A Model-Based Approach to Cognitive Radio Design

Jörg Lotze, *Student Member, IEEE,* Suhaib A. Fahmy, *Member, IEEE,* Juanjo Noguera, *Member, IEEE,* and Linda E. Doyle, *Member, IEEE*

***Abstract*—Cognitive radio is a promising technology for fulfilling the spectrum and service requirements of future wireless communication systems. Real experimentation is a key factor for driving research forward. However, the experimentation testbeds available today are cumbersome to use, require detailed platform knowledge, and often lack high level design methods and tools. In this paper we propose a novel cognitive radio design technique, based on a high-level model which is implementation independent, supports design-time correctness checks, and clearly defines the underlying execution semantics. A radio designed using this technique can be synthesised to various real radio platforms automatically; detailed knowledge of the target platform is not required. The proposed technique therefore simplifies cognitive radio design and implementation significantly, allowing researchers to validate ideas in experiments without extensive engineering effort. One example target platform is proposed, comprising software and reconfigurable hardware. The design technique is demonstrated for this platform through the development of two realistic cognitive radio applications.**

***Index Terms*—cognitive radio, radio design, design methods, radio platforms.**

I. Introduction

COGNITIVE radio is "a type of radio in which communication systems are aware of their environment and internal state and can make decisions about their radio operating behaviour based on that information and predefined objectives" [1]. Experimentation using realistic platforms and testbeds is key to bringing cognitive radios [2], [3] and cognitive networks [4] closer to reality. As cognitive radio research advances, and nodes become more numerous, autonomous, and flexible, a number of design challenges relating to coexistence and interference arise, which cannot be analysed and tackled at design time. Only through real experimentation on testbeds can such systems be analysed, tested and verified.

This paper presents a detailed review and comparison of existing cognitive radio test and experimentation platforms and frameworks, including a discussion of their limitations (Section II). We find that design approaches vary greatly, require in-depth understanding of the low-level platform architecture, and lack any systematic way of dealing with reconfiguration. The model of computation (MoC), defining the execution semantics of the radio, i.e., how a radio is executed, is often not clearly defined. This results in a cumbersome, ad-hoc approach to radio development, severely limited design portability, and experimental results which are often impossible for other researchers to fully understand and reproduce. This ad-hoc approach also creates a situation where validating promising research ideas in a real setting becomes a complex engineering task, that distracts from developing the ideas further on a solid experimental foundation.

Additionally, many existing platforms focus solely on software and do not support embedded hardware-accelerated systems, e.g. field programmable gate arrays (FPGAs). Those which do integrate hardware often require the user to have extensive hardware design experience and are thus considered too complex for use by cognitive radio researchers. This makes the use of computationally-intensive signal processing techniques, which often require hardware acceleration for real-time performance, unfeasible for experimental systems.

A platform-independent model for describing the system composition of CRs, based on functional blocks, is proposed (Section III). It allows designs to be understood and checked for computational functionality without first being implemented. The model is also the first to combine both the radio processing and reconfiguration within the same description, simplifying computational verification and precluding the need for implementation platform knowledge. This enables radio researchers to focus on applications of interest, designing, iterating and evaluating real systems, without the need to focus on low-level details. It reduces design effort, provides higher reliability, and allows easy deployment on different target platforms.

As an example target platform, we show how this model can be used to implement FPGA-based CRs through a customised tool flow we have developed (Section IV). Components are mapped through an existing library of software and hardware components, creating a real system that matches the model, including reconfigurability. The tools take care of generating all the necessary configurations, meaning a designer with no FPGA experience can still target such a platform, greatly mitigating previous barriers to entry. Two example applications are presented to show the portability of our approach (Section V). They were mapped onto two different FPGA platforms and demonstrated at the IEEE International Symposium on Field

Manuscript received 1 December 2009; revised 31 May 2010. This material is based upon work supported by Enterprise Ireland under its Innovation Partnership scheme, project IP20060367 and by Science Foundation Ireland under Grant No. 03/CE3/I405.

J. Lotze and L. E. Doyle are with CTVR, The Telecommunications Research Centre, at University of Dublin, Trinity College, Ireland (e-mail: jlotze@tcd.ie, linda.doyle@tcd.ie).

S. A. Fahmy is with the School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: sfahmy@ntu.edu.sg).

J. Noguera is with Xilinx Research Labs, Dublin, Ireland. (e-mail: juanjo.nogurea@xilinx.com).

Digital Object Identifier 10.1109/JSAC.2011.110217.

Programmable Custom Computing Machines (FCCM) and ACM SIGCOMM in 2009.

The methods outlined in this paper are beneficial to two main groups of researchers. Those working on physical layer components, e.g., on new modulation, detection, or coding techniques, can quickly compose full cognitive radio systems which include their new components. They do not need to concern themselves with the communication with other components, the connection to the cognitive decision making process, and reconfiguration management. Secondly, researchers working on networks can use the proposed methods to quickly compose physical layers by combining existing components and re-using other designs, while reconfiguring aspects as needed at run-time. They need not concern themselves with platform-specific details. Both groups are able to test their techniques on real platforms in realistic scenarios, without being reliant solely on simulations results. Furthermore, they can describe the behaviour of their radio and communicate their designs to other researchers easily.

II. Existing Radio Experimentation Platforms

In this section we give an overview of existing frameworks for designing and implementing software-defined radios (SDRs) and cognitive radios. Their properties are analysed in Section II-C, and the challenges facing cognitive radio designers using these frameworks are identified.

A. High-Level Design

The frameworks presented here provide a rich set of tools and re-usable components to aid radio design. Their key feature is allowing radio development at a high level of abstraction.

1) GNU Radio: GNU Radio is an open source software development toolkit for software radio applications [5]. It runs on general purpose processors (GPPs), allowing for easy, low-cost software radio experimentation and development. Radio applications are constructed by connecting and configuring existing re-useable signal processing components through Python scripts. The performance-critical components are compiled from C++. A large number of processing components are available in GNU Radio.

To reconfigure radios, the Python application can pause the execution, reconfigure the components and connections, and resume execution. Information about the radio in execution can be retrieved through message queues, although this feature is rarely used in the current version of GNU Radio.

2) Open-Source SCA Implementation::Embedded: OSSIE is a lightweight Software Communication Architecture (SCA)[1] implementation [6]. The SCA is an interoperable, multi-platform architecture framework for software radio systems [7]. OSSIE is implemented purely in C++ for execution on general purpose processors (GPPs), allowing fast and easy prototyping for experimentation and research. A library of pre-built components and waveforms is available. Radio applications (called *waveforms* in SCA terms) are composed from signal processing components using graphical tools or XML.

[1]The SCA is at the core of the JTRS project, hosted by the U.S. DoD.

OSSIE processing components run as separate processes, communicating via message queues provided by the CORBA middleware. Since the SCA is for SDRs, rather than cognitive radios [1], OSSIE does not support run-time reconfiguration.

In addition to GPPs, the OSSIE project has targeted embedded systems using digital signal processors (DSPs) as the main processor [8].

3) Iris: Iris [9] is a flexible and reconfigurable framework for a variety of platforms, also supporting FPGA-based platforms with some processing components running in hardware [10], [11]. Radios for Iris, i.e., the configuration and connection of signal processing components, are described in XML. A Decision Engine can be implemented by the user to subscribe and react to events triggered by radio components. Reactions can comprise of anything from diagnostic output to full reconfiguration of the radio application.

The Iris engine, the component library and the Decision Engine are compiled from C++. FPGA processing components can be developed using any hardware design flow, though typically, a hardware description language is used.

Iris radios are executed using a simple synchronous scheduler, executing all components once per scheduling cycle. This ensures efficient execution, but limits the range of radio systems that can be implemented. Currently a new release of Iris is under development, which allows more powerful and flexible radio configurations [12].

B. Low-Level Design

Another set of cognitive radio/software radio frameworks provides tools and application programming interfaces (APIs) at a lower level. These are used to develop radio applications from the ground up, using programming languages and/or hardware description languages. No library of generic re-usable processing components is provided. This gives the developers a higher degree of freedom in how they implement the radio. They can decide how the signal processing is implemented, how it is executed, and how reconfiguration is realised. However, this results in more effort being needed to realise a particular system.

1) Wireless Open-Access Research Platform: WARP is a scalable, extensible and programmable hardware platform with a Xilinx Virtex-II Pro FPGA as its baseband processor and up to four RF daughter boards [13]. The physical layer of a radio is implemented in the FPGA logic fabric, while MAC layer functionality can be implemented in C using the embedded PowerPC processor cores (without an operating system). In October 2009 a new version of the WARP board was released, featuring, among other capabilities, the more powerful Virtex-4 FPGA and Gigabit Ethernet connectivity [14].

Various reference designs and tutorials are available detailing a variety of radio implementations on the WARP board. Standard Xilinx tools are used to program the WARP board, allowing efficient software radio implementations. Hence, significant hardware expertise is required to take advantage of the WARP platform.

2) Lyrtech: Lyrtech is a company that offers a variety of SDR development platforms, together with software and hardware development kits [15]. The radio architecture used

TABLE I
SUMMARY OF THE DISCUSSED PLATFORMS.

	Design Level	Reconf. Runtime	Comp. Tools[1]	Defines MoC	Real-time Hardware
GNURadio [5]	high	●	●	×	×
OSSIE [6]	high	×	○[2]	×	○
Iris [9]	high	●	●	×[3]	●
WARP [13]	low	×	×	×	●
Lyrtech [15]	low	×	○	×	●
KUAR [17]	low	○	○	×	●
WiNC2R [18]	low	●	×	×	●

● : fully supported; ○ : partly supported; × : not supported;
[1] high-level radio composition tools, to assemble a full system
[2] the design framework does not include cognition
[3] will be supported in future release [12]

is based on the SCA, extended to support FPGA and DSP components [16]. Lyrtech's hardware platforms are modular, consisting of antennas, an RF module, an interface module, a baseband processor and optional expansion modules. The baseband processor is a hybrid system with a DSP processor, an ARM processor, and an FPGA. A software development kit for DSP and ARM development is provided, along with an extension of Xilinx System Generator for MATLAB-based FPGA design.

The Lyrtech platforms are powerful, flexible and come in a small form factor. Since they are SCA compliant, it is possible to compose a radio from SCA components at a relatively high level. However, the SCA component granularity is usually coarse, for instance comprising full narrowband receive and transmit waveforms. This limits the re-useability for the design and experimentations with new radio configurations and often, desired waveforms must be developed from the ground up.

The FPGA, DSP, and ARM are capable of run-time reconfiguration, but since Lyrtech is targeting SDRs, rather than cognitive radios, no design or run-time tools for cognition and run-time reconfiguration are provided.

3) Kansas University Agile Radio: KUAR is a compact, powerful, and flexible software radio development and experimentation platform for cognitive radio research [17]. It consists of a full embedded Pentium PC running the Linux operating system, a Xilinx Virtex-II Pro FPGA, an RF front-end, and active antennas. Its small form factor and the optional battery pack make it easily portable. KUAR provides software and hardware APIs for configuration and control and for processor-FPGA communication. Radio applications are developed using standard compilers and low-level FPGA design tools. A small library of radio components is provided.

Radio designers have to implement the signal processing, execution model, radio life-cycle management, and control using the APIs provided. This ensures a high degree of freedom and high performance, at the cost of an increased development effort and tight coupling to the implementation platform.

4) WINLAB Network Centric Cognitive Radio: The WiNC2R is a cognitive radio platform that uses flexible hardware accelerators to achieve programmability and high performance at each layer of the protocol stack [18]. The prototype consists of one or more baseband modules (with an FPGA at their core), each connected to an RF module, a networking module, and a CPU. The performance intense radio functions are executed on dedicated hardware accelerators (implemented in FPGA logic), while control intensive functions are performed by data processors (implemented as soft CPU cores on the FPGAs). A system scheduler manages the synchronisation of all the processing elements as well as data transfers. System reconfiguration is managed by the external CPU which can reconfigure hardware accelerators, data processors, and the system scheduler if required. The WiNC2R is still at an early stage of development; the prototype is not fully developed and no design tools are available thus far.

C. Analysis

The cognitive radio frameworks discussed in Section II-A all provide a library of re-usable signal processing components and compose radios by connecting these together. This design method is often described as *dataflow-based design*. A dataflow graph is a directed graph where the nodes represent processing components and the edges represent the flow of data through the system. This is an intuitive and established approach for describing signal processing systems and is used by many design and simulation tools outside of cognitive radio (e.g., [19]–[21]). It decouples the signal processing implementation from the system composition and allows for the composition of systems without detailed platform knowledge.

However, in order to *efficiently* use dataflow graphs for design, the dataflow model of computation (MoC) must be known to the designer. The execution semantics, i.e., how components are executed and how data is passed between components, must be clearly defined in the MoC, ideally in terms independent of low-level implementation details. The three described frameworks do not define the applied MoC in an implementation-independent way, thus, radio designers need detailed platform knowledge in order to design efficient radios.

The platforms discussed in Section II-B require low-level design expertise. This allows for a great level of freedom, allowing almost any radio system to be implemented. However, the considerable implementation effort required is often impractical for research and experimentation.

It is also clear that the frameworks discussed often lack a well-defined way of expressing reconfigurability. Even more importantly, no existent modelling approaches allow the designer to do so in an implementation-independent way. Mechanisms for reconfiguration are provided for some, but vary greatly between frameworks and require in-depth understanding of the low-level architecture. This makes it difficult for radio designers to experiment with and reason about cognitive radios and networks. It is also important to note that the high-level frameworks primarily target software on general purpose processors, while the low-level frameworks focus more on hardware implementations, mostly using FPGAs.

Table I shows a comparison of the frameworks discussed. The comparison criteria are based on the discussions above.

To achieve the processing performance and power efficiency required for realistic, cutting edge, real-time cognitive radio implementations, it is important that high performance computational resources like FPGAs and DSPs are available to the radio designer. It is also important for the designer to be able to think independently of the implementation platform. In this paper, we attempt to offer the best of both worlds: a high-level platform-independent model for cognitive radio design, that includes both composition and reconfiguration; and an example design flow and tools showing how the model can be mapped to a real high performance implementation platform.

III. Cognitive Radio Model

It is hugely beneficial for researchers and developers of cognitive radios to be able to design such systems at a high level of abstraction, avoiding detailed platform knowledge where possible. This reduces the design effort, increases reliability, and allows easy deployment on different target platforms. It also prevents the limitations of a single platform from working their way into the radio description, resulting in constraints that are of no relation to the application being described. Developers instead focus on the radio's function and cognitive decision-making and need not concern themselves with platform details.

We believe that the dataflow-based design approach adopted by the frameworks discussed in Section II-A is an important foundation for describing such systems. However, such a description can only be platform-independent if the MoC is defined independently of implementation details. More importantly, dataflow graphs do not capture the dynamic nature of cognitive radios; they can only represent a single configuration. Incorporating other configurations that might arise as a result of the cognitive decision-making would need the model to be extended.

This section proposes a model for describing cognitive radio designs that harnesses the strengths of existing dataflow-based methods while overcoming the current limitations of existing design frameworks. It describes the radio's function, all reconfiguration options, and defines an interface to the cognitive decision-making process without relying on any platform implementation details. It can be used not only as a basis for developing real cognitive radio implementations, but also as a formal description of cognitive radios which can then be understood and analysed by other researchers.

A. Data Plane

Dataflow graphs are an established approach for designing signal processing systems and so, we base our model on this approach. It is common in signal processing to use a hierarchical flow graph, where a nodes can represent other flow graphs. For example, an OFDM node in a graph might be broken down into a graph with nodes for subcarrier modulation, inverse FFT, and cyclic prefix insertion. To capture the dynamic nature of cognitive radios, we must allow multiple options to be selectable for a node in the flow graph. For example, the subcarrier modulation node in such an OFDM system might contain multiple modulators from BPSK to 64-QAM, only one of which is active at a time. Here we present the model we have developed.

Each level of hierarchy is a dataflow graph, i.e., a directed connected graph, where the nodes represent either processing *components* or *subsystems* and the edges (*links*) represent the flow of data.

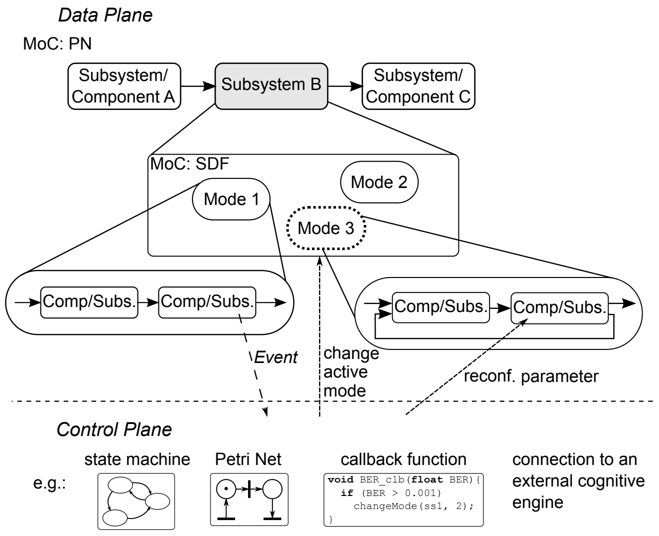
Components are the atomic entities for processing data in the model. They take data from their input *ports*, perform processing, and produce data on their output *ports*. An example is a QPSK modulator, which maps information bits to complex signal samples. A component has a set of *parameters*, which may be the empty set. For example, a pulse-shaping filter might have parameters for the number of taps and the roll-off factor. Components have a set of input and output ports. Arbitrary numbers of inputs and outputs are permitted. For instance, a 'select' component has two input ports and one output port, routing one of the inputs to the output port depending on a parameter. Components without inputs are *sources*, components without outputs are *sinks*. Components without ports are not permitted. Note that this model does not include the internal representation of components; it concerns itself with the composition of radios from a library of components.

A *subsystem* represents a new level of hierarchy. It can combine multiple mutually-exclusive dataflow graphs, only one of which is active at any given time. Each of these graphs is what we call a *mode* in the subsystem. An example is a modulator subsystem which contains modes for different QAM modulators ranging from QPSK to 64-QAM. The radio can then be configured for any type of QAM modulation by activating the corresponding mode. When viewed from the next level up in the hierarchy, the subsystem is, at any point in time, represented by the the dataflow graph for the active mode. The graphs for each mode in a subsystem are allowed to have external input and output ports, with the same restrictions as for components. To ensure that each of the graphs in a subsystem (i.e., each mode) is compatible with the connections of the subsystem in the higher level graph, we restrict all modes with one subsystem to have the same number of input and output ports. Subsystems are used to build a hierarchy of flow graphs for the cognitive radio system. The depth of the hierarchy is not bounded in this model, becoming more detailed when going down the hierarchy until the dataflow graphs no longer contain any subsystems at the lowest level. This extension to existing dataflow models allows us to capture reconfiguration of arbitrary complexity.

Component *parameters* are used to tune the operation of each component. They have data types, a specification of allowed values (which might be an interval or a list), and a value. Additionally, some components might be run-time reconfigurable. For example, a scaling component might expose the run-time reconfigurable parameter 'gain' of type real, with allowed values in the interval $[0, \infty)$, to allow tuning of the scaler at run-time.

Component or subsystem *ports* can be either input or output ports. A data type is associated with each port, and defines the type of the data items consumed or produced. When active, components consume data from their input ports and produce data on their output ports, for example by mapping information bits to signal samples in a modulator component.

A *link* is a connection of output ports to input ports of components or subsystems. It resembles a first in first out (FIFO) buffer, i.e., multiple data items can be stored on a link. Data items produced on output ports are queued at the end of the buffer and items consumed from input ports are



```
void BER_clb(float BER){
  if (BER > 0.001)
    changeMode(ss1, 2);
}
```

Fig. 1. The proposed cognitive radio model. The data plane shows two levels of hierarchy, using a subsystem with three modes. Two different MoCs are applied: Dataflow Process Networks (PN) and Synchronous Dataflow (SDF).

taken from the beginning. For instance, information bits are passed from a convolutional coder component to a modulator component via a link.

The data plane of the proposed cognitive radio model is illustrated in the top section of Fig. 1, showing two hierarchy levels and using a subsystem with three modes.

As discussed in the beginning of this section, it is essential to define the MoC of the model, i.e., its execution semantics, in terms that are independent of implementation details. With this information, radio designers can successfully construct a model of the radio and understand its execution behaviour.

The MoC for the proposed model is defined as follows. Each dataflow graph at any level in the model hierarchy can be treated as a flat dataflow graph by viewing the subsystems it contains as single components. Well-known and well-investigated dataflow MoCs can then be applied to these flat graphs. We therefore require that the top level of the model, as well as every subsystem within it, define the dataflow MoC applied. Every mode within a subsystem must apply the same MoC, though it is possible to combine different MoCs at different hierarchy levels. However, this can only be done if the MoCs are compatible (this is discussed later). A large number of dataflow MoCs have been proposed in the literature, (see [22] for a summary of many of these). We have constructed the proposed model based on the two most widely used for signal processing: the extremely flexible Process Networks (PN) MoC and the highly efficient Synchronous Dataflow (SDF) MoC, combining the two major requirements for cognitive radio systems: flexibility and efficiency.

1) Dataflow Process Networks: The PN MoC was originally published by Kahn [23], and was later refined by Lee and Parks [24]. It is the most flexible dataflow MoC. In the PN MoC all components are processes that execute asynchronously. They read their inputs by blocking reads on the input ports and write to the output ports using non-blocking write operations. The only permitted way for processes to communicate is through input and output links. This ensures deterministic behaviour. There are no restrictions on when processes output data and how much data they output. PNs have been shown to be Turing complete, i.e., all algorithms that can be realised on a theoretical Turing machine can also be realised using a PN graph.

A PN graph might execute in unbounded memory and it is generally undecidable whether bounded memory is sufficient for its execution. This makes it impossible to execute general PN graphs on a real machine. Lee and Parks [24] proposed an execution method using blocking writes, starting with small buffer sizes on each link and dynamically increasing them as needed at run-time up to a limit. This method allows the execution of a large subset of PN graphs on real machines. Note that due to the blocking reads and associated context switches, the PN MoC can pose a significant run-time overhead.

The PN MoC is suitable for graphs containing components producing or consuming variable numbers of data items. For instance, a flexible receiver radio containing a bandwidth-adaptive OFDM demodulator [25] requires the application of the PN MoC due to unpredictable input and output data rates of that component.

2) Synchronous Dataflow: A dataflow graph is called synchronous if fixed sampling rates can be given on each port of all components in the graph [26]. That is, the number of items consumed on each input port and the corresponding number of output items produced per component execution is fixed and known at design-time. This knowledge allows a thorough analysis of the graph at design-time: required buffer sizes on each link can be determined, guarantees for deadlock-free operation can be given, and static and efficient execution schedules can be computed. These properties make SDF graphs extremely valuable for signal processing applications and allow for execution with minimum overhead. Methods to check for SDF graph correctness and to construct schedules are given in [27].

SDF graphs are a subclass of PN graphs, i.e., all SDF graphs can also be executed using the PN MoC. This implies for the proposed cognitive radio model that PN graphs may contain SDF subsystems, but not vice versa. SDF graphs are not Turing complete, i.e., some algorithms cannot be modelled using SDF graphs. For example, a ‘select’ component, which routes one of its input ports to the output, depending on the value of a boolean input, cannot be modelled using SDF.

Many physical layer components typically used in radios are synchronous. For example, a pulse-shaping filter always produces a single output item for each input item. A QPSK modulator maps two input bits to one complex sample at the output. Graphs containing these components can be efficiently executed using the SDF MoC.

B. Reconfiguration

Cognitive radios need to support frequent reconfiguration to adapt their behaviour to the current environmental conditions. These reconfigurations can be classified into three types: *parametric*, *structural*, or *functional*. A parametric reconfiguration is a change of a component parameter in one or more of the components in the data plane. For instance, the gain of a scaling component can be adjusted. A structural reconfiguration is a change of the radio's structure, i.e., of the currently executing dataflow graph. For instance, replacing a QPSK modulator component with a 16-QAM modulator. This structural reconfiguration is represented by a switch of the active mode in a subsystem. A functional reconfiguration involves a complete overhaul of the radio's function, e.g., from WiFi to UMTS. This can be treated as a special case of structural reconfigurations, where a single subsystem at the top level contains modes for each function. Thus, the proposed hierarchical structure of the radio data plane along with component parameters enables all types of reconfiguration.

It has proven valuable for checking the model's correctness (see Section III-D) and to aid automatic synthesis tools, to make all reconfiguration options explicit in the radio model's data plane. That is, parameters that might be changed at run-time, along with their possible values, are added to the set of reconfigurable parameters; no other parameters may be reconfigured. This set of reconfigurable parameters forms what we call the *Control Specification*.

C. Control Plane

The control plane of the radio model is responsible for monitoring the execution of the data plane, performing radio reconfigurations, and setting the active modes in all subsystems of the data plane. It executes the *cognition cycle* of a cognitive radio, which involves three stages: *observe*, *decide*, and *act* (a simplified version of the cycle introduced in [28]). During the *observe* stage, the control plane monitors the state of the radio and can retrieve information from the data plane. The *decide* stage involves decision-making using the observed information, and might include machine learning techniques. During the *act* stage, the chosen reconfiguration actions are applied to the data plane. In the literature, the part of a cognitive radio which executes the cognition cycle is often called *cognitive engine*.

The control plane is connected to the data plane using the well-known Publish and Subscribe MoC (see for example [22]). That is, components in the data plane publish events which they may trigger and the control plane can subscribe to them. Subscribing to an event means the control plane is notified when it occurs. Events can carry information, e.g., the current bit error rate (BER), or a map of available spectrum. This can be used to obtain information from the environment and the radio itself; it therefore represents the *observe* stage of the cognition cycle. Information about the radio may also be retrieved without an event being triggered, for example by querying the current state of the data plane.

Due to the wide variety of approaches and methods for decision-making in a cognitive radio [29], i.e., the *decide* stage, the definition of a strict MoC for this step would severely limit the range of algorithms implementable. Instead, the model for this part is unspecified, increasing the radio model's flexibility. For example, simple radio controllers might be implemented using the finite state machine (FSM) or Petri net [30] MoCs, where switches of the active mode in subsystems are dependent on simple conditions. An example would be switching the applied coding scheme based on predefined thresholds of the BER. More complex controllers can use callback functions in software code, invoked when certain events are triggered. Those could implement a genetic algorithm maximising an objective function given on a number of possible actions, to find the optimum radio configuration in a given situation.

The controller applies the chosen actions to the data plane in the *act* stage. Those can be changes in parameter values of components or switches of the active mode in a subsystem. For example, one set of actions might change the modulation scheme from QPSK to 16-QAM by switching the active mode in a modulator subsystem and choose a higher roll-off factor for the pulse-shaping filter by adjusting the corresponding parameter of that component.

D. Model Correctness

Defining the cognitive radio model and its MoC in platform-independent terms has the additional advantage that a model can be checked for correct construction. It is possible to detect errors that violate the model's constraints and its MoC, ensuring that the radio is executable on a real platform. This is a powerful feature of the proposed model since it allows early detection of errors, saving development time and effort.

1) General Criteria: The following criteria are straightforward to check given a specific radio model, therefore we do not give details here:

- All component parameters exist and the assigned values are within range,
- parameter reconfigurations are allowed and within range,
- all link data types can be uniquely determined,
- the link data types are consistent and supported by components, and
- SDF graphs do not contain PN subsystems.

To enable the verification of these criteria, information about the ports and parameters of existing components (from the component library) must be available. If no such component implementation exists, a set of requirements for component implementation can be derived from the model (parameters, data types, reconfigurability). These can be used later in the design process to implement the missing components.

As discussed in Section III-A1, it is generally undecidable whether a PN graph is executable on a real system. Therefore, if the MoC used in all hierarchy levels of the given model is PN, there is nothing more that can be done at design-time. The radio designer must synthesise the model onto a real platform and execute the radio in order to find out whether it can be executed in bounded memory and without deadlocks.

2) Criteria for Models Using the SDF MoC: The SDF MoC allows a model to be checked for correct construction without executing it (see Section III-A2). That is, guarantees for the existence of a cyclic execution schedule and the absence of deadlocks, and the boundedness of the buffer memory on the links can be given. This feature can be used to develop correctness checks for the proposed model if some portion of it employs the SDF MoC. In order to do this, we introduce the following notation.

Let S_n denote the set of all modes in subsystem n of a given model. Further we denote the set of desired reconfiguration values for parameter m as P_m. Then, at any given time instant, the global state of the data plane is given as the Cartesian product of all S_n and P_m in the model, i.e.,

$$G = \prod_{n=0}^{N-1} S_n \times \prod_{m=0}^{M-1} P_m, \qquad (1)$$

where N denotes the total number of subsystems in the model, M is the total number of reconfigured parameters, and $\prod$ is the Cartesian product of the arguments. The resulting set G is a set of tuples, each representing a state of the data plane.

For each of the states in G, the hierarchy can be flattened to a single dataflow graph with fixed parameters. Therefore it is possible to apply the well-known SDF graph correctness checks [26], [27] for the SDF portions of the data plane. By checking each of the states it can be guaranteed that the model is correctly constructed.

However, the number of states $|G|$ can get very large due to the Cartesian product in (1), which results in a potentially very complex model checking procedure. To reduce the number of states for model checking, the following rules can be applied:

1) Only SDF subsystems need to be considered, subsystems using the PN MoC can be eliminated.
2) All reconfigured parameters that do not affect the input/output sampling rates of a component can be eliminated.
3) On multiple hierarchy levels, some subsystems can never be active simultaneously; these combinations can be eliminated.
4) In each SDF subsystem, modes with the same external input/output sampling rates can be reduced to one mode.
5) In each SDF subsystem, modes where the external input/output ratios are multiples of each other can be eliminated by computing the lowest common multiple of the rates and repeating the mode schedules accordingly, so that they have the same external input/output rate.
6) If some modes or parameter settings are mutually exclusive, as may be given in constraints by the radio designer, the number of combinations can be reduced.

Thus, the reduced state space can then be expressed as

$$\check{G} = \prod_{i \in I} \mathrm{reduce}(S_i) \times \prod_{j \in J} P_j \qquad (2)$$

where the index set I is the set of SDF subsystem indices, the index set J is the set of indices of reconfigured parameters that affect the component's input/output sampling rates, and the function 'reduce' applies the mode reduction techniques as explained above in rules 3 to 6.

The model can be flattened to a single dataflow graph for each state in $\check{G}$, correctness checks can be applied, and SDF schedules can be computed. If all checks succeed, the model is correctly constructed. Note that these checks are performed at design-time and are therefore not performance-critical.

E. Model Transformation

Due to the hierarchical nature of the proposed model, it is often possible to express the same functionality in different ways. In fact, it is possible to transform one representation into another by applying simple rules, without affecting the functionality of the radio. This is important since the model can be transformed into representations that are easier to synthesise, align more naturally with the resources, and execute more efficiently on any given platform.

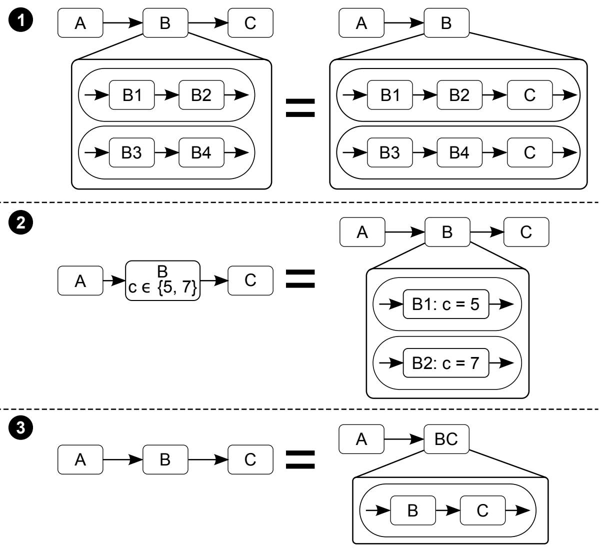


Fig. 2. Model transformation relationships.

The following behaviour-preserving transformation rules can be given for the proposed radio model (see Fig. 2):

1) A component at the input or output of a subsystem may be moved into the subsystem by replicating it in all of the subsystem's modes.
2) A component with a parameter with discrete reconfiguration values may be transformed into a subsystem with separate modes for each of the possible parameter values, making them fixed parameters in each of the modes.
3) Subgraphs of a dataflow graph might be placed into a subsystem containing only one mode.

Note that the application of transformation rules 1) and 2) might result in different behaviour to the original graph if the affected component has an internal state. To be truly equivalent, the duplicated components must share their internal state. However, depending on the application, this strict equivalence may not be required. For instance, if a pulse-shaping filter with reconfigurable roll-off factor is transformed using rule 2), the delay line does not need sharing since after a reconfiguration, the output is considered invalid until after the filter delay has passed anyway.

F. Summary

The model presented here completely and accurately describes the behaviour of a cognitive radio, including its reconfiguration options. The execution semantics are clearly defined. Correctness checks are given to allow for detection of composition errors at design-time. The model is implementation-independent, yet, the transformations presented allow it to be modified to a given target platform. It allows radio designers to compose cognitive radios, understand their behaviour and check them for composition errors without considering a target platform. This greatly reduces the effort needed for testing cognitive radio algorithms in the real world

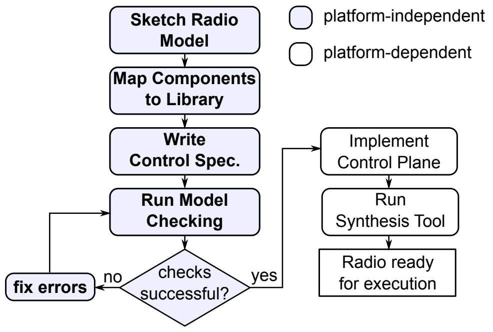


Fig. 3. Proposed Design Flow.

and allows researchers to communicate their designs to each other.

IV. From Model to Architecture

Having defined the modelling approach, we now discuss how a model can by implemented on physical platforms, creating real cognitive radios for experimentation. First, we look at the general design flow for generating real radio configurations. Then we discuss synthesis of the model for an FPGA-based cognitive radio platform running Iris (see Section II-A3). This platform is just one example. Since FPGA-based systems are more complicated to handle than pure software implementations, this example is sufficient to prove that the design approach presented can also be applied to pure software platforms.

A. Design Flow

Radio design using the proposed model-based cognitive radio design technique facilitates system-level design and composition of radios using components existent in a component library. Radio designers follow the flow given in Fig. 3.

The two aspects of a cognitive radio design are the data plane and the control plane. First, a radio designer describes the data plane using a hierarchical flow graph of radio components (see Section III-A). Then the designer maps the desired components to those available in the component library, and describes the radio model as presented in Section III.

The *Control Specification*, as introduced in Section III-B, specifies which component parameters are required to be run-time reconfigurable and the range or set of values they can take. This is required for the model correctness checks as discussed in Section III-D.

The radio model and the Control Specification are passed to the model checking tool which checks if the radio is composed correctly using the methods presented in Section III-D. Errors in the model are indicated to the developer who may then fix them and re-run the checks until they succeed.

Up to this point in the design flow, the radio design is platform-independent. Now the radio developer may choose a target platform, provided that model synthesis tools are available for the given platform. If this is not the case, developers have to implement the radio manually. In such a case, the model-based design technique is still beneficial since the model can be used as a design specification for manual implementation.

For the platform-specific synthesis step we first outline the architecture of our FPGA-based platform running the Iris software radio.

B. Platform Architecture

Our example radio platform is built around an FPGA. FPGAs are customisable silicon devices which can be used to implement arbitrary hardware datapaths and can be reconfigured when needed. Hence, they represent a platform that can provide the high performance of custom processing datapaths that exploit algorithmic parallelism, with the flexibility of reconfiguration at run-time. Moreover, Xilinx FPGAs can be partially reconfigured; one part of the device can be reconfigured while the other continues to function – a clear enabler for cognitive radios.

We base our system architecture on the Iris software radio platform introduced in Section II-A3. Iris has been extensively re-engineered to adhere to the model presented in Section III, and allows for the incorporation of hardware as well as software processing components [10], [11].

The data plane is implemented as a combination of software and hardware components, and described in an XML format. In the case of hardware, it is implemented in the FPGA fabric, in what we term *customisable processing regions*. These can be reconfigured at run-time using FPGA partial reconfiguration. Software components, as well as the Iris Runtime System, run on the *processor subsystem*. This contains all the basic hardware required to run the Linux operating system on an embedded processor. It remains active throughout the lifetime of the system and can be programmed in software.

The architecture is flexible in terms of the capabilities of the FPGA used and the number of customisable processing regions present; the transformations discussed in Section III-E enable the same radio model to be mapped to alternatively configured FPGAs. An FPGA architecture with one customisable processing region is shown in Fig. 4.

The control plane of the radio model runs the *Decision Engine* in the Iris software radio. This is implemented in C++, and uses an API to subscribe to component events and perform reconfigurations of the data plane. The *Iris Runtime System* manages the radio execution and provides the API for the Decision Engine. Fig. 5 shows an outline of the system architecture.

The library of available processing components details their input/output characteristics including data types and can be used by the designer to implement whichever waveforms are needed. During radio design, it does not matter whether these components are implemented in hardware or software.

C. Data Plane Implementation

In this framework, we distinguish between the design of processing components and their composition into radios. At present, a large number of components are available in the Iris component library as software, with a growing number of hardware components being developed. It is possible to add custom hardware or software components to the library.

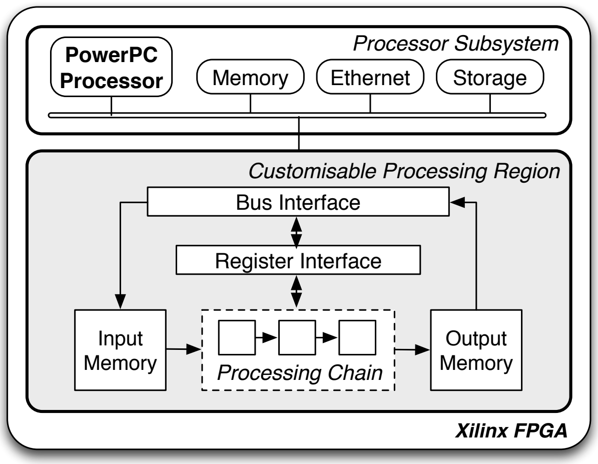


Fig. 4. The FPGA architecture, showing the processor subsystem with the basic components to run Linux, and one customisable processing region to hold radio processing chains.

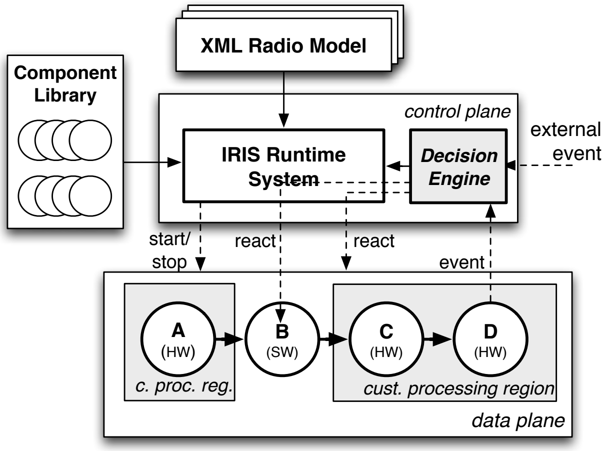


Fig. 5. The run-time system architecture for a target with two customisable processing regions.

Components must implement a simple interface, either in software or in hardware. Software components are implemented in C++ using class inheritance. The interface for hardware components consists of FIFO signals for the input and output ports and signals for triggering events and setting parameters. We do not specify how hardware components should be implemented internally; designers may use standard hardware description languages or high-level design tools (e.g., AccelDSP for MATLAB-based design on Xilinx FPGAs).

Since Iris supports both software and hardware components, we can combine the two within a single system, and this provides a powerful verification and debugging mechanism. It is possible to process data through hardware, then store it in a file for offline analysis, or transfer the data for visualisation in MATLAB. This is also beneficial when developing a custom hardware component, since this data can be be used to verify the implementation against software versions.

D. Control Plane Implementation

The operations of the control plane are, by definition, control-driven and are hence amenable to implementation in software; the performance benefit in moving such operations into hardware is marginal and likely to be outweighed by the complexity of doing so. Thus, we implement the control plane in software.

The control plane in the Iris framework consists of two parts, the Decision Engine and the Iris Runtime System, as shown in Fig. 5. The Decision Engine implements the cognitive decision making process of the radio, and uses a simple API to subscribe to component events and perform reconfiguration actions, as described in Section III-C. This API is provided by the Iris Runtime System, which manages the radio execution and performs the low-level actions associated with reconfiguration commands.

The hierarchical radio model (see Section III) for execution by Iris is described in XML form. The Iris Runtime System parses the specified XML file, instantiates and connects the required components, loads the Decision Engine, and controls and monitors system execution.

E. Implementing Reconfiguration

As described in Section III-B, reconfigurations can be either a switch of the active mode within a subsystem (a functional or structural reconfiguration), or a reconfiguration of a component parameter (parametric reconfiguration). Reconfigurations in software are straightforward and do not need special treatment. In the following we focus on reconfigurations involving hardware components.

Mode switches are facilitated through reconfiguration of the FPGA; swapping modes for a subsystem implies reconfiguring the corresponding customisable processing region. When the number of available regions in the platform is sufficient to accommodate the number of subsystems in the radio being modelled, configurations have a one-to-one mapping. Consider now, the case where the number of regions in the platform is less than the number required for the high level radio model. In such a case, we can proceed to transform our radio model using the transformations given in Section III-E until we have the same number of subsystems containing only hardware components as customisable processing regions on the FPGA. These hardware subsystems cannot contain further subsystems and must therefore be flattened using the transformation rules. Extra subsystems are allowed, but can only contain software components.

Parametric reconfigurations are less straightforward. If the parameter to be reconfigured can be mapped to a simple hardware parameter, e.g. the gain parameter of a scaling component, the software writes the new parameter value into a hardware register. The hardware reads this register and changes its behaviour accordingly. If a parameter to be changed implies considerable changes in the hardware circuitry, e.g. the generating polynomial of a Viterbi decoder, a subsystem must be generated during model synthesis with a replicated Viterbi decoder, each with a different polynomial (applying transformation rule 2) in Section III-E). That is, when the generating polynomial is to be changed, the hardware needs to be reconfigured, as represented by the modes in the subsystem.

Our aim is to maintain the interface to the Decision Engine as defined in the high-level model during design. Radio designers should not concern themselves with the transformations that have to be applied to the model to fit the target platform, and it should be transparent to the designer whether components run in software or hardware. This requires a translation layer, which converts high-level events and reconfiguration commands to the low-level equivalents that execute on the target platform.

Events can be triggered from the hardware in the customisable processing region by setting a value in a register. The translation layer reads these registers and triggers a software event to the Decision Engine which can then react. Reconfiguration actions from the Decision Engine are converted into low-level reconfiguration actions by the translation layer, performing hardware reconfigurations (mode switches) or register write operations as needed.

F. Automatic Radio Model Synthesis

For our target platform, the user implements the Decision Engine using the Control Specification and a simple API in C++. It can subscribe to events published by components, and apply functional, structural or parametric reconfiguration to the data plane.

The only two inputs that the radio developer needs to specify are: the XML radio model description, including the Control Specification; and the C++ Decision Engine. The Decision Engine does not need to consider implementation details of the components it is controlling (i.e., software or hardware), since the interfaces are abstracted.

Since the Decision Engine is implemented in software, cognitive algorithms are limited by the processing power and capabilities of the embedded processor. However, the designer has the flexibility to interface the Decision Engine with external event triggers, other hardware, or alternative cognitive engines.

These two inputs are given to the platform synthesis tool, which we have named the *Composer*. It uses the Control Specification, the XML description of the radio model, and the component library to generate the FPGA configurations. First, it determines which components to implement in hardware, based on the availability of hardware implementations, the available hardware resources, and performance constraints. It then applies the model transformation rules given in Section III-E to fit the FPGA architecture (as explained in Section IV-E). It then generates the required FPGA configurations, the translation layer as outlined above, and an XML file for the Iris Runtime System. The radio is now ready to be executed on the FPGA-based platform.

G. Summary

We showed how the platform-independent model presented in Section III can be mapped to a real implementation platform. We are able to use the transformations presented in Section III-E to make the model amenable to implementation on a wide variety of platforms. The high-level model allows designers to develop and reason about cognitive radio applications while being decoupled from specific implementations. We showed our custom design flow for mapping models to FPGA platforms. Since the model captures the information required for automatic model synthesis, developers do not need to concern themselves with low-level details. We also showed how the model facilitates the incorporation of new processing components and complex control capability, making it scalable to different types of systems.

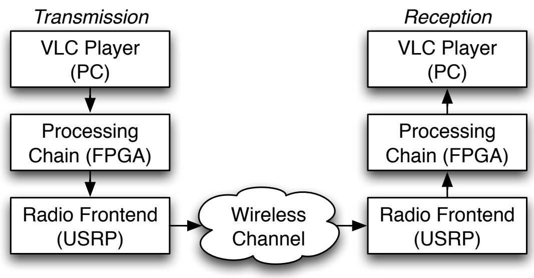


Fig. 6. Demonstration setup. We used the Xilinx University Program (XUP) Virtex-II Pro FPGA board for Section V-A, the Xilinx ML507 board with a Virtex 5 FPGA for Section V-B, and the USRP radio frontend [35].

V. Example Radio Applications

There is a clear trend towards more adaptability in emerging wireless communication standards, where several radio parameters are changed at run-time according to channel quality. Examples are LTE [31] and WiMAX [32], where the coding and modulation schemes are adapted.

In this section, we apply the proposed cognitive radio design methods to the development of a similar radio which can adapt its coding method to current channel conditions. We then extend that system to include spectrum sensing at the receiver. The application data is a video stream, provided by VLC Player through its UDP interface. The implementation platform is the Xilinx University Program (XUP) board [33] which hosts a Xilinx Virtex-II Pro FPGA for the adaptive coding demonstrator. The design is then ported to the Xilinx ML507 board [34] hosting a more powerful Virtex 5 FPGA and extended with sensing. We use the Universal Software Radio Peripheral (USRP) radio frontend [35] which we connect to through a TCP socket via a PC bridge. This was necessary due to a lack of USB 2.0 connectors on the development boards. Figure 6 shows an overview of the demo setup used for the examples below.

A. Adaptive Coding Link

For simplicity, we have chosen a simple point-to-point link, where the transmitter encodes video data using convolutional codes with rate 1/2, and the receiver decodes the data using a Viterbi decoder. The error correction capability of a code improves with larger constraint lengths. On the other hand, larger constraint lengths result in much more complex implementations of the Viterbi decoder, resulting in significant increases in area usage and power consumption on the FPGA [10]. Therefore we have designed a system to switch

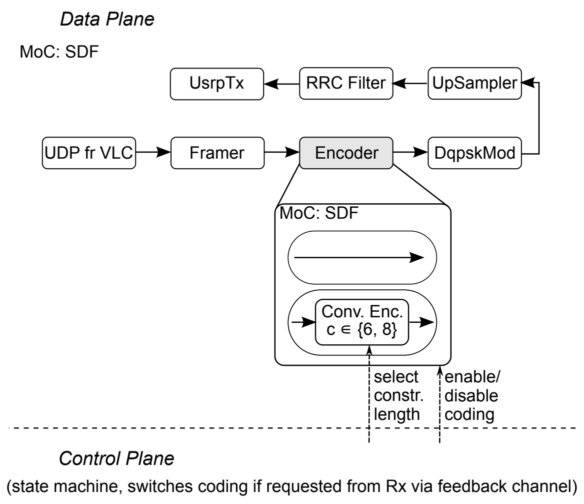


Fig. 7. The adaptive coding transmitter model. Either no coding, or coding with constraint length 6 or 8 is applied. The controller obtains information about which coding method to apply from the receiver through a feedback channel.

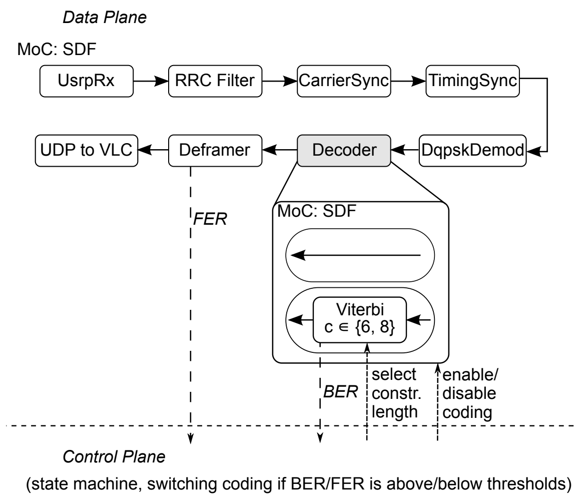


Fig. 8. The adaptive coding receiver model. Either no decoder, or decoders with constraint lengths 6 or 8 can be used.

between codes with constraint lengths 6 and 8, as well as no coding, always selecting the least complex scheme that can still achieve a target bit error rate (BER) or frame error rate (FER), in order to minimise power dissipation.

1) Transmitter: Fig. 7 shows the transmitter radio model. It operates as follows. Data is passed from the video source to the transmission chain via UDP (a VLC Player is running on a laptop connected to the Tx FPGA board). The *Framer* adds a 64-bit frame access code to each frame of data to identify it, and appends a checksum. The *Convolutional Encoder* encodes the data with 1/2 code rate and a reconfigurable constraint length code. The *Differential QPSK Modulator* maps information dibits (2-bit units) into phase changes in the complex base band signal. The *UpSampler* and *RRC Filter* (root-raised cosine filter) act as a pulse shaper, used to control the transmission bandwidth and signal-to-noise ratio (SNR). Data is transmitted to the receiver by the *UsrpTx* component, which controls the USRP radio front-end used for this demonstration [35].

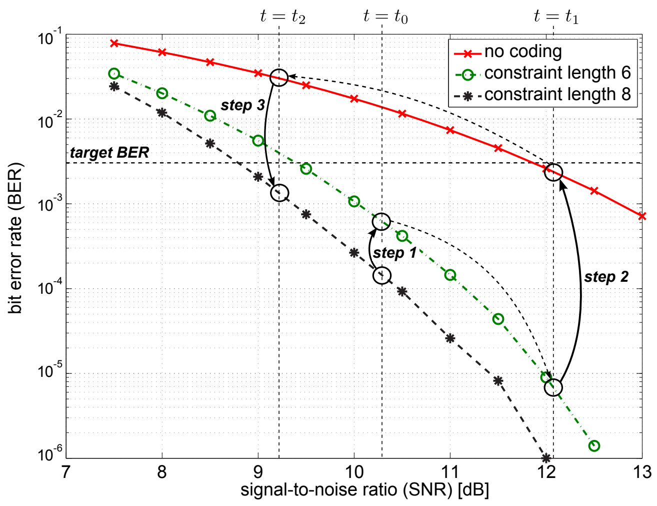


Fig. 9. Simulated BER vs. SNR for different codes, and an example code switching sequence.

2) Receiver: The receiver model is shown in Fig. 7. Data is obtained from the USRP radio frontend. It passes through the *RRC Filter*, which acts as a matched filter, increasing the SNR. When a modulated signal is received, the frequency of the local oscillator is typically off by some variable margin, which is corrected by the *Carrier Synchronisation* component. *Timing Synchronisation* finds the correct sampling points for each symbol, eliminating the effects of timing shifts and jitter. The *Differential QPSK Demodulator* is then used to map the complex signal samples back into information dibits. The *Viterbi Decoder* decodes the data, using codes that match the convolutional coder at the transmitter. The *Deframer* correlates the fixed 64-bit frame access code inserted at the transmitter with the data, in order to identify the start of a frame. It extracts the data from the frames and performs checksum verification. The data is then passed to a VLC player running on a separate laptop via UDP, which plays the received video stream.

3) Radio Controller: The radio controllers work as follows. At the receiver side, the controller subscribes to events holding the current BER and FER and implements a simple state machine which changes the active coding configuration depending on pre-defined BER/FER thresholds. When a new coding scheme is selected, the transmitter is informed through a feedback channel. In this example we implemented the feedback channel using Ethernet for simplicity, though this can be done wirelessly. The transmitter controller reconfigures the radio according to the information received on this feedback channel.

Figure 9 shows simulated BER vs. SNR curves. The thresholds for switching the coding scheme are chosen based on these curves, so that the target BER is maintained.

The figure also illustrates an example adaptation sequence. The system starts in the most robust coding state ($c = 8$) at time $t = t_0$. The controller monitors the BER output of the Viterbi decoder at the receiver. The threshold for switching to less robust coding is passed at time $t = t_1$, so it initiates a parametric reconfiguration by setting the c parameter of the Encoder and Decoder blocks to 6 (step 1 in Fig. 9). At time $t = t_1$, the signal quality increased so that the BER is lower

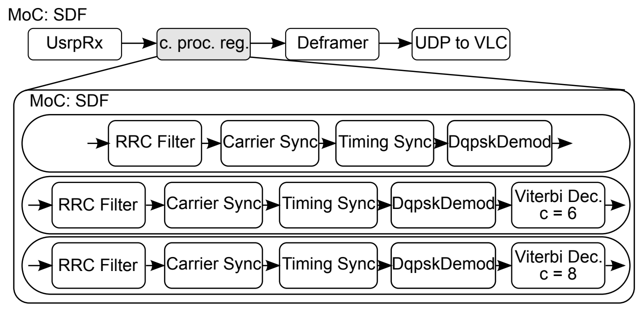


Fig. 10. Adaptive coding receiver data plane synthesised for an FPGA platform with one customisable processing region. A mode switch is mapped to hardware reconfiguration.

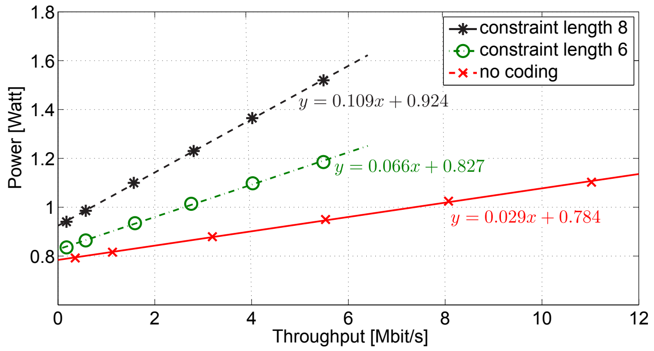


Fig. 11. Measured power consumption with linear regression curves.

than the threshold. The controller disables coding (step 2 in Fig. 9). At time $t = t_2$, the channel conditions have worsened considerably and the BER has risen above the threshold, so the Encoder and Decoder are reconfigured to the constraint length 8 code (step 3 in Fig. 9). In this manner, the system continues to use the most suitable type of coding for the current channel conditions.

4) Model Synthesis: At this point in the design, the model can be checked for correctness using the methods described in Section III-D. The XML model description and the Control Specification are given to the model checking tool, which performs the correctness checks. Once successful, the Decision Engine for transmitter and receiver are implemented as outlined above, and the Composer tool is used to synthesise the designs for our FPGA-based platform running Iris.

In this example, the FPGA is configured to have only one customisable processing region. The component library holds efficient hardware implementations of the computational complex components, i.e., the encoder, decoder, synchronisation, modulation, demodulation, and filter components. The Composer applies the model transformations given in Section III-E and synthesises the data plane. The synthesised receiver is shown in Fig. 10, the transmitter is synthesised similarly. It further generates the required bitstreams for the FPGA and a translation layer mapping the events and reconfiguration commands from the controllers to hardware register read and write operations and hardware reconfiguration commands.

The resulting configurations are ready to be executed on our FPGA-based Iris platform. The radio designer did not have to deal with low-level implementation details, the radio model was checked for errors before synthesis, and the radio can be executed on a real system.

5) Results: A throughput of 6.25 Mbit/s with coding is achievable if the radio front-end is integrated into hardware. Since the USRP front-end is connected to the XUP board via an Ethernet bridge through a PC, the performance at the Framer/Deframer was limited to 0.5 Mbit/s.

In order to quantify the power benefits of adaptation, we have measured the *average* power consumption of the three receiver radio chain configurations (i.e., no coding and coding with constraint lengths 6 and 8). Figure 11 shows the power consumption when increasing the throughput for the complete FPGA; it increases linearly with throughput. Additionally, the graph shows the large impact constraint length has on power consumption given the increased amount of FPGA resources required. As we can see in the figure, at a throughput of 6 Mbit/s, the constraint length 8 implementation consumes approximately 1.6 W. If the received SNR improves to the point where a non-coding radio chain implementation can provide the required BER, an adaptive reconfiguration can reduce power consumption to under 1 W, a saving of 60 %.

This adaptive coding system was demonstrated at FCCM 2009 [10].

B. Extending the Radio With Spectrum Sensing

Extending the above radio to more complex systems is a relatively easy task using the proposed design methods. We need to add a sensing mode to the receiver, for searching for the transmission frequency without prior knowledge.

1) Receiver: To achieve that, the receiver is modified to include the USRP receive component and a single subsystem at the top level. That subsystem contains two modes, one for normal reception, containing the full receiver radio for adaptive coding as described above, and another for sensing. This sensing mode consists of a component to estimate the power spectral density (PSD) and a detector component for detecting the transmitter's frequency.

2) Transmitter: The transmitter is only modified slightly. The controller changes the transmit frequency in the UsrpTx component randomly every 30 seconds to demonstrate the sensing receiver functionality. Thus the transmitter model is similar to Fig. 7.

3) Radio Controller: The controller at the transmitter is unmodified from Section V-A. The controller at the receiver subscribes to a *LostSignal* event published by the Deframer, which is triggered when no frames are found. It also subscribes to the *CarrierFreq* event published by the Detector component, which holds the estimated transmit frequency. If the *LostSignal* event is triggered, the controller switches to sensing mode, and tries to locate the transmit frequency. Once found (the *CarrierFreq* event is received), it reconfigures the frequency in the UsrpRx component and switches back to receive mode. During receive mode, the adaptive coding scheme is applied as described above.

Figure 12 shows the receiver radio model.

4) Model Synthesis: The component library for the Iris/FPGA target platform contains an efficient hardware implementation of a PSD estimator component, while the detector is available in software. Thus, Composer synthesises

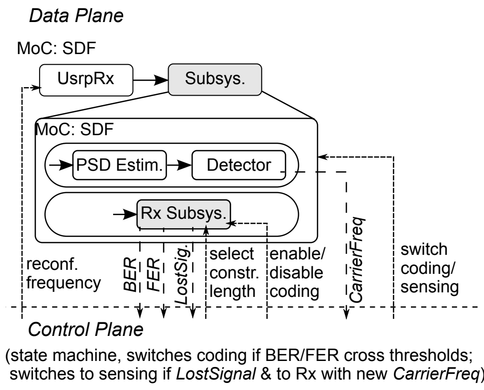


Fig. 12. Adaptive coding receiver with sensing mode. The *Rx Subsystem* is the full radio shown in Fig. 8.

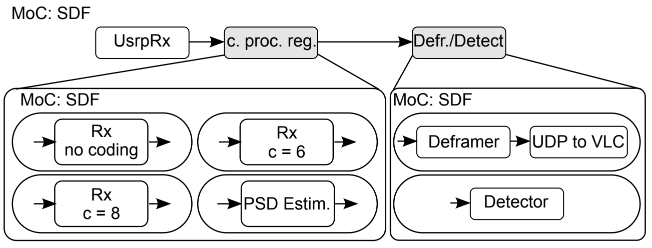


Fig. 13. Synthesised adaptive coding receiver with sensing mode (data plane). The model is transformed to map to one customisable processing subsystem on the FPGA and another in software. The Rx systems are the same as shown in Fig. 10, the Deframer/Detector subsystem is running in software.

the system as shown in Fig. 13, applying the transformations discussed in Section III-E. Again, only one customisable processing region is used. A second subsystem running in software is generated, to hold the software components at the end of the radio chains.

This demonstrator was implemented on a different board, the Xilinx ML507, featuring a newer Virtex-5 FPGA, showing the portability of our design approach. No changes to the model were neccessary to synthesise for this new board.

5) Results: The combined sensing and adaptive coding demonstrator is able to follow frequency changes of the transmitter with minimal interruption of the video stream. It adapts its coding scheme to the current channel conditions, always selecting the code with the lowest power consumption while ensuring the BER is below the target. This demonstrator combines software and hardware components, a mode-switching radio controller, and uses FPGA partial reconfiguration, yet, it was composed from a high-level platform-independent model automatically.

This combined adaptive coding and sensing radio has been demonstrated at SIGCOMM 2009 [36].

C. Summary

As shown by these two simple example applications, the proposed model-based design technique is suitable for developing real cognitive radios. It can be used for FPGA-based embedded systems, combining hardware and software components, as well as for pure software platform. Extending a radio is a simple task due to the re-usability of model subsystems and automatic mapping to physical subsystems.

VI. Conclusion

In this paper, we identified some major challenges faced when using existing cognitive/software radio platforms for the development of and experimentation with cognitive radios. We have presented a novel approach to developing such radios, based on a high-level radio model. The model describes radio behaviour, including possible reconfigurations, independent of the target platform. Correctness checks have been presented, allowing error detection at design-time. It has been shown that the model can be transformed to fit a variety of target platforms.

Automatic tools take the high-level model and synthesise a specific implementation for a chosen target platform. We chose an FPGA-based port of the Iris software radio as a demonstrator platform. An adaptive coding and sensing cognitive radio was successfully developed using the proposed techniques, underlining the applicability to real systems.

With this approach, radio developers do not require detailed platform knowledge, the development process requires less manual effort, and the design can be easily ported to other target platforms. Hence, researchers are able to test their algorithms in the real world with significantly reduced effort. We believe that this contribution will drive cognitive radio research forward significantly.

References

[1] *IEEE Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management*, IEEE Std. 1900.1-2008, Sep. 2008.

[2] J. Mitola III, “Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio,” Ph.D. dissertation, Royal Institute of Technology, Stockholm, Sweden, May 2000.

[3] S. Haykin, “Cognitive radio: Brain-empowered wireless communications,” *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 201–220, 2005.

[4] R. W. Thomas, L. A. DaSilva, and A. B. MacKenzie, “Cognitive networks,” in *IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, Nov. 2005, pp. 352–360.

[5] Free Software Foundation, Inc. (2009) GNU Radio - the GNU software radio. [Online]. Available: http://www.gnu.org/software/gnuradio/

[6] C. R. A. Gonzalez, C. B. Dietrich, S. Sayed, H. I. Volos, J. D. Gaeddert, P. M. Robert, J. H. Reed, and F. E. Kragh, “Open-source SCA-based core framework and rapid development tools enable software-defined radio education and research,” *IEEE Commun. Mag.*, vol. 47, no. 10, pp. 48–55, Oct. 2009.

[7] Joint Tactical Radio System (JTRS) Joint Program Executive Office (JPEO), “Software communications architecture specification,” JTRS Standards, final, version 2.2.2, 15 May 2006. [Online]. Available: http://jtrs.spawar.navy.mil/sca/downloads.asp?ID=2.2.2

[8] C. R. A. González, “Design and implementation of an efficient SCA framework for software-defined radios,” M.Sc. thesis, Virginia Tech., 2006.

[9] P. MacKenzie, “Software and reconfigurability for software radio systems,” Ph.D. dissertation, Trinity College Dublin, Ireland, 2004.

[10] S. Fahmy, J. Lotze, J. Noguera, L. E. Doyle, and R. Esser, “Generic software framework for adaptive applications on FPGAs,” in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 2009.

[11] J. Lotze, S. A. Fahmy, J. Noguera, B. Özgül, L. E. Doyle, and R. Esser, “Development framework for implementing FPGA-based cognitive network nodes,” in *IEEE Global Communications Conference (GLOBECOM)*, Honolulu, Hawaii, USA, Dec. 2009.

[12] P. D. Sutton, J. Lotze, H. Lahlou, S. A. Fahmy, K. E. Nolan, B. Özgül, T. W. Rondeau, J. Noguera, and L. E. Doyle, “Iris - an architecture for cognitive radio networking testbeds,” *IEEE Commun. Mag.*, vol. 48, no. 9, Sep. 2010.

[13] K. Amiri *et al.*, "WARP, a unified wireless network testbed for education and research," in *IEEE International Conference on Microelectronic Systems Education (MSE)*, San Diego, CA, USA, 3–4 Jun. 2007.

[14] (2009) WARP FPGA Board – Hardware Version 2.2. Mango Communications, Inc. [Online]. Available: http://mangocomm.com/products/boards/warp-fpga-board-v2

[15] R. Sathappan, M. Dumas, L. Belanger, and M. Uhm, "New architecture for development platform targeted to portable applications," in *SDR Forum Technical Conference (SDR)*, Orlando, Florida, USA, 2006.

[16] J. Jacob and M. Dumas, "CORBA for FPGA the missing link for SCA radios," Lyrtech, Quebec, Canada, white paper, Feb. 2007.

[17] G. J. Minden *et al.*, "KUAR: A flexibile software-defined radio development platform," in *Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, Dublin, Ireland, 17-22 Apr. 2007.

[18] Z. Miljanic, I. Seskar, K. Le, and D. Raychaudhuri, "The WINLAB network centric cognitive radio hardware platform – WiNC2R," *Mobile Networks and Applications*, vol. 13, no. 5, pp. 533–541, Oct. 2008.

[19] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," *Proc. IEEE*, vol. 91, no. 1, pp. 127–144, Jan. 2003.

[20] *Simulink 7*, The MathWorks, Inc., Sep. 2007, no. 9320v06. [Online]. Available: http://www.mathworks.com/mason/tag/proxy.html?dataid=9798&fileid=43815

[21] D. Silage, *Digital Communication Systems Using SystemVue*. London, UK: Charles River Media, Jan. 2006.

[22] E. A. Lee, "Embedded software — an agenda for research," University of California at Berkely, UCB ERL Memorandum M99/63, Dec. 1999.

[23] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing '74: Proceedings of the IFIP Congress*, New York, USA, 1974, pp. 471–475.

[24] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proc. IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.

[25] P. D. Sutton, B. Ozgul, K. E. Nolan, and L. E. Doyle, "Bandwidth-adaptive waveforms for dynamic spectrum access networks," in *3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, 14–17 Oct. 2008, pp. 1–7.

[26] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[27] ——, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 24–35, Jan. 1987.

[28] J. Mitola III and G. Q. Maguire, Jr., "Cognitive radio: Making software radios more personal," *IEEE Personal Commun.*, vol. 6, no. 4, pp. 13–18, 1999.

[29] A. He, K. K. Bae, T. R. Newman, J. Gaeddert, K. Kim, R. Menon, L. Morales-Tirado, J. Neel, Y. Zhao, J. H. Reed, and W. H. Tranter, "A survey of artificial intelligence for cognitive radios," *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1578–1592, 2010.

[30] C. A. Petri, "Communication with automata," Reconnaisesance-Intellegence Data Handling Branch, Rome Air Development Center, New York, suppl. I to tech. rep. no. RADC-TR-65-377, Jan. 1966.

[31] J. Zyren and W. McCoy, "Overview of the 3GPP long term evolution physical layer," Freescale Semiconductor, Inc., white paper, Jul. 2007.

[32] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX – Understanding Broadband Wireless Networking*, ser. Communications Engineering and Emerging Technologies, T. S. Rappaport, Ed. New Jersey, USA: Prentice Hall, 2007.

[33] *Xilinx University Program Virtex-II Pro Development System – Hardware Reference Manual*, Xilinx, Inc., San Jose, CA, USA, 8 Mar. 2005, UG069 v1.0. [Online]. Available: http://www.xilinx.com/univ/XUPV2P/Documentation/XUPV2P_User_Guide.pdf

[34] *ML505/ML506/ML507 Evaluation Platform: User Guide*, Xilinx, Inc., San Jose, CA, USA, 7 Oct. 2009, UG347 v3.1.1. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[35] *Universal Software Radio Peripheral – The Foundation for Complete Software Radio Systems*, Ettus Research LLC, Mountain View, California, USA, Nov. 2006. [Online]. Available: http://www.ettus.com/downloads/usrp_v4.pdf

[36] J. Lotze, S. A. Fahmy, L. E. Doyle, and J. Noguera, "An FPGA-based autonomous adaptive radio," in *ACM SIGCOMM Conference*, Barcelona, Spain, Aug. 2009, demonstration paper.

Jörg Lotze received his Dipl-Ing. degree in Computer Engineering from the Ilmenau University of Technology, Germany in 2006. He is currently in his final year as a PhD student with CTVR, the Telecommunications Research Centre, at Trinity College Dublin. He has been actively involved in research and development of cognitive radio systems since 2007. His work has addressed key challenges in cognitive radio design and experimentation and cognitive radio platform architectures.

Suhaib A. Fahmy received the MEng degree in Information Systems Engineering and the PhD degree in Reconfigurable Computing from Imperial College London in 2003 and 2007 respectively. From 2007 to 2009, he was a Research Fellow at the University of Dublin, Trinity College, and Visiting Research Engineer at Xilinx Research Labs in Ireland. He joined Nanyang Technological University as Assistant Professor in the School of Computer Engineering in late 2009. His research interests include reconfigurable computing, high-level system design and computational acceleration of complex algorithms.

Juanjo Noguera received his B.Sc. degree in Computer Science from the Autonomous University of Barcelona, Spain, in 1997, and his PhD degree in Computer Science from the Technical University of Catalonia, Barcelona, Spain, in 2005. He has worked for the Spanish National Centre for Microelectronics, the Technical University of Catalonia, and Hewlett-Packard Inkjet Commercial Division. Since 2006, he has been with the Xilinx Research Labs, Ireland. His interests include high-level system design, reconfigurable architectures and next-generation wireless communications.

Professor Linda E. Doyle is a member of faculty in the School of Engineering, University of Dublin, Trinity College, Ireland. She is currently the Director of CTVR, the Telecommunications Research Centre, a national research centre headquartered in Trinity College and based in five other universities in Ireland. CTVR carries out industry-informed research in the area of telecommunications and focuses both on wireless and optical communication systems. Prof. Doyle is responsible for the direction of the CTVR as well as running a large research group that is part of the centre, which focuses on cognitive radio, reconfigurable networks, spectrum management and telecommunications and digital art.