmer's perspective. These protocols introduce harder real-time and dynamicity constraints that make usual computation models inefficient. Historically, structured programming led to imperative programming followed by fruitful evolutions for general purpose programming: object-oriented programming, functional languages, threads etc. Simultaneously, domain-specific programming models have been adopted in many fields, the most well known being reactive programming model for real-time control, and dataflow programming model for signal processing.

Dataflow programming model has been popularized by the *dataflow domain* of Ptolemy (Eker, 2003), implementing Khan's process networks (Kahn, 1974). Dataflow programming assumes that the flow of data is statically known and that the executed computation does not depend on data values. This condition has been verified for fifty years of signal processing but is not satisfied anymore by, for instance, LTE protocol. A more complete analysis of the existing computation models for SDR is given in the cognitive radio programming framework survey, below we simply give an example of the practical problems encountered when programming LTE on an SDR platform.

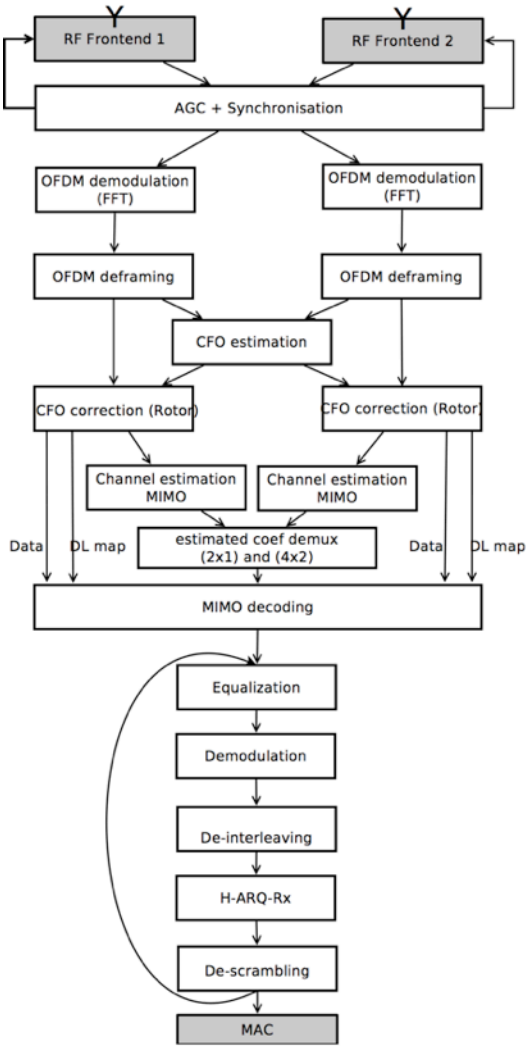


*Figure 2.LTE pipeline flow, CFO correction and PDCCH decoding require specific attention.*

LTE is a mobile communication standard, developed by the 3GPP (3rd Generation Partnership Project) and approved into ITU (International Telecommunications Union). Figure 2 represents the global flow for decoding a LTE frame (release 8, mode 5), from the mobile equipment point of view. A LTE frame is composed of 10 sub-frames of 1ms each, each sub-frame being composed of 14 symbols in time and 2048 sub-carriers in frequency for a 20 MHz bandwidth. The transmission uses MIMO technology (Multiple-Input and Multiple-Output), with up to 4 antennas for transmission on the base station and 2 antennas for reception on the mobile equipment.

In this flow, we pay attention to the carrier frequency offset (CFO) correction. This correction is performed by analyzing specific resource elements of the frame called reference signals or *pilots*. Pilots are *within* the sub-frame that is corrected, hence the estimation of the CFO (CFO-estimation on Figure 2) must be performed very quickly, and, more challenging, the carrier frequency offset correction (CFO-correction on Figure 2) must be configured using the result of the CFO estimation. Hence this reconfiguration is dynamic and real-time, it should occur in less than 100µs (10% of the computation time for a sub-frame). This is the first problem that SDR platform designers (and programmers as well) encounter: how to design a system that can reconfigure so quickly?

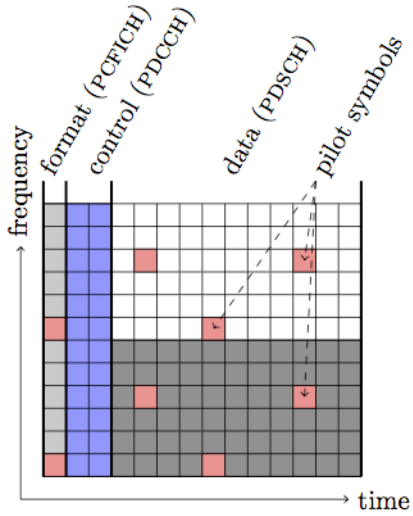


*Figure 3. Resource allocation in an LTE PRB (Physical Resource Block).*

The second problem occurs after the FFT and MIMO decoding have brought samples in the frequency domain. The sub-frame is then composed of a matrix of symbols, not all of them being addressed to a given user, because LTE encodes several users in the same sub-frame. Extracting the symbols for a given user requires decoding the Physical Downlink Control CHannel (PDCCH). However, PDCCH format is encoded within the sub-frame itself in the Physical Control Format Indicator CHannel (PCFICH) (see Figure 3), which must therefore be decoded beforehand. Depending on the PDCCH a certain number of symbols in the PDSCH (Physical Downlink Shared CHannel) will have to be sent to the rest of the flow (for demodulation etc.). Moreover, the LTE standard states that the modulation scheme (QPSK or 16QAM for instance) should also be encoded in the PDCCH. Figure 3 illustrates the symbol matrix in the frame, the successive decoding of PCFICH, PDCCH and PDSCH are illustrated in Figure 2 by the loop back arrow after de-scrambling.
This is the second problem that we highlight which definitely cannot be expressed in a static dataflow programming model: the number of data to be transmitted, as well as the computation to perform on each piece of data, are dependent on the data themselves. This is one of the main motivations of the work presented here: how to express such a computation in a language that is generic enough to be compiled on various SDR platforms?

## 3 SURVEY OF HARDWARE PLATFORMS FOR SDR

In order to classify the SDR platforms, we need to define objective criteria. Trying to define criteria based on used technology can be tricky, as most platforms are heterogeneous. Moreover, the technology used may not be a relevant criterion for platform users. The user will mainly be interested in the three following features: *programmability* and *computing power*, which will condition the supported protocols, and *energy consumption* which we believe will be the limiting factor for technology adoption. Choosing a computing platform for a given application is a trade-off between these features.
However, from the programmer point of view, the architecture is of major importance because it will have a crucial impact on programming models and tools used on the platform. We end up with five categories of SDR architectures:

1. General-purpose CPU approach
2. Co-processor approach
3. Multi-processor approach
4. Configurable units approach
5. Programmable blocks approach

Each approach is described in its corresponding subsection, and examples of existing implementations are given.

## 3.1 General-purpose CPU approach

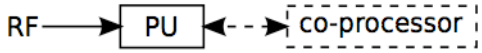


*Figure 4.General-purpose CPU approach with optional co-processor.*

The general-purpose CPU approach (depicted in part of Figure 4 not dashed) uses a general purpose computer processor to provide a computing platform. It offers a flexible and easy way to program the platform, but with a high energy consumption for a performance objective.

With the CMOS technology continuous evolution, one could imagine that future computers will be able to compute all protocols in real-time. However, the increase in data throughput is higher than the increase in computing power. Therefore, this kind of architecture will only be able to support past protocols, unless it can make use of higher parallelism.

The Universal Software Radio Peripheral (USRP, 2014) is representative of the General-purpose CPU approach. It is composed of high frequency ADC/DAC that sample the signal in intermediate frequency. A FPGA converts and stores baseband signal. Most of the signal processing is done by a CPU connected to the FPGA by a USB link (USRP1) or an Ethernet link (USRP2). The platform is widespread and supported by third party software. It is aimed to work with GNU radio, but is also compatible with National Instruments' LabView and Mathworks' Matlab.

Recently, Microsoft developed SORA (Tan, 2011). This platform is connected to the computer by a PCIe bus, which permits low latency and high throughput data transmission. It makes extensive use of modern CPU features to perform 802.11b/g processing in real-time.

## 3.2 Co-processor approach

In order to accelerate the signal processing, optimizations of the general-purpose CPU approach have been explored recently. As depicted in Figure 4, they rely on the addition of a co-processor to perform heavy processing. It reduces the price to pay in terms of energy while keeping high programmability and flexibility.

The work presented in (Horrein, 2011) uses a GPU as a co-processor in a GNU radio flow. It permits gains of a factor 3 to 4 in processing speed.

The Kansas University Agile Radio (KUAR) (Minden, 2007) uses an embedded PC associated to a FPGA. The choice of the model of computation is left to the programmer, ranging from a full VHDL implementation to a full processor implementation close to the GNU radio flow.

Other developments use generic DSP as central processor, which provides higher efficiency while keeping high programmability.

Texas instruments offer a three-core DSP with specialized symbol and chip rate accelerators. This product provides programming flexibility for WCDMA base cells, with support for up to 64 users and different protocols (Agarwala, 2007).

The ADRES (Architecture for Dynamically Reconfigurable Embedded Systems) (Bougard, 2008) developed by Imec is a coarse-grain reconfigurable architecture. It is built around a main CPU and the ADRES accelerator. The ADRES is seen by the processor as a VLIW co-processor, while being an array of 16 functional units. Each one is an SIMD processor, which leverages data parallelism. The processor is programmed using the DRESC compiler (Mei, 2002), in ANSI C. The DRESC compiler generates code to unroll loops and compute them using the ADRES accelerator. It targets telecommunications with benchmarks on 802.11n up to 108 Mbps and LTE up to 18 Mbps, and an average consumption of 333mW (Bougard, 2008).

## 3.3 Multi-processor approach

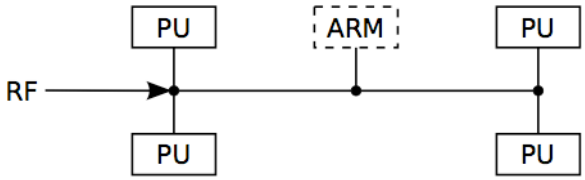


*Figure 5. Multi-processor approach with optional central control processor.*

Previous architectures offer only limited task parallelism. The next categories fill this gap using tailored architectures with heterogeneous types of processors. One approach to get efficient and specialized platforms is to use dedicated processors. In this approach, dedicated processors are used to compute signal processing. Both central and distributed controls are considered in this section.
The multi-processor approach has a high programmability, but the flexibility of the platform is reduced by its specific architecture. The architecture concept is depicted on Figure 5.
The NXP EVP16 (Berkel, 2005), presented in 2005, is composed of several computing units. An ARM processor provides control and LINK/MAC layers. A conventional DSP, a vector processor and several hardware accelerators are used for signal processing. The vector processor is built as a vectorized pipeline and addressed as a VLIW. It performs UMTS for a 640 kbps throughput at 35 MHz, with a maximum of 300 MHz (Berkel, 2005).
Infineon built the MuSIC (Ramacher, 2007) as a multi-DSP solution for SDR. The control is done by an ARM processor. Signal computation is processed by 4 SIMD DSP and dedicated processors for filtering and channel encoding. Power consumption in WCDMA mode is 382 mW for the worst case and 280 mW for a typical case. This chip is provided as a commercial solution under the name X-GOLD SDR 20 by Infineon (Ramacher, 2011). It is programmed using a mix of C code and assembly code for critical processing.
The Sandblaster architecture (Schulte, 2004) is built around 3 entities: the fetch and branch unit, the integer and load/store unit, and the SIMD vector unit. Task parallelism is managed by a Token Triggered Threading component, which provides hardware support for multithreading. On the SB3011 (Glossner, 2007), 4 sandblaster cores are integrated and controlled by an ARM processor. It is programmed in ANSI C with a dedicated compiler. Maximum consumption is 171 mW for WCDMA at 384 kbps (Glossner, 2007).
The University of Michigan at Ann Arbor developed the SODA (Woh, 2006) SDR platform, and its prototype version ARDBEG (Woh, 2008). SODA was developed as a complete software SDR solution. It consists of an ARM for control and 4 SIMD DSPs for signal processing. ARDBEG builds on that platform by adding a hardware turbo decoder and optimizing DSPs for signal processing. All programming is made using C code. Consumption results on ARDBEG for WCDMA and 802.11a are under 500 mW (Woh, 2008).
The University of Dresden, Germany, developed the Tomahawk SDR chip (Limberg, 2008), aiming at LTE and WiMAX. It uses two Tensilica RISC processors for control, six vector DSPs and two scalar DSPs for signal processing, as well as ASIC accelerators for filtering and decoding. The scheduling is done by dedicated hardware and C code is used for programming. No protocol has been implemented yet on this platform. From the authors' estimation, the platform consumption is about 1.5 W (Limber, 2008).
Picochip (Pulley, 2003) approaches signal processing using many small cores. These cores are mapped on a deterministic matrix. The company provides a C-based development tool flow. No benchmark is provided for this chip. However, the company is announcing OFDM and 4G base stations as reference applications on its website.
The University of California at Davis developed the Asynchronous Array of Simple Processors (Truong, 2009). This project aims at providing signal processing computation using small processors. All processors can communicate with their nearest neighbours, in a grid-like array. Version 2 adds hardware

accelerators for FFT, Viterbi and video motion estimation, while increasing the total number of cores to 167. Complete 802.11a/g is processed at 54 Mbps using 198 mW (Truong, 2009).

## 3.4 Configurable units approach

In order to offer lower energy consumption, some platforms substitute DSP for configurable units. The difference between specialized DSPs and configurable units is very thin: a DSP is able to process any computation, whereas a configurable unit is too specialized to do so. This implies a big difference in term of programmability: to gain more performance, the DSP flexibility is abandoned in favor of configurable units. This leads to platforms which are much more difficult to program.

Fujitsu developed the SDR LSI (Saito, 2006) in 2005. The platform makes extensive use of hardware accelerators, associated to reconfigurable processors. All these components are connected to a crossbar data network, and controlled by a central ARM processor. The chip was able to run 802.11a/b with a maximum throughput of 43 Mbps (Saito, 2006).

The BEAR SDR platform (Derudder, 2009) is the evolution of the ADRES from Imec. It is constituted of an ARM processor for control and three ASIPs for coarse time synchronization on different front-ends. Two ADRES coarse-grain configurable architectures are used for baseband processing with a Viterbi accelerator. The platform can be programmed with C or Matlab code, using the Imec development chain. In terms of energy consumption, BEAR achieves 2×2 MIMO OFDM at 108 Mbps for 231 mW (Derudder, 2009). Imec is licensing the BEAR platform as an IP block.

The Magali SDR chip (Clermidy, 2009) is developed by the CEA-Leti as a telecommunication demonstration platform. It is built on a Network-on-Chip, each peripheral having an access to the network, with an ARM processor controlling configurations. Computation is done by coarse-grain reconfigurable cores called Mephisto and reconfigurable IPs for OFDM, decoding and deinterleaving. Smart memory engines (Martin, 2009) are distributed on the Network-on-Chip and act like DMAs, while also providing data rearrangement capabilities. The chip performs 4×2 MIMO LTE reception in the most demanding scenario with a consumption of 236 mW (Jallier, 2010).

CEA-Leti Genepy (Jallier, 2010) is using a larger granularity for its distributed approach. It is based on Magali (Clermidy, 2009) technology, using the Network-on-Chip and the coarse-grain configurable cores. The control carried out by the ARM processor is undertaken by distributed small RISC processors. Each cell on the network is composed of two Mephisto cores, one Smart Memory Engine and a RISC controller. The platform is purely homogeneous, with no hardware accelerators. In terms of computing power, 4×2 MIMO LTE reception is processed with a total consumption of 192 mW (Jallier, 2010).

The ExpressMIMO is developed as a configurable units approach on a FPGA by EURECOM (Nussbaum, 2009). All the configurable units share a common network interface, DMA engine and microcontroller, and each has a specific configurable IP for data processing. The board targets OFDM MIMO implementations and uses the OpenAirInterface open-source framework (Open Air Interface, 2014). A more recent implementation should be available soon (Schmidt-Knorreck, 2012).

University of Twente, Netherlands, developed the Annabelle SDR chip. It is also built on a Network-on-Chip, using coarse-grain reconfigurable cores. An ARM processor is used for control, and accelerator modules (Viterbi, etc.) are connected to the ARM through an AMBA bus. Only OFDM specific benchmarks have been published at the time of submission (Zhang, 2009).

## 3.5 Programmable blocks approach

The last approach uses programmables blocks and is mainly constituted of FPGAs. It doesn't provide programmability as it is, but great flexibility to create tailored architectures. Programmable blocks offer high computing power for moderate energy consumption.

The XiSystem (Lodi, 2006) is a VLIW architecture featuring 3 concurrent datapaths, including a PiCoGA (Pipelined Configurable Gate Array). The PiCoGA is an oriented datapath FPGA which executes specific instructions for the processor at run-time. The development is made with C to provide code for both the

VLIW and the PiCoGA. It is aimed at embedded signal processing in general, with a benchmark on MPEG2 encoding and an average consumption of 300 mW (Lodi, 2006).
The Rice University has developed WARP (WARP, 2014), an open SDR platform. A Xilinx Virtex FPGA does the computation. Programming uses VHDL language. An open source community is led by the Rice University to offer open source implementations on the platform. For instance, it contains a MIMO OFDM Reference Design that can be extended based on Xilinx XPS tool.
WINC2R is an original platform for SDR developed by the Rutgers University. The platform is built on a FPGA, with softcore processors and accelerators. Softcore processors can be programmed with GNU radio. Computation flow can be balanced on processors or accelerators, depending on the constraints. Moreover, by using an FPGA, accelerators can be chosen and tuned during development. 802.11a has been implemented on the platform (Satarkar, 2009).
The Nutaq company (Nutaq, 2014) offers development tools and platforms for SDR based on FPGA. Development is done using Simulink model-based approach or in VHDL. The platform is presented as supporting MIMO WiMAX. Many other companies offer similar products based on FPGA (Pentek, 2014; Sundance, 2014).

## 3.6 Analysis

| Platforms | Availability | Application | Prog. | Cons. |
|---|---|---|---|---|
| (USRP, 2014) | commercial | N/A | C++ | ~PC |
| TI C64+ (Agarwala, 2007) | commercial | base station | C/ASM | 6000 mW |
| X-GOLD SDR (Ramacher, 2011) | commercial | WCDMA | C/ASM | ≤ 382 mW |
| Sandblaster (Glossner, 2007) | IP licence | WCDMA | C | 171 mW |
| ARDBEG (Woh, 2008) | prototype | WCDMA | C | ≤ 500 mW |
| BEAR (Derudder, 2009) | OP licence | MIMO OFDM | matlab/C | 231 mW |
| Magali (Clermidy, 2009) | prototype | MIMO OFDM | C/ASM | 236 mW |
| ExpressMIMO (Nussbaum, 2009) | prototype | MIMO OFDM | C | N/A |
| (WARP, 2014) | commercial | MIMO OFDM | VHDL | N/A |
| (Nutaq, 2014) | commercial | N/A | matlab/VHDL | N/A |
| ASAP (Truong, 2009) | prototype | 802.11 a/g | N/A | 198 mW |
| Genepy (Jalier, 2010) | prototype | MIMO OFDM | C/ASM | 192 mW |

*Table 1. Comparison of key SDR platforms based on the published performance results.*

In order to better understand each category, we summarize the main characteristics for key platforms that use different approaches in Table 1. Energy consumption is not defined for FPGA-based platforms because it is heavily dependent on the configuration. Based on these key platforms, we draw trends on the application fields of each category.
If you don't want to study energy consumption nor architecture algorithm adequacy, the general-purpose CPU approach is the easiest way to go. However, if you intend to study energy consumption or computing power impact, this approach is not recommended. Indeed, dedicated hardware platforms have very different behaviours compared to generic processors. This makes it difficult to establish a relationship between computing power and energy consumption for the generic approach and others. As an example, for a given protocol, computing requirements in terms of number of operations per second may vary with a factor of 100 in the literature, depending on the architecture granularity.
In order to study computing power and to have the lowest energy consumption, a heterogeneous approach, which exploits hardware acceleration, is a better starting point. In this family, using DSPs as in Imec's solution (Derudder, 2009) or configurable blocks as in Magali (Clermidy, 2009) seems a pragmatic and efficient approach, these platforms being dedicated, and hence optimized, for SDR.

Unfortunately, using such a solution makes you heavily dependent on the platform architecture, and porting a waveform to a different architecture can be tricky. Providing a common HAL is a real challenging but promising way to develop practical multi-platform SDR.
Alternatively, the programmable blocks approach provides a flexible and efficient platform for prototyping thanks to the large adoption of FPGA technology. It can be versatile in the architecture choice, see the radically different approaches from (WARP, 2014) and (Satarkar, 2009) for example.
The most obvious conclusion from this SDR architecture survey is that no common architecture model could be extracted to provide, as it is the case for the general purpose *processor*, a *hardware abstraction layer* that could be used to help programming cognitive radio applications. We are now going to study the efforts that have been made to provide a programming environment adapted to cognitive radio.

# 4 COGNITIVE RADIO PROGRAMMING FRAMEWORK SURVEY

As we have seen before, there have been a lot of efforts to set up dedicated SDR hardware. From these works, we can conclude that i) hardware support is necessary to match performances and low-power requirements of modern radio protocols and ii) it is not feasible to write traditional C/ASM code and map it manually anymore.
Programming and executing waveforms is clearly an application scope of the general problem of programming parallel machines, and this has to be taken into account when programming an SDR hardware. We now review research efforts that have been made to program SDR platforms efficiently. We first give an insight of some programming environments used to program *more than one* SDR architecture. Then we focus on one central aspect of radio programming: *waveform programming*. Expressing a waveform, i.e. the physical part of the radio communication protocol, in a high-level language is a challenge. We classify the radio programming environments according to the programming model they use to express waveforms in the following sections.

## 4.1 Cognitive Radio programming environment

There are two distinct important issues to address in the programming environment. The first one is the programming model used to specify the waveform (described in the following section), and the second is the global programming framework that will enable this programming model to be efficiently implemented on most of the platforms mentioned in previous section. Choosing the right programming framework is not a simple matter of comparing objectively pros and cons, it highly depends on strategic choices in companies and cultural acceptance by programmers.
The Software Communication Architecture (SCA) applicative framework was launched by the US department of defense within the Joint Tactical Radio System project (JTRS). It is an example of *top-down designed* framework. SCA re-uses major technologies coming from distributed software programming such as CORBA (Common Object Request Broker Architecture) for instance. The SCA has been implemented in OSSIE (González, 2009) and in military devices too. The OSSIE set of tools of the SCA framework is an initiative from the American department of defense intending to provide a graphical environment for rapid prototyping of waveforms. It allows connection of components and generation of the corresponding code. However, the SCA framework is probably doomed to failure as the department of defense cancelled the project after it failed the Army's Network Integrated Environment testing.
Relying on the success of open-source software development, the GNUradio project (GNU radio) proposes to implement software defined radio systems using a library of signal processing blocks written in C++ for performance-critical parts, with Python programming language to interface these blocks. Initially dedicated to the Universal Software Radio Peripheral (USRP, 2014) SDR hardware from Ettus Research, it recently received attention from many other hardware providers. However, this approach is currently implemented in general purpose CPU platforms and will encounter timing problems when complex MIMO OFDM protocols will have to be implemented. Some implementations, as for instance the OpenAir Interface (Open air interface, 2014), use real-time OS such as RT-Linux to improve the quality of real-time signal processing handling.

Many dedicated environments are based on a graphical interface coupled with dedicated IPs, as for instance Simulink coupled with Mathworks tools to program FPGAs or LabView. Recent trends based on OpenMP or OpenCL are emerging (Wang, 2007), but have not gain enough attention yet.

## 4.2 Imperative concurrent waveform programming

For an embedded software programmer, the easiest way to program an SDR platform is to use an imperative language (generally C language) associated with threads to express parallelism. It has been used to program waveforms for both heterogeneous and homogeneous parallel platforms. For instance, the different units of the BEAR SDR platform (Derudder, 2009) are programmed using C and Matlab code.

The efficient programming and execution of waveforms is tightly coupled with advances in the programming techniques for heterogeneous platforms. Although not yet evaluated for waveform programming, the ExoCHI (Wang, 2007) programming environment and the Merge (Linderman, 2007) framework (based on ExoCHI) are proposals aiming at easing the programming of heterogeneous platforms while achieving good performances. The proposed solution is to extend OpenMP with intrinsic functions and dynamically map the software on available resources.

Cohen *et al.* (Cohen, 2010) propose a similar approach in which programs are compiled into a specific bytecode and then compiled dynamically to the different accelerators available on the platform. This approach has not been evaluated on SDR platforms yet.

Many isolated works concentrate on the use of hardware accelerators. The Dresc (Mei, 2002) compiler allows unrolling loops in order to execute parallelized code on a specific accelerator made of 64 functional units.

The integration of the GPU in a SDR programming model has also been studied. Horrein *et al.* compare (Horrein, 2011) different system architectures for using the GPU for SDR programming. Their work is based on OpenCL and GNU radio (GNU radio).

## 4.3 Dataflow waveform programming

Numerous research works present arguments in favor of a paradigm shift and propose to program waveforms using dataflow languages. These languages rely on a *Model of Computation* (MoC) where a program is represented as a directed graph $G = (V, E)$. An actor $v \in V$ represents a computational module or a hierarchically nested subgraph. A directed edge $e \in E$ represents a FIFO buffer from its source actor $S$ to its destination actor $D$. Dataflow graphs follow a data-driven execution: an actor v can be executed (fired) only when enough data samples are available on its input edges. When firing, v consumes a certain amount of samples from its input edges and produces a certain number of samples on its output edges.

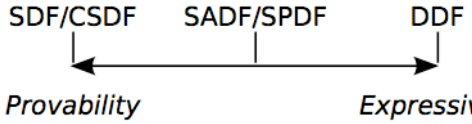


*Figure 6. Representation of the balance between provability and expressivity in dataflow computation models.*

Many dataflow-compliant programming models have been proposed for specific applications; they are illustrated in Figure 6. *Synchronous DataFlow* (SDF) (Lee, 1987) means that the number of tokens necessary for an actor to fire is known at compile-time. In this case, static scheduling of actors can be performed and the size of the buffers between actors can be bounded. In *Dynamic DataFlow* (Buck, 1994), data samples consumed and produced by an actor at each firing can vary dynamically at runtime, and can even be 0 in order to provide more flexibility for programming. As a drawback, theoretical analysis capabilities are reduced. Between synchronous and dynamic dataflow formalisms, a wide amount of models have been proposed, e.g. *Cyclo- Static Dataflow* (CSDF) (Bilsen, 1996), *Scenario-Aware*

*DataFlow* (SADF) (Stuijk, 2011), *Schedulable Parametric DataFlow* (SPDF) (Fradet, 2012). The goal was to look for a trade-off between the ability to statically analyze programs and the expressivity of the languages. For instance, using SDF to model a LTE waveform will lead to over-estimate the necessary resources at runtime because dynamic behaviour will not be captured.

StreamIt (Thies, 2009) is a programming language that allows to describe programs in an SDF manner, through the use of *filters* and *split-join* operators. It comes with tools able to perform static analyses and optimization's of the dataflow graph. The compiler can generate C code for threads, that the programmer has to map manually on the available hardware resources. The underlying CSDF MoC is restricted to a single flow, which makes StreamIt not usable for complex and dynamic waveforms such as LTE.

ΣC (Goubier, 2009) is a proposal to program waveforms using an extension of C. The corresponding MoC is more expressive than SDF thanks to non-deterministic extensions but still allows some static analyses to be performed such as bounding memory usage. However it does not allow dynamic behavior of actors, which is a limiting approach when attempting to describe waveforms such as LTE. The experimental platform used for ΣC is a many-core processor.

Past works have demonstrated the interest of programming using a general purpose language augmented with some primitives that allow to build the dataflow graph. Following the *Stream Virtual Machine* (Labonte, 2004) approach, StreamWare (Gummaraju, 2008) proposes to write dataflow graphs in a dedicated C API and schedule them at runtime on top of a general purpose processor. The same approach was applied to LTE (Ben Abdallah, 2010) using a virtual machine (LUA). The waveform program contains dedicated reconfiguration primitives written in LUA language and interpreted directly on a controller. Those works do not restrict to a particular dataflow MoC.

In a similar approach, the Nucleus tool flow (Castrillon, 2011) comprises a set of tools able to compile and map waveforms. It uses the MAPS (Castrillon, 2013) framework in order to describe actors (so-called *nuclei*) in the CPN language. Different implementations can be provided for each actor, and a user-guided mapping computes a scheduling.

The non-open tool *SystemVue* (Agilent SystemVue, 2014) allows to model waveforms in SDF or TSDF (*Timed* SDF) form. It was used as a basis for a recent work (Hsu, 2010) attempting to address the dynamic behavior of LTE by introducing vectorizers and serializers in the dataflow graph.

The DiplodocusDF approach (Gonzalez-Pina, 2012) extends UML profiles to model dataflow applications. Thanks to a formal semantic, the resulting dedicated UML language can be simulated. Code for the underlying hardware can also be generated, but the mapping has to be done manually.

## 4.4 Mixing programming paradigms

The SPEX approach (Lin, 2006) proposes to program waveforms using three paradigms. *Kernel* SPEX allows a sequential, C-style imperative programming that can be useful for SIMD or VLIW compilation. *Stream* SPEX can be used to program using the dataflow paradigm, following the KPN MoC. *Synchronous* SPEX relies on the paradigm used in synchronous languages such as Esterel or Signal. The distribution of the paradigms is left to the programmer but all parts are included in a C++ program in which 1) the choice of the paradigm is indicated by a keyword, 2) no dynamic object creation is allowed. The compilation of this program involves one compiler for each paradigm.

In a similar manner, IRIS (Sutton, 2010) proposes to write sPHY and fPHY engines. sPHY implements SDF components while fPHY implements KPN components. The mapping of the engines is left to the programmer. The framework provides support for reconfiguration: components may trigger a signal which will lead to the reconfiguration of the kernels.

Lime (Auerbach, 2010) is a Java-based language with extensions to express more parallelism. In Lime, the same method body can be used as a standard function or as an actor in order to program in a dataflow style. In this case, Lime also provides a *match* operator allowing actors to execute at different rates to communicate, thus extending SDF while keeping analysis capabilities. It is associated with a compilation/execution that generates Java bytecode, C, or Verilog, in order to be able to choose between different implementations for each actor.

Finally, It is worth mentioning that many research teams have been working on designing complete system from high level specification in the so-called *hardware-software co-design* domain. These works brought advances in specific aspects such as platform based design or high level synthesis tools such as CatapultC for instance. Although these works did not led to a dedicated SDR environment but might, in the near future, lead to refinement-based SDR programming environment.

## 4.5 Discussion

The survey of SDR programming environments provided above shows that, as it was the case for hardware architectures, there is no agreement on what should be a programming environment for cognitive radio. However, there is a clear trend toward a paradigm shift in order to handle protocols such as LTE. These new protocols are very different from previous signal processing applications, that can be programmed with static traditional parallelization techniques (SDF and/or traditional compilation techniques).

The arguments in favor of dataflow programming models for SDR are:

- Radio waveforms are inherently dataflow because they operate on large data sequences. Although not infinite — they are grouped into frames — radio waveforms still require static (software or hardware) filters that are easily expressed through dataflow actors.
- Software defined radio applications require huge computation performances and hence need to efficiently use parallelism available in hardware. Dataflow formalisms allow for better parallel implementation because it naturally exposes parallelism in many ways: task parallelism, data parallelism and pipeline structure of the program.
- Dataflow programs have a restricted expressivity that allows them to be analyzed in order to verify some properties such as the absence of deadlock, or to improve timing analyses. Such analyses are important since waveforms are becoming more and more complex. New analysis tools will be needed to ensure properties on these programs.

Although they introduce a paradigm shift, dataflow approaches seem necessary. We believe that, at least mixed approaches between this paradigm and imperative concurrent languages will succeed in providing a compromise between programmability, performance and provability. Next section reports on open issues we have identified concerning the programming of SDR platforms, and reviews different basic research tracks to address them.

# 5 OPEN ISSUES

In previous section we have seen that, in order to address the challenges of new communication protocols such as LTE, many works are based on dataflow computation models and dataflow programming languages. However, there is a gap between these works and experimental prototypes. We now report on issues to be addressed in order to fill this gap. We have identified two main directions in which technology should be improved: mapping flows and hardware abstractions.

## 5.1 Mapping flows

One open problem with existing programming frameworks is that they all require a manual *mapping* of the application onto the SDR architecture. The mapping is the phase where the initial specification is split into blocks that are assigned to the different IPs of the architecture. We review below some recent works that attempt to take into account waveform characteristics in SDR programming languages and provide tools to improve the mapping flow.

### Handling dynamicity

Recent works propose new dataflow MoCs that take into account dynamic adaptations required by new communication protocols. One problem with such solutions is to provide languages and compilers for such MoCs. Although there have been many advances in this field, providing such tools requires an

important research and development effort. An example of a new dataflow MoC is Schedulable Parametric DataFlow (Fradet, 2012), language in which it is possible to change actors' parameters while still allowing static analyses.
New compilers such as ORCC (Gorin, 2010) provide support for dataflow programming and dynamic dataflow using just-in-time (JIT) compilation to modify dataflow at runtime, targeting CPUs. Other work (Delahaye, 2007) considers dynamical reconfiguration on heterogeneous platforms based on FPGAs.

### High-level data structures

Another issue in the portability of SDR applications is the ability to directly handle high-level data structures. Indeed, in the second section, we saw that the LTE protocol operates on vectors and matrices. However, current dataflow MoCs and languages only allow manipulation of token flows without making high-level data structures apparent. This prevents compilers and execution layers from: i) optimizing the placement of data, and ii) taking into account the specifics of communication features (DMA, Network-on-Chip, interrupts etc.). The same issue has been reported in non telecom application fields (Thies, 2009).
MVDF (Hsu, 2010) makes one step in this direction by proposing to write dynamic vectorization actors able to produce vectors from a dynamic number of tokens. ArrayOL (Boulet, 2007) and Slice (de Oliveira Castro, 2010) propose to specify static transformations on multi-dimensional arrays. The Sequoia (Fatahalian, 2006) programming language allows programmers to explicitly divide programs into data movement and computation steps in order to optimize data placement at runtime. These works propose static solutions that do not take into account the dynamic variation of data type and/or size. Manipulating dynamic high-level dataflow structures remains an unsolved problem today.

## 5.2 HAL for SDR

The problem of a common *hardware abstraction layer* (HAL) for SDR is definitely not solved. An idea that is emerging slowly is that an Application Programming Interface (API) should be standardized for SDR hardware platforms. This API should include for example an FFT function with various parameters, and probably high-level telecom-specific functions such as Viterbi or Turbo encoding and decoding. However the precise specification of this API has not been done yet and it is not clear on which set of platforms a given API can execute.
Related to this issue, some works attempt to abstract specific hardware in order to lower the need for manual adaptation of the mapping flow. As described in the previous section, one common approach is to consider the use of a dataflow virtual machine (Labonte, 2004; Gummaraju, 2008; Ben Abdallah, 2010). These approaches do not address the problem of mapping waveforms onto the hardware. On the contrary, the Nucleus approach (Castrillon, 2011) and ExoCHI (Wang, 2007) are able to map computation units at runtime, but their mapping procedure cannot be easily extended to many hardware platforms.
Recently, The HDCRAM environment was prototyped (Lazrak, 2012) by Moy et al. This environment will target dynamic reconfiguration of radio protocols on various platforms (DSP, FPGA). It has been used with GNUradio and still has to be tested in other environments.

## 5.3 Resource sharing

Another open problem when programming SDR and specifying waveforms, is to take into account, at the specification level, the concurrent execution of multiple waveforms on the same platform. Many hardware platforms (Clermidy, 2009 ; Schmidt Knorreck, 2012) include hardware mechanisms to ease this *radio context switch* but very few programming environments address this issue.
Siyoum et al. (Siyoum, 2011) show the interest of building different scenarios for a given waveform and express the relationship between each one at the MoC level. This allows static verification of timing properties and optimization of resource usage at runtime. This approach is limited to scenarios written in SDF form.

## 5.4 Discussion

An illustrative example of the difficulty of providing a programming environment portable to different hardware platforms, as are today's retargetable compilers, is given by the Magali chip (Clermidy, 2009). This chip, dedicated to 4G Telecommunication applications contains an OFDM IP which performs FFT as well as deframing (suppression of the band guard). Hence, a mapping tool should be able to gather the software block for FFT and deframing and to map them onto the OFDM IP: there is not necessarily a one-to-one correspondence between actors and hardware IPs (Dardaillon, 2014).

A way to reach portability is to agree on a single API for programming SDR applications. Current solutions are far from this goal: each hardware platform comes with its own specific abstraction.

New MoCs have improved analysis capabilities but are currently not considered in actual design flows. Hence we lack information concerning the performances of these new models once compiled and executed.

One way to bridge the gap between defining new, high-level, analyzable models and providing enhanced execution layers is to statically compute some information and properties on the programs, and use them at runtime to take accurate decisions. Such an approach is currently used by MAPS (Castrillon, 2013), but in a very limited manner since it only uses traces and hand-written information.

Another approach that seems promising to improve portability and performances is the dynamic compilation. The goal is to use a JIT compiler as in (Cohen, 2010) or (Auerbach, 2012) in order to compile dynamically code embedded in a high-level form such as a bytecode. The benefit from this approach is to take advantage of runtime information to compile and map more efficiently.

# 6 CONCLUSION

In this chapter we reviewed the cognitive radio technologies from hardware and software points of view. We started by illustrating new constraints introduced by protocols such as LTE and their impact on current programming models. We provided a review of the different categories of SDR platforms and their possible application fields, and we discussed the programming models used to program these platforms, with a current shift to a new dynamic dataflow programming paradigm. After these observations, we described open issues to bridge the gap between hardware and software, highlighting i) the need for new mapping flows to program SDR platforms efficiently and ii) the need for SDR HAL allowing software reuse from one SDR generation to another.

## AKNOWLEDGEMENT

This work is partially supported by Région Rhône Alpes ADR 11 01302401.

## REFERENCES

Agarwala, S., Rajagopal, A., Hill, A., Joshi, M., Mullinnix, S., Anderson, T., … others. (2007). A 65nm C64x+ multi-core DSP platform for communications infrastructure. In *Solid-State Circuits Conference, ISSCC. Digest of Technical Papers. IEEE International* (pp. 262–601).

Agilent SystemVue. (2014). Retrieved from http://www.agilent.com/find/eesof-systemvue

Auerbach, J., Bacon, D. F., Cheng, P., & Rabbah, R. (2010). Lime: a Java-Compatible and Synthesizable Language for Heterogeneous Architectures. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications - OOPSLA '10* (p. 89). Reno, NV.

Auerbach, J., Bacon, D. F., Burcea, I., Cheng, P., Fink, S. J., Rabbah, R., & Shukla, S. (2012). A compiler and runtime for heterogeneous computing. In *Proceedings of the 49th Annual Design Automation Conference on - DAC '12* (p. 271). San Francisco, CA.

Ben Abdallah, R., Risset, T., Fraboulet, A., & Martin, J. (2010). Virtual Machine for Software Defined Radio: Evaluating the Software VM Approach. In *2010 10th IEEE International Conference on Computer and Information Technology* (pp. 1970–1977). Bradford, UK.

Berkel, K. Van, Heinle, F., Meuwissen, P. P. E., Moerman, K., & Weiss, M. (2005). Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices. *EURASIP Journal on Advances in Signal Processing*, (16), 2613–2625.

Bilsen, G., Engels, M., Lauwereins, R., & Peperstraete, J. (1996). Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*, *44*(2), 397–408.

Bougard, B., De Sutter, B., Verkest, D., Van der Perre, L., & Lauwereins, R. (2008). A Coarse-Grained Array Accelerator for Software-Defined Radio Baseband Processing. *IEEE Micro*, *28*(4), 41–50.

Boulet, P. (2007). *Array-OL revisited, multidimensional intensive signal processing specification*. *INRIA research report num 6113*.

Buck, J. T. (1994). A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications. In *Third International Workshop on Hardware/Software Codesign* (pp. 165–172). Grenoble, France.

Castrillon, J. et. al. (2011). Component-based waveform development: the Nucleus tool flow for efficient and portable software defined radio. *Analog Integrated Circuits and Signal Processing*, *69*(2-3), 173–190.

Castrillon, J., Leupers, R., & Ascheid, G. (2013). MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics*, *9*(1), 527–545.

Clermidy, F., Lemaire, R., Popon, X., Ktenas, D., & Thonnart, Y. (2009). An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application. In *Euromicro Conference on Digital System Design, Architectures, Methods and Tools* (pp. 449–456). Patras, Greece.

Cohen, A., & Rohou, E. (2010). Processor virtualization and split compilation for heterogeneous multicore embedded systems. In *Proceedings of the 47th Design Automation Conference* (pp. 102 - 107). Anaheim, California.

Dardaillon, M., Marquet, K., Risset, T., Martin, J., & Charles, H. (2014). Compilation for heterogeneous SoCs : bridging the gap between software and target-specific mechanisms . In *workshop on High Performance Energy Efficient Embedded Systems - HIPEAC*. Vienna, Austria.

de Oliveira Castro, P., Louise, S., & Barthou, D. (2010). A Multidimensional Array Slicing DSL for Stream Programming. In *International Conference on Complex, Intelligent and Software Intensive Systems* (pp. 913–918). Krakow, Poland.

Delahaye, J., Palicot, J., Moy, C., & Leray, P. (2007). Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform. In *16th IST Mobile and Wireless Communications Summit* (pp. 1–5). Budapest, Hungary.

Derudder, V. et. al. (2009). A 200Mbps+ 2.14 nJ/b digital baseband multi processor system-on-chip for SDRs. In *Symposium on VLSI Circuits* (pp. 292–293). Kyoto, Japan.

Eker, J. et. al. (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, *91*(1), 127–144.

Fatahalian, K. et. al. (2006). Sequoia: Programming the Memory Hierarchy. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (p. 83). Tampa, FL.

Fradet, P., Girault, A., & Poplavko, P. (2012). SPDF: A schedulable parametric data-flow MoC. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 769–774). Dresden, Germany.

Glossner, J., Iancu, D., Moudgill, M., Nacer, G., Jinturkar, S., Stanley, S., & Schulte, M. (2007). The Sandbridge SB3011 Platform. *EURASIP Journal on Embedded Systems*, *2007*(1), 1–16.

GNU radio framework. (n.d.). Retrieved from http://gnuradio.org

Gonzalez, C. R. A. et. al. (2009). Open-source SCA-based core framework and rapid development tools enable software-defined radio education and research. *IEEE Communications Magazine*, *47*(10), 48–55.

Gonzalez-Pina, J., Ameur-Boulifa, R., & Pacalet, R. (2012). DiplodocusDF, a Domain-Specific Modelling Language for Software Defined Radio Applications. In *38th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 1–8). Cesme, Izmir.

Gorin, J., Wipliez, M., Preteux, F., & Raulet, M. (2010). A portable Video Tool Library for MPEG Reconfigurable Video Coding using LLVM representation. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)* (pp. 183–190). Edinburgh, Scotland.

Goubier, T., Sirdey, R., Louise, S., & David, V. (2011). ∑C A Programming Model and Language for Embedded Manycores. In *Algorithms and Architectures for Parallel Processing - 11th International Conference, ICA3PP* (pp. 385–394). Melbourne, Australia.

Gummaraju, J., Coburn, J., Turner, Y., & Rosenblum, M. (2008). Streamware: Programming General-Purpose Multicore Processors Using Streams. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems - ASPLOS XIII* (p. 297). Seattle, WA.

Horrein, P.-H., Hennebert, C., & Pétrot, F. (2011). Integration of GPU Computing in a Software Radio Environment. *Journal of Signal Processing Systems*, *69*(1), 55–65.

Hsu, C.-J., Pino, J. L., & Hu, F.-J. (2010). A mixed-mode vector-based dataflow approach for modeling and simulating LTE physical layer. In *Design Automation Conference (DAC)* (pp. 18–23). Anaheim, California.

Jääskeläinen, P. O., de La Lama, C. S., Huerta, P., & Takala, J. H. (2010). OpenCL-based design methodology for application-specific processors. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation* (pp. 223–230). Samos, Greece.

Jalier, C., Lattard, D., Jerraya, A., Sassatelli, G., Benoit, P., & Torres, L. (2010). Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem. In *Conference on Design, Automation and Test in Europe* (pp. 184–189). Dresden, Germany.

Kahn, G. (1974). The semantics of a simple language for parallel programming. In *Information Processing : Proceedings of the IFIP Congress* (pp. 471 – 475). Stockholm, Sweden.

Labonte, F., Mattson, P., Thies, W., Buck, I., Kozyrakis, C., & Horowitz, M. (2004). The stream virtual machine. In *13th International Conference on Parallel Architecture and Compilation Techniques (PACT)*. (pp. 267–277). Standford, California.

Lazrak, O., Leray, P., & Moy, C. (2012). HDCRAM Proof-of-Concept for Opportunistic Spectrum Access. In *15th Euromicro Conference on Digital System Design* (pp. 453–458). Izmir, Turkey.

Lee, E. A., & Messerschmitt, D. G. (1987). Synchronous Data Flow. *Proceedings of the IEEE*, *75*(9), 1235 – 1245.

Limberg, T. et. al. (2008). A fully programmable 40 GOPS SDR single chip baseband for LTE/WiMAX terminals. In *Solid-State Circuits Conference, ESSCIRC. 34th European* (pp. 466–469). Edinburgh, Scotland.

Linderman, M. D., Collins, J. D., Wang, H., & Meng, T. H. Y. (2008). Merge : A Programming Model for Heterogeneous Multi-core Systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 287–296). Seattle, WA.

Lin, Y. et. al. (2006). SPEX: A programming language for software defined radio. In *SDR Forum Technical Conference* (pp. 13 – 17). Orlando, Florida.

Lodi, A., Cappelli, A., Bocchi, M., Mucci, C., Innocenti, M., DeBartolomeis, C., … Guerrieri, R. (2006). XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module. *IEEE Journal of Solid-State Circuits*, *41*(1), 85–96.

Martin, J., Bernard, C., Clermidy, F., & Durand, Y. (2009). A Microprogrammable Memory Controller for high-performance dataflow applications. In *Proceedings of ESSCIRC* (pp. 348–351). Athens, Greece.

Mei, B., Vernalde, S., Verkest, D., De Man, H., & Lauwereins, R. (2002). DRESC: A retargetable compiler for coarse-grained reconfigurable architectures. In *IEEE International Conference on Field-Programmable Technology (FPT)* (pp. 166–173). Hong Kong.

Minden, G. J. et. al. (2007). KUAR: A Flexible Software-Defined Radio Development Platform. In *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks* (pp. 428–439). Dublin, Ireland.

Mitola, J. (1992). Software radios-survey, critical evaluation and future directions. In *National Telesystems Conference* (pp. 13/15–13/23). Washington, DC , USA.

Nussbaum, D. et. al. (2009). Open Platform for Prototyping of Advanced Software Defined Radio and Cognitive Radio Techniques. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools* (pp. 435–440). Patras, Greece.

Nutaq. (2014). Retrieved from http://www.nutaq.com

Open Air Interface. (2014). Retrieved from http://www.openairinterface.org

Pentek. (2014). Retrieved from http://www.pentek.com

Pulley, D., & Baines, R. (2003). Software defined baseband processing for 3G base stations. In *Fourth International Conference on 3G Mobile Communication Technologies* (Vol. 2003, pp. 123–127). London, UK.

Ramacher, U. (2007). Software-Defined Radio Prospects for Multistandard Mobile Phones. *Computer*, *40*(10), 62–69.

Ramacher, U. et. al. (2011). Architecture and implementation of a Software-Defined Radio baseband processor. In *International Symposium on Circuits and Systems (ISCAS)* (pp. 2193–2196). Rio de Janeiro, Brazil.

Saito, V. S. N. V. M., & Sugiyama, V. I. (2006). Single-Chip Baseband Signal Processor for Software-Defined Radio. *FUJITSU Sci. Tech. J*, *42*(2), 240–247.

Satarkar, S. (2009). *Performance analysis of the WiNC2R platform*. PhD Thesis, Rutgers, The State University of New Jersey.

Schmidt-Knorreck, C. et. al. (2012). Flexible front-end processing for software defined radio applications using application specific instruction-set processors. In *Conference on Design and Architectures for Signal and Image Processing (DASIP),* (pp. 1 – 8). Karlsruhe, Germany.

Schulte, M. J., Glossner, J., Mamidi, S., Moudgill, M., & Vassiliadis, S. (2004). A low-power multithreaded processor for baseband communication systems. *Computer Systems: Architectures, Modeling, and Simulation*, (LNCS 3133), 333–346.

Siyoum, F., Geilen, M., Moreira, O., Nas, R., & Corporaal, H. (2011). Analyzing synchronous dataflow scenarios for dynamic software-defined radio applications. In *2011 International Symposium on System on Chip (SoC)* (pp. 14–21). Eindhoven, Netherlands.

Stuijk, S., Geilen, M., Theelen, B., & Basten, T. (2011). Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation* (pp. 404–411). Samos, Greece.

Sundance. (2014). Retrieved from http://www.sundance.com

Sutton, P. D. et. al. (2010). Iris: an architecture for cognitive radio networking testbeds. *IEEE Communications Magazine*, *48*(9), 114–122.

Tan, K., Liu, H., Zhang, J., Zhang, Y., Fang, J., & Voelker, G. M. (2011). Sora: high-performance software radio using general-purpose multi-core processors. *Communications of the ACM*, *54*(1), 99–107.

Thies, W. (2009). Language and Compiler Support for Stream Programs. PhD Thesis, Massachusetts Institute of Technology.

Truong, D. N. et. al. (2009). A 167-processor computational platform in 65 nm CMOS. *Solid-State Circuits, IEEE Journal of*, *44*(4), 123–127.

UMTS Forum Report 44, *Mobile traffic forecasts 2010-2020*, report January 2011.

Universal software radio peripheral (USRP). (2014). Retrieved from http://www.ettus.com

Wang, P. H. et. al. (2007). EXOCHI: Architecture and Programming Environment for A Heterogeneous Multi-core Multithreaded System. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation* (pp. 156 – 166). San Diego, CA.

WARP. (2014). Retrieved from http://warp.rice.edu

Woh, M., Harel, Y., Mahlke, S., Mudge, T., Chakrabarti, C., & Flautner, K. (2006). SODA: A Low-power Architecture For Software Radio. In *33rd International Symposium on Computer Architecture, ISCA* (pp. 89–101). Boston, MA.

Woh, M. et. al. (2008). From SODA to scotch: The evolution of a wireless baseband processor. In *41st IEEE/ACM International Symposium on Microarchitecture* (pp. 152–163). Como, Italy.

Zhang, Q., Kokkeler, a. B. J., Smit, G. J. M., & Walters, K. H. G. (2009). Cognitive Radio baseband processing on a reconfigurable platform. *Physical Communication*, *2*(1-2), 33–46.

## ADDITIONAL READING SECTION

Anjum, O., Ahonen, T., Garzia, F., Nurmi, J., Brunelli, C., & Berg, H. (2011). State of the art baseband DSP platforms for Software Defined Radio: A survey. *EURASIP Journal on Wireless Communications and Networking*, *2011*(1), 5.

Arnold, M., Fink, S. J., Grove, D., Hind, M., & Sweeney, P. F. (2005). A Survey of Adaptive Optimization in Virtual Machines. *Proceedings of the IEEE*, *93*(2), 449–466.

Bebelis, V., Fradet, P., Girault, A., & Lavigueur, B. (2013). BPDF: A statically analyzable dataflow model with integer and boolean parameters. In *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)* (pp. 1–10). Montreal, QC.

Berg, H., Brunelli, C., & Lucking, U. (2008). Analyzing models of computation for software defined radio applications. In *International Symposium on System-on-Chip* (pp. 1–4). Tampere, Finland.

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., & Menon, R. (2001). *Parallel Programming in OpenMP*. Morgan Kaufmann.

Dardaillon, M., Marquet, K., Risset, T., & Scherrer, A. (2012). Software Defined Radio Architecture Survey for Cognitive Testbeds. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*. Limassol, Cyprus.

De Sutter, B., Raghavan, P., & Lambrechts, A. (2013). Coarse-grained reconfigurable array architectures. In *Handbook of signal processing systems* (pp. 553-592).

Dutta, P., Kuo, Y.-S., Ledeczi, A., Schmid, T., & Volgyesi, P. (2010). Putting the software radio on a low-calorie diet. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets* (pp. 1–6). New Delhi, India.

Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., & Burger, D. (2011). Dark silicon and the end of multicore scaling. In *Proceeding of the 38th annual international symposium on Computer architecture - ISCA* (p. 365). San Jose, CA.

Gonzalez, J., & Pacalet, R. (2014). Model-Driven Design of Software Defined Radio Applications Based on UML. In *Embedded Systems Development* (pp. 69 – 83). Springer New York.

Grayver, E. (2013). Implementing software defined radio. Springer.

Gustafsson, O. et. al. (2010). Architectures for Cognitive Radio Testbeds and Demonstrators – An Overview. In *Proceedings of the Fifth International Conference on Cognitive Radio Oriented Wireless Networks & Communications (CROWNCOM)* (pp. 1 – 6). Cannes, FR.

Jantsch, A., & Sander, I. (2005). Models of computation and languages for embedded system design. *IEE Proceedings - Computers and Digital Techniques*, *152*(2), 114.

Johnston, W. M., Hanna, J. R. P., & Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Computing Surveys*, *36*(1), 1–34.

Jouini, W., Moy, C., & Palicot, J. (2012). Decision making for cognitive radio equipment: analysis of the first 10 years of exploration. *EURASIP Journal on Wireless Communications and Networking*, *2012*(1), 26.

Lee, E. A., & Neuendorffer, S. (2005). Concurrent models of computation for embedded software. *IEE Proceedings - Computers and Digital Techniques*, *152*(2), 239.

Lee, H., Lin, Y., Harel, Y., Woh, M., Mahlke, S., Mudge, T., & Flautner, K. (2005). Software defined radio–a high performance embedded challenge. *High Performance Embedded Architectures and Compilers*, (LNCS 3793), 6–26.

Moreira, O. (2012). Temporal analysis and scheduling of hard real-time radios running on a multi-processor. PhD thesis, TU Eindhoven.

Munk, H. et. al. (2011). Acotes project: Advanced compiler technologies for embedded streaming. *International Journal of Parallel Programming*, *39*(3), 397-450.

Palkovic, M., Raghavan, P., Li, M., Dejonghe, A., Van der Perre, L., & Catthoor, F. (2010). Future Software-Defined Radio Platforms and Mapping Flows. *Signal Processing Magazine, IEEE*, *27*(2), 22–33.

Pop, A., & Cohen, A. (2013). OpenStream: Expressiveness and Data-Flow Compilation of OpenMP Streaming Programs. *ACM Transactions on Architecture and Code Optimization*, *9*(4), 1–25.

Sajjad, K. (2011). *Porting Different Compilation Phases to Runtime*. PhD thesis, Versailles Saint-Quentin-en-Yvelines.

Ulversoy, T. (2010). Software defined radio: Challenges and opportunities. *Communications Surveys & Tutorials, IEEE*, *12*(4), 531–550.

Woh, M. et. al. (2007). The next generation challenge for software defined radio. In *Embedded Computer Systems: Architectures, Modeling, and Simulation* (pp. 343–354). Samos, Greece.

Zyren, J., & McCoy, W. (2007). Overview of the 3GPP long term evolution physical layer. *Freescale Semiconductor, Inc., White Paper*.

## KEY TERMS AND DEFINITIONS

Cognitive radio: wireless communication system that can sense the air and decide to configure itself in a given mode.
Dataflow: Model of computation in which a program is represented as a graph, nodes represent operations and edges data communications.
Digital Signal Processor (DSP): Specialized microprocessor with an architecture optimized for the operational needs of digital signal processing.
Field Programmable Gate Array (FPGA): Integrated circuit designed to be configured after manufacturing, using a hardware description language.
Hardware Abstraction Layer: Software routines that abstract platform-specific details, giving programs direct access to hardware resources.
Programming model: Fundamental style of computer programming, a way of building the structure and elements of computer programs.
Software Defined Radio (SDR): Radio system where parts of the signal processing are done in software.