Iris: An Architecture for Cognitive Radio Networking Testbeds

Paul D. Sutton, Jörg Lotze, and Hicham Lahlou, Trinity College Dublin
Suhaib A. Fahmy, Nanyang Technological University
Keith E. Nolan and Barış Özgül, Trinity College Dublin
Thomas W. Rondeau, IDA/CCR
Juanjo Noguera, Xilinx Research Labs
Linda E. Doyle, Trinity College Dublin

ABSTRACT

Iris is a software architecture for building highly reconfigurable radio networks. It has formed the basis for a wide range of dynamic spectrum access and cognitive radio demonstration systems presented at a number of international conferences between 2007 and 2010. These systems have been developed using heterogeneous processing platforms including general-purpose processors, field-programmable gate arrays and the Cell Broadband Engine. Focusing on runtime reconfiguration, Iris offers support for all layers of the network stack and provides a platform for the development of not only reconfigurable point-to-point radio links but complete networks of cognitive radios. This article provides an overview of Iris, presenting the unique features of the architecture and illustrating how it can be used to develop a cognitive radio testbed.

INTRODUCTION

Over the past five years, the Iris architecture has formed the basis for demonstration systems addressing key challenges in dynamic spectrum access and cognitive radio networking. These have included:

- An opportunistic dynamic spectrum access network using novel physical (PHY)-layer signaling techniques for network rendezvous and coordination
- A TV white space network using orthogonal frequency-division multiplexing (OFDM)-based waveforms in licensed test spectrum awarded by Comreg, the Irish communications regulator
- A reconfigurable field programmable gate array (FPGA)-based system employing sensing and using a single carrier waveform with adaptive coding
- A real-time cyclostationary analyzer implemented using the Cell Broadband Engine (CellBE) platform
- A demonstration of coexistence between reconfigurable non-contiguous OFDM-based and co-channel single-carrier waveforms.

These systems have served to verify theory-based research and have been demonstrated at a number of key international conferences on cognitive radio and reconfigurable radio systems. In this article we provide an overview of the architecture and take a closer look at examples in which the unique features of Iris have been exploited to realize highly reconfigurable radio networks.

Modern radio communication standards and protocols feature increasing flexibility and reconfigurability as designers strive to use the resources available in the most efficient manner possible. From the self-organization and coexistence capabilities required by Third Generation Partnership Project Long Term Evolution (3GPP LTE) femtocells to the dynamic spectrum access of emerging cognitive radio standards such as IEEE 802.22, the ability of network nodes to examine the operating environment and reconfigure accordingly is becoming more and more important.

With increasing system flexibility and automation come the challenges of interference mitigation and coexistence both within a single homogeneous network and between heterogeneous networks. These are challenges that cannot be overcome at design time alone, so there is a need for testbeds to play a greater role in the development, testing, and verification of these systems.

Through real-world experimentation, we can develop reconfigurable network designs to ensure their stability and verify that they do not cause harmful interference to other spectrum users. This, in turn, informs regulators of the practical limitations of such systems, promoting the use of fair and efficient spectrum management regimes.

The following sections provide an overview of Iris, a software architecture designed from the

This material is based on work supported by Science Foundation Ireland under Grant no. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain Research (CTVR) at the University of Dublin, Trinity College, Ireland, and by Enterprise Ireland under its Innovation Partnership scheme, project IP20060367.

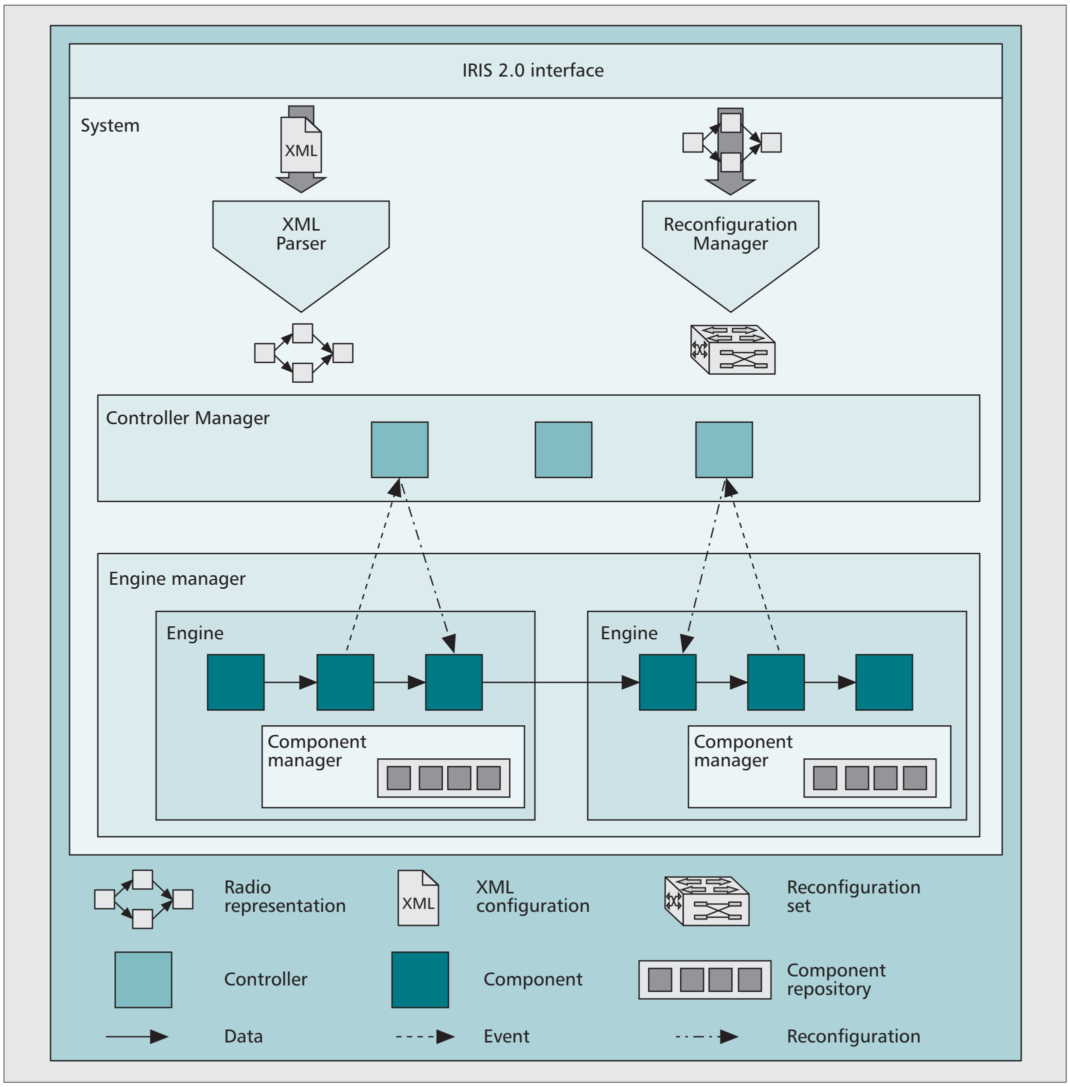

Figure 1. *Key elements of the Iris architecture.*

When implemented in software, any part of the radio becomes dynamically reconfigurable. With the addition of a software radio controller, this reconfigurability can be harnessed to enable concepts such as dynamic spectrum access and cognitive radio networks.

ground up for the implementation of runtime reconfigurable radio network testbeds. Key features of the Iris architecture include:

- Full support for runtime reconfiguration of a radio
- Support for all layers of the network stack, not just the PHY layer
- A well defined interface to controllers, decision making processes that reconfigure the radio in response to observations of the environment and the radio itself
- Support for advanced processing platforms including FPGAs and the CellBE.
- Portable C++ for multiple operating systems and CPU architectures

The Iris Architecture: Reconfigurable Radio

A key motivation for software radio is the flexibility afforded by general-purpose processing platforms. When implemented in software, any part of the radio becomes dynamically reconfigurable. With the addition of a software radio controller, this reconfigurability can be harnessed to enable concepts such as dynamic spectrum access and cognitive radio networks. In designing the Iris software radio architecture, the fundamental design goal was thus to maximize the reconfigurability of the system. The following sections present the mechanisms provided to realize this flexibility.

Core Architecture

Iris was originally designed and implemented in 2004 [1]. In 2008 the architecture was completely redesigned in order to provide increased flexibility, support higher layers of the network stack, and leverage the parallel processing capabilities of emerging processor platforms. As in software radio architectures such as GNU Radio [2] and the Software Communication Architecture (SCA) [3], Iris is designed to be component-based. Discrete signal processing functions such as a digital filter or modulator are implemented as components with generic interfaces for life-cycle control, data passing, and reconfiguration. These components are linked together to build radio transmit and receive chains, which are in turn connected to higher layers of the network stack to create the complete network node. In

A switch component can be used to direct the flow of data in the system to one of two or more signal processing branches. A parameter exposed on the component specifies the active output port to which data is written, and thus the active signal processing branch.

Iris these structures of components are represented using Extensible Markup Language (XML). XML has the advantage of being human-readable while unambiguously capturing the complex flow graph structures that can be supported by Iris. In addition, XML allows us to use a simple interface for external reconfiguration of a running radio.

In building demonstration systems using the first version of Iris (v1.0), a number of key lessons were learned, which motivated the redesign of the system in 2008. Among these lessons was the need for flexibility in data passing mechanisms to support a high degree of component reconfiguration. Reconfiguration of a component can often change the rate at which that component consumes and produces data. As these dynamic rate changes may not be predicted, they must be supported by the underlying architecture. This flexibility was a key design goal for Iris 2.0. A second lesson was that no single mechanism for data passing, component execution, and reconfiguration is sufficient to support all parts of a reconfigurable network node. This lesson prompted the introduction of multiple *Iris engines*. Iris engines define a particular domain within which components can execute, and a single network node design may consist of more than one type of engine. Different engines support different mechanisms for data passing, execution, and reconfiguration, providing a high degree of flexibility for the system designer. Engines are key to supporting higher layers of the network stack in Iris, leveraging emerging processing platforms and making the Iris core architecture modular and extensible. The next section provides more detail about the Iris engines. In designing Iris 2.0, many of the weaknesses of Iris 1.0 were addressed; however, the key strengths of the architecture were preserved. These strengths include the focus on reconfigurability and the modular component-based design.

Both the Iris runtime system and the Iris components are implemented completely in portable C++, simplifying code management, permitting straightforward debugging of a running radio, and supporting cross-platform development. Figure 1 illustrates the principal elements that constitute the Iris architecture. These are examined in more detail in the following sections.

The Key Design Goal of Iris: Reconfiguration

In contrast with software architectures such as GNU Radio and the SCA, Iris is designed specifically to support maximum reconfigurability while the radio is running. This permits a network node to carry out reconfigurations seamlessly in response to observed changes in the operating environment. This reconfigurability is realized through a number of mechanisms that were built into the Iris architecture. The first of these is the *component parameter*. When implementing an Iris component, the designer can choose to expose a number of parameters. While the radio is running, these parameters can be dynamically reconfigured to adjust the operation of the component. For example, an OFDM modulator may expose a parameter that controls the length of cyclic prefix used on modulated symbols. Both the IEEE 802.16e [4], and the 3GPP LTE [5] standards use this approach to adjust the robustness of modulated symbols to multipath channel delay. In Fig. 1 the reconfiguration of component parameters is illustrated by a dashed and dotted line.

Another powerful way in which component parameters can be used is to implement dataflow switching. A switch component can be used to direct the flow of data in the system to one of two or more signal processing branches. A parameter exposed on the component specifies the active output port to which data is written, and thus the active signal processing branch. This approach can be used, for example, in a receiver with independent receive chains for demodulation, and for signal detection and classification. When searching for a signal of interest, the flow of data is directed to the processing chain for signal detection and classification. When a signal is detected and classified, the switch component is reconfigured to direct the flow of data to the demodulation chain for data extraction.

In addition to the mechanisms provided to support parametric and structural reconfiguration, the Iris architecture provides specific support for the *triggers* which determine when these reconfigurations must take place. The first type of trigger that can occur within a radio is *internal* and typically takes place in response to a change detected by one of the radio components. Such a trigger could be used, for example, by a digital television (DTV) or wireless microphone signal detection component within an IEEE 802.22 system to prompt reconfiguration of the network and avoid the creation of harmful interference. In order to support such a trigger, Iris provides support for component *events*, illustrated by a dashed line in Fig. 1. These events are specified by the component designer and may be triggered by the component at any time.

The elements of the Iris architecture with responsibility for listening for and responding to these events are the *controllers*. Controllers have a global view of the running radio and are capable of reconfiguring all aspects of it in response to component events. The Controller Manager is illustrated in Fig. 1, and is responsible for loading and managing the life cycle of controllers within Iris. In addition to reconfiguring component parameters, controllers can adjust the structure of a running radio by inserting and removing components. Like components, controllers are implemented in portable C++ and are loaded into a radio according to the XML configuration file. A key advantage of using controllers in this way is the avoidance of inter-component dependencies. By ensuring that components remain independent of one another, maximum reusability can be achieved. Controllers may consist of simple reconfiguration responses to predefined events or they may be much more complex entities, listening for multiple events, monitoring the state of the overall radio, reconfiguring many different aspects of it when necessary, and learning over time. In this way a controller can be

	Language	Runtime reconfiguration	Network stack support	Embedded systems support	Component-based architecture
Iris	C++	•	•	•	•
GNU Radio	C++, Python	∘	×	∘	•
OSSIE	C++	×	×	•	•

• : Fully supported   ∘ : Partly supported   × : Not supported

Table 1. *Software radio architecture comparison.*

As components in the scheduled PHY domain typically operate on all input data samples, data passing is implemented using memory buffers at each component port. Components read data from the buffer at an input port, operate on it, and write it to the buffer at one or more output ports.

used to implement a cognitive engine which drives the operation of the entire radio.

As well as reconfiguring in response to internal triggers, the Iris architecture also provides support for *external reconfiguration triggers*. These external triggers may come, for example, from a user interface. Through a simple C-based application programming interface (API), new XML configuration documents can be passed to the system. These are parsed by an XML Parser to produce a radio representation, a data model of the radio. This representation is then examined by a *Reconfiguration Manager* and used to generate a reconfiguration set. This set of reconfiguration steps is enacted to reconfigure the running radio. Figure 1 illustrates both the XML Parser and the Reconfiguration Manager.

Table 1 compares the Iris architecture with two other software radio architectures: *GNU Radio* and *Open Source SCA Implementation::Embedded* (OSSIE) [6], an open source implementation of the SCA. All three architectures are implemented in C++, although GNU Radio uses Python for high-level construction and management of signal processing chains. Iris provides full support for runtime reconfiguration, while within GNU Radio runtime reconfiguration is possible by pausing an executing chain, reconfiguring, and resuming. As the SCA is designed for SDR, rather than cognitive radio networks, OSSIE does not support runtime reconfiguration. Iris is alone in providing support for a network stack within the architecture. Both Iris and OSSIE provide support for embedded systems, digital signal processors (DSPs) in the case of OSSIE and FPGAs in the case of Iris. GNU Radio is currently being ported to the Texas Instruments Open Multimedia Application Platform (OMAP) embedded processor. All three platforms use a component-based architecture.

Iris Engines

One of the motivations behind the design of the Iris architecture was to move from experimenting with simple point-to-point links toward larger networks of cognitive radio nodes. The support provided for not just the PHY layer but also higher layers of the network stack is something that differentiates Iris from other architectures such as GNU Radio and the SCA.

By examining the constituent parts of a node within a cognitive network, we can identify a number of domains, each with different requirements for reconfiguration, datapassing, and execution. Implementing an architecture that caters for just one of those domains will lead to reduced efficiency and greater development challenges. Within the Iris architecture, the concept of a modular domain *engine* is used. The Iris engine encapsulates one or more components of the overall data flow graph of the node and defines the datapassing, execution, and reconfiguration semantics for those components. A radio implemented in Iris may consist of one or more of these engines.

We define three domains within a cognitive network node and provide an engine for each. These domains are the scheduled PHY, the flexible PHY, and the network stack.

Scheduled PHY Engine

Components within the scheduled PHY domain typically lie close to the digital/analog interface, operate at high sample rates, and are not reconfigured on a regular basis. For these components, a fixed relationship between the rates of data consumed and produced can often be established. Data flow is generally unidirectional, and components typically operate on all input data samples. Examples of components which might operate within the scheduled PHY domain are fixed filter or interpolation components. Within this domain, an engine which supports highly efficient runtime operation and a low degree of reconfiguration is required.

The scheduled PHY (sPHY) engine of the Iris architecture implements a synchronous data flow (SDF) [7] model of computation (MoC). SDF MoCs are characterized by the static sample rates that exist on each port of components within the flow graph. This knowledge allows a thorough analysis of the graph at design time. The model can determine the required buffer sizes for each link, and the static and efficient execution schedules. It can also guarantee deadlock-free operation. The SDF MoC provides the most efficient execution for components implementing static signal processing functions that have fixed input and output data rates. Reconfiguration of an SDF graph at runtime is possible; however, it may require recalculation of the execution schedule and buffer sizes, which may incur significant overhead.

As components in the scheduled PHY domain typically operate on all input data samples, data passing is implemented using memory buffers at each component port. Components read data from the buffer at an input port, oper-

As with components in the scheduled PHY domain, data flow is generally unidirectional and operations are carried out on all input data elements. A reconfigurable OFDM demodulation component might operate within the flexible PHY domain for example.

ate on it, and write it to the buffer at one or more output ports.

Flexible PHY Engine

The flexible PHY domain encapsulates the highly reconfigurable components that provide the key flexibility of the PHY layer of the node. These components typically exhibit no fixed relationship between input and output data rates. As with components in the scheduled PHY domain, data flow is generally unidirectional and operations are carried out on all input data elements. A reconfigurable OFDM demodulation component might operate within the flexible PHY domain, for example.

The flexible PHY (fPHY) engine within the Iris architecture implements a model of computation based on data flow process networks (PNs) [8]. The PN MoC supports the greatest degree of data flow flexibility. In a 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 behavior. There are no restrictions on when processes output data and how much data they output.

Implementation of the fPHY engine using a straightforward PN MoC would involve assigning a single thread of execution to each component. One disadvantage of this approach, however, is the overhead incurred as a result of thread context switches in the case where many components execute in parallel on a general-purpose processor (GPP) platform with a low number of processor cores. In order to avoid this overhead and achieve tighter control over the number of executing threads in a given radio, a different approach was adopted. Components that execute in an fPHY engine operate according to a PN MoC. However, each fPHY engine contains a single thread of execution, and within that engine a PN MoC is emulated by that single thread. Parallel execution of components is achieved through use of multiple fPHY engines. Thus, given a chain of three components, one, two, or three parallel threads of execution can be employed through use of one, two, or three fPHY engines. In this way, the fPHY engine design leverages the flexibility of the PN MoC while providing the radio designer with tight control over the number of executing threads in the radio. While the flexible PHY engine supports significant component flexibility, the associated dynamic allocation of computing and memory resources can incur overhead at runtime.

As in the sPHY engine, data passing in the flexible PHY engine is implemented using data buffers for each input and output port.

Network Stack Engine

A limitation associated with existing software architectures for reconfigurable radio is the focus on the PHY layer and lack of support for higher layers in the network stack such as the media access control (MAC) and network (NET). One of the key design goals in redesigning the Iris architecture was to build in this support for higher layers, while providing the same support for runtime reconfiguration as in the PHY. This is achieved in Iris 2.0 by the network stack engine.

The network stack domain is characterized by a bidirectional flow of data through components. These components are often highly reconfigurable and capable of spontaneously generating data which flows through the node. Rather than operating on the individual data elements in a block, components in the network stack domain typically operate on packets of data, and add or remove headers and footers to these packets. A component implementing a Time-Division Multiple Access (TDMA) MAC would operate in the network stack domain for example.

Like the fPHY engine, the Iris network stack engine implements an MoC based on PNs. This MoC provides the flexibility needed for runtime reconfiguration of components in this domain. However, in contrast to the sPHY and fPHY engines, data passing is not implemented using input and output buffers. Instead, a block of data within the network stack engine remains in the same memory buffer as it passes through a component. Pointers to these memory buffers are passed between components. As components in this domain typically operate on packets of data, this approach provides greater efficiency.

The network stack engine supports the creation of cognitive radio networks through implementation of MAC and NET protocol layers. A number of additional Iris features may be used in such implementations. One of these features is the metadata associated with each block of signal data that passes through an Iris radio. This metadata can, for example, include timing information provided from an RF front-end that supports the VITA Radio Transport (VRT) protocol. Such timing information can be used in the implementation of a TDMA MAC layer implementation.

All engines within the Iris architecture implement a common interface for data passing, reconfiguration, events, and life cycle management. This common engine interface ensures that all engine types can be handled in the same manner by the overall Iris system. One advantage of this approach is the ability to implement controllers that are independent of engine type. A single controller can for example reconfigure a component in a network stack engine in response to an event triggered by a component in an fPHY engine. A further advantage of the common engine interface is the ability to extend the architecture through the addition of new Iris engines in the future if required.

Advanced Processing Platforms

Iris is designed to run on a wide variety of general-purpose platforms, ranging from high-performance multicore machines to embedded systems. All code is written in portable C++ and can be compiled for the Windows, Linux, and Mac OS X operating systems. Supported and tested CPU architectures include x86 (32- and 64-bit), PowerPC, and ARM. However, realistic cognitive radio systems often require dedicated hardware or specialized processors to achieve the high performance goals. This section describes how such systems are supported within Iris.

Dedicated hardware (e.g., FPGAs), or specialized processors (e.g., DSPs or the CellBE) provide high compute performance by exploiting the parallelism inherent in many signal processing algorithms. Therefore they are often used as accelerators in cognitive and software radio testbeds. Iris supports the integration of such hardware through the concept of a *software wrapper*. This is a software component for Iris, typically for the sPHY or fPHY engine, which manages all interfacing to the FPGA, DSP, or CellBE. All parameters and data inputs and outputs of the software wrapper component are routed to the accelerator hardware within this component, encapsulating all low-level tasks. The software wrapper component behaves like an ordinary software component on the outside but performs compute functions on dedicated hardware or processors. Figure 2 illustrates the software wrapper concept using the example of an FPGA-based receiver system.

Testbeds using this architecture have been demonstrated at a number of international conferences [9]. For example, a video streaming link using FPGAs at the transmitter (Tx) and receiver (Rx) was demonstrated at SIGCOMM 2009 (the Rx is outlined in Fig. 2). The data was modulated using differential quadrature phase shift keying (DQPSK) and encoded with convolutional codes of different strength. The coding strength was adapted to the current channel conditions through monitoring the current bit error rate at the receiver, demonstrating runtime reconfigurability of Iris on FPGAs. Figure 3 shows a photo of the demonstration setup. More detail on the use of Iris on FPGA platforms can be found in [10]. During DySPAN 2008, one demonstration used Iris to perform real-time computation of the full spectral correlation function over a 4 MHz bandwidth using the CellBE processor in a Playstation 3. While the computational complexity involved would make this impossible using the standard GPPs found in PCs today, it is feasible using the parallel processing power of the CellBE.

A second possibility for integrating specialized hardware in Iris is enabled by the modularity of the engine concept, as explained in the previous section. This would be appropriate, for example, in the case where multiple components execute on the hardware. It is relatively easy to add new types of specialized engines that wrap all functionality required to handle a particular type of accelerator hardware. For example, a chain of components to be executed on an FPGA could be handled by a dedicated FPGA engine, responsible for configuring the FPGA, carrying out reconfigurations (by setting register values or loading a different FPGA configuration), and all other interaction with the FPGA components. The FPGA components themselves would not need a wrapper component in this case. This is a very powerful feature which we intend to use extensively in the future.

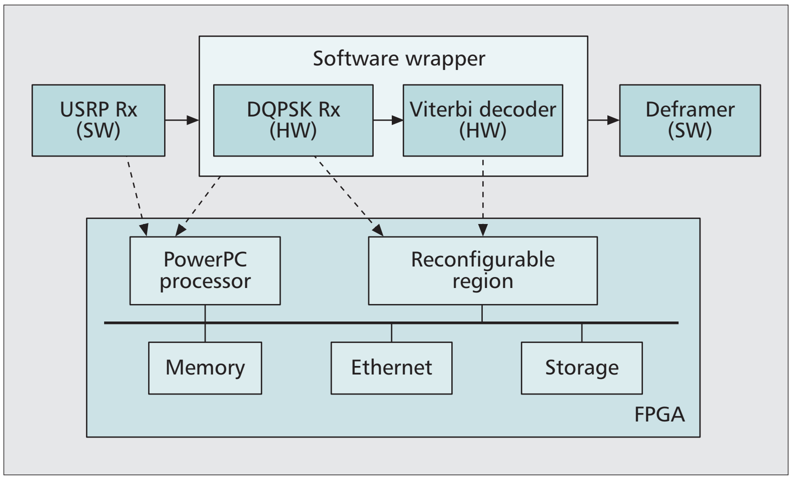

Figure 2. *The software wrapper concept for an FPGA-based narrowband receiver. The DQPSK Rx and Viterbi decoder components are executing in hardware (FPGA logic fabric). A software wrapper component manages the hardware/software interface. All hardware required to run Iris on a Linux operating system is contained in the FPGA.*

Example Application Using Iris

Iris was not designed with a single particular standard in mind. However, it contains many of the features required by a number of emerging cognitive radio standards such as IEEE 802.22 [11]. The IEEE 802.22 standard addresses the use of VHF and UHF TV band spectrum on a non-interfering basis to provide wireless regional area networks (WRANs) with the aim of providing broadband internet access to areas of low population density, typically in rural areas. A centralized base station (BS) uses vacant television channels to form a network with one or more subscriber stations. These vacant TV channels may be identified using spectrum sensing or geolocation together with a database. These approaches are in line with those proposed by the Federal Communication Commission (FCC) in their Second Report and Order [12] adopting rules to allow unlicensed radio transmitters to operate in the broadcast television spectrum.

To demonstrate the capabilities of the Iris architecture, a network of highly reconfigurable nodes was designed and implemented. While this network is not an IEEE 802.22 network, the design goals of both are similar. Nodes within the network opportunistically use spectrum *white spaces* to establish communication links, form a network, and exchange information. In order to accomplish this, a highly flexible OFDM waveform is employed. By adapting the carrier frequency and the bandwidth of this waveform, network nodes can take advantage of a wide range of detected spectrum opportunities.

A significant challenge associated with the use of such a reconfigurable waveform is network coordination. In order to establish communications and create a network, nodes must converge on a common waveform configuration. In our network this is achieved through use of intentionally embedded *cyclostationary signatures* [13, 14]. These signatures consist of correlation patterns inserted in the spectrum of a waveform. They can be detected using a low-complexity

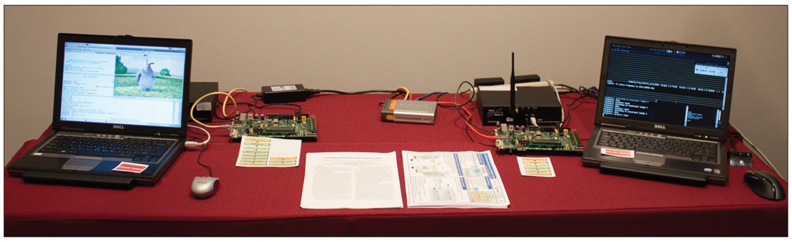

Figure 3. *Setup of the video streaming demo with adaptive coding using FPGAs at SIGCOMM 2009.*

detector with minimal information about the parameters of the waveform, and used for node identification, carrier frequency acquisition, and bandwidth estimation. Upon detection of such a signature in a received waveform, a node can determine all information required to establish a communications link. Under the proposed IEEE 802.22 standard, the use of predefined TV channels means that a superframe control header (SCH) can be broadcast in each channel used. In this case subscriber stations can simply scan channels to find an SCH, configure their waveform, and join the network. Using cyclostationary signatures, however, such a predefined channelization scheme is not required. Instead, nodes can form a network using any detected spectrum opportunity.

The structure of a single node within our network is illustrated in Fig. 4. In the following sections we examine the node design and illustrate how key features of the Iris architecture are used to support the flexibility required by the network. In particular, we look at:
- Reconfigurable component parameters
- Component events
- Controllers
- Iris engines

The node structure includes transmit and receive signal processing chains, linked by a TDMA MAC component. The transmit chain consists of a data scrambler, quadrature amplitude modulation (QAM) symbol mapper, OFDM modulator and USRP transmit (Tx) component. The Universal Software Radio Peripheral (USRP) Tx component links to the reconfigurable USRP RF front-end, feeding data to the hardware for transmission and controlling parameters such as carrier frequency, signal bandwidth, and transmit power. The first component in the receive chain is a USRP receive (Rx) component, which provides baseband in-phase and quadrature (IQ) signal samples received by the USRP. The switch component in the receive chain switches the data flow between two paths. The first path consists of two components: a simple spectrum analyzer, which performs energy detection, and a low-complexity cyclostationary signature detector. This path provides the analysis required for the node to detect spectrum opportunities and coordinate waveform parameters with other nodes in the network. The second path performs demodulation of received waveforms and comprises an OFDM demodulator, QAM symbol demapper, and data descrambler.

The node architecture makes heavy use of component parameters to support dynamic reconfiguration. The OFDM modulation component exposes parameters to adjust the cyclic prefix length and the number of subcarriers used in accordance with the bandwidth of the transmitting USRP hardware. In this way a constant subcarrier spacing is maintained as the bandwidth of the transmit waveform is adjusted. This bandwidth flexibility supports, for example, the channel-bonding mechanism required in IEEE 802.22 to achieve required data rates of up to 19 Mb/s over 30 km. As the OFDM modulation component also inserts cyclostationary signatures into the transmit waveform, additional parameters are exposed to permit adjustment of these signatures. Also in the transmit chain, the QAM symbol mapper component exposes a parameter to

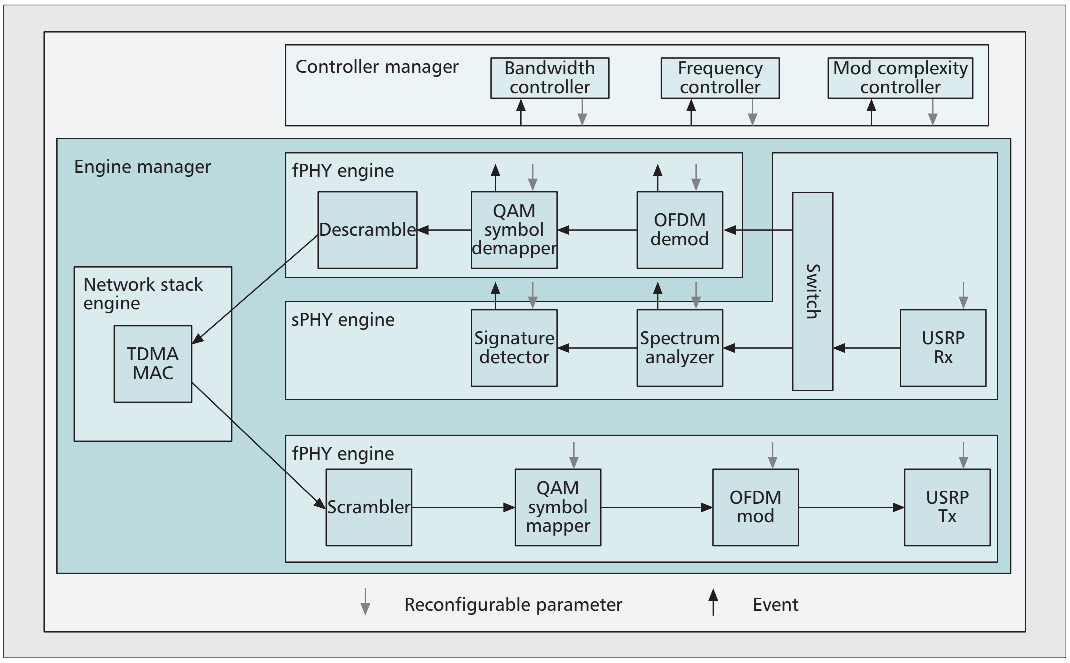

Figure 4. *Design of a reconfigurable network node using Iris.*

Conference	Demonstration system
DySPAN 2007	Cyclostationary signatures for detection, identification, and carrier frequency estimation of OFDM-based waveforms
DySPAN 2007	Coexistence of non-contiguous OFDM-based waveforms with co-channel single-carrier waveforms
DySPAN 2008	A reconfigurable FPGA-based system using a single-carrier waveform, employing sensing for carrier-frequency estimation
DySPAN 2008	A DSA network using a bandwidth-adaptive OFDM waveform with embedded cyclostationary signatures for blind bandwidth estimation and network coordination
DySPAN 2008	A real-time cyclostationary analyzer using the Cell Broadband Engine
FCCM 2009	A reconfigurable FPGA-based system using a single-carrier waveform with adaptive coding
SIGCOMM 2009	A reconfigurable FPGA-based system employing sensing for carrier-frequency estimation and adaptive coding
DySPAN 2010	A DSA network using reconfigurable pulse-shaped OFDM waveforms to control out-of-band (OOB) emissions

Table 2. *Demonstration systems built using the Iris architecture.*

control the modulation complexity used according to the channel conditions. In the receive chain complementary parameters are exposed by the OFDM demodulation and QAM symbol demapper components. Further parameters are exposed by the spectrum analysis and signature detection components to provide a trade-off between computational complexity and performance.

Reconfiguration of component parameters within the node is triggered internally by a number of *component events*. One such event is triggered by the spectrum analysis component to indicate that a spectrum opportunity has been identified. A second event can be triggered by the signature detection component upon detection of a cyclostationary signature. In order to choose the correct modulation complexity for the channel conditions, the OFDM demodulation component can trigger an event according to the estimated signal-to-noise ratio (SNR). Reconfiguration of the radio in response to these events is carried out by a number of controllers. Dedicated controllers are provided to select the appropriate carrier frequency, bandwidth and modulation complexity to be used by the node. In order to determine spectrum opportunities using the geolocation and database lookup approach specified in IEEE 802.22, an additional controller could be added to the structure. This controller would determine the node location using a GPS device and query a database using a backhaul link. Using data returned from the database, the controller would then reconfigure the node to use the spectrum resources specified.

The use of multiple *Iris engine types* can also be seen in Fig. 4. Our node architecture makes use of four engines in total. An sPHY engine is adopted for the receive chain branch performing signal analysis. Components in this chain have a fixed relationship between input and output data rates, and exhibit significant computational complexity. The low runtime overhead and limited flexibility of the sPHY engine is well suited to this scenario. Two fPHY engines are used in the node; one each for the transmit signal chain and demodulation receive chain. Components within these engines are highly reconfigurable and change the output:input data rate ratio according to their configuration. The flexibility of the fPHY engine supports the level of reconfiguration required by these components. Finally, the network stack engine is used to support the TDMA MAC component. As the MAC differs from the other components in the structure in that it operates on packets of data and requires its own thread of execution, this engine ensures that efficient data passing and execution mechanisms are used.

In this section we have presented one example network implemented using the Iris architecture. Table 2 lists a number of further systems using Iris, demonstrated at international conferences over the past 5 years.

Conclusions

The unique challenges inherent in developing standards for cognitive radio networking require highly flexible yet robust testbeds for real-world implementation, development, and verification. In this article we have presented Iris, an architecture designed specifically for building highly reconfigurable radio network testbeds. The architecture contains built-in features to support runtime reconfiguration across all layers of the network stack and a well defined interface to decision making processes that tailor the operation of the radio to the changing network environment.

Our future plans for Iris include making it available to the wider research community and building a base of active users who can benefit from the architecture and contribute to improving it. We plan to build on existing PHY layer implementations and focus on the design of MAC layer protocols, including carrier-sense multiple access (CSMA) and TDMA, as well as ad-hoc NET layer routing protocols such as Optimized Link State Routing (OLSR) and Dynamic Source Routing (DSR). We look forward to working with the research community to make Iris available for testbed implementation, *Our future plans for Iris include making it available to the wider research community and building a base of active users who can benefit from the architecture and contribute to improve it.*

and to enable the development of future standards for cognitive radio and dynamic spectrum access.

References

[1] P. Mackenzie, *Reconfigurable Software Radio Systems*, Ph.D. dissertation, Trinity College Dublin, 2004.
[2] Free Software Foundation, Inc., "GNU Radio — The GNU Software Radio," 2009; http://www.gnu.org/software/gnuradio/
[3] Joint Tactical Radio System (JTRS) Joint Program Executive Office (JPEO), "Software Communications Architecture Specification," JTRS Standards, v. 2.2.2, May 15, 2006; http://jtrs.spawar.navy.mil/sca/downloads.asp?ID=2.2.2
[4] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX — Understanding Broadband Wireless Networking*, Prentice Hall, 2007.
[5] J. Zyren and W. McCoy, "Overview of the 3GPP Long Term Evolution Physical Layer," Freescale Semiconductor, Inc., white paper, July 2007.
[6] C. R. A. Gonzalez *et al*., "Open-Source SCA-Based Core Framework and Rapid Development Tools Enable Software-Defined Radio Education and Research," *IEEE Commun. Mag*., vol. 47, no. 10, Oct. 2009, pp. 48–55.
[7] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *Proc. IEEE*, vol. 75, no. 9, Sept. 1987, pp. 1235–45.
[8] E. A. Lee and T. M. Parks, "Dataflow Process Networks," *Proc. IEEE*, vol. 83, no. 5, May 1995, pp. 773–801.
[9] P. D. Sutton *et al*., "Multi-Platform Demonstrations using the Iris Architecture for Cognitive Radio Network Testbeds," *CROWNCOM*, June 2010.
[10] J. Lotze *et al*., "Development Framework for Implementing FPGA-Based Cognitive Network Nodes," *IEEE GLOBECOM*, Nov. 2009.
[11] C. Cordeiro *et al*., "IEEE 802.22: The First Worldwide Wireless Standard Based on Cognitive Radios," *IEEE DySPAN*, Nov. 8–11, 2005, pp. 328–37.
[12] Federal Communication Commission, "Second Report and Order and Memorandum Opinion and Order in the Matter of Unlicensed Operation in the TV Broadcast Bands Additional Spectrum for Unlicensed Devices below 900 MHz and in the 3 GHz Band," Nov. 14, 2008.
[13] P. D. Sutton, K. E. Nolan, and L. E. Doyle, "Cyclostationary Signatures in Practical Cognitive Radio Applications," *IEEE JSAC*, vol. 26, no. 1, 2008, pp. 13–24.
[14] P. D. Sutton *et al*., "Bandwidth-Adaptive Waveforms for Dynamic Spectrum Access Networks," *IEEE DySPAN*, Oct. 2008.

Biographies

PAUL SUTTON (suttonpd@tcd.ie) is a research fellow with CTVR, the Telecommunications Research Centre based at the University of Dublin, Trinity College, Ireland. He has been actively involved in the research and development of software-defined radio systems since 2004. In this time his work has addressed key challenges in the areas of software radio architecture design, and rendezvous and coordination in dynamic spectrum access networks. He received his Ph.D. in electronic engineering in 2008 from the University of Dublin, Trinity College. He is currently serving as a director of the Wireless Innovation Forum.

JÖRG LOTZE (jlotze@tcd.ie) 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 Ph.D. student with CTVR, the Telecommunications Research Centre, at University of Dublin, Trinity College. His research interests include high-level design for cognitive radios, experimentation and testbeds, and communications algorithms.

HICHAM LAHLOU (hlahlou@tcd.ie) received his Master's degree in electrical engineering from the Institut National des Sciences Appliques de Lyon (INSA), France, in 2006, after which he joined CTVR, the Telecommunications Research Centre, at University of Dublin, Trinity College as a research fellow. His research interests include scalable computing, stream processing, and high-level system design.

SUHAIB A. FAHMY (sfahmy@ntu.edu.sg) received an M.Eng, degree in information systems engineering and Ph.D. degree in digital electronic systems from Imperial College London in 2003 and 2007, respectively. From 2007 to 2009 he was a postdoctoral research fellow at the University of Dublin, Trinity College, and a visiting research engineer at Xilinx Research Labs in Ireland. He joined Nanyang Technological University as an 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 in the domains of computer vision and communications.

KEITH NOLAN (keithnolan@mee.tcd.ie) received his Ph.D. degree in electronic engineering from the University of Dublin, Trinity College in 2005. He is a research fellow with the Telecommunications Research Centre (CTVR) at the University of Dublin, Trinity College. He organized the world's first public collaborative trials of cognitive radio and dynamic spectrum access in 2007, has served as Chair and Co-chair of demonstrations for IEEE DySPAN symposia, and on numerous TPCs for conferences concerning these areas also. He currently serves on the management committee for COST Actions IC0902 and IC0905, in addition to being a member of SCC41 where he is a technical co-author of the IEEE P1900.1 standard.

BARIŞ ÖZGÜL [M'04] (ozgulb@tcd.ie) is a postdoctoral research fellow at CTVR, University of Dublin, Trinity College. He received his Ph.D. degree in electrical and electronics engineering from Boĝaziçi University, Istanbul, Turkey in March 2008. From 1998 to 2003 he worked with Nortel Networks/Netaş, as an R&D engineer and software architect. He joined Turkcell İletişim Hizmetleri A.Ş. in 2003, where he worked as an expert in the Network Platform Development Department until June 2005.His research interests include communications and signal processing, dynamic spectrum access networks, software radio systems and prototyping, multicarrier systems, and MIMO communications.

THOMAS RONDEAU (twronde@idaccr.org) holds a Ph.D. from Virginia Tech in electrical engineering, graduating in 2007. He worked as a postdoctoral researcher with CTVR, University of Dublin, Trinity College from 2007 to 2008. He is now on the research staff of the Center for Communications Research in Princeton, New Jersey. His research spans areas of communications theory, signal processing, and software design, which are all part of his larger interests in software and cognitive radios.

JUANJO NOGUERA (juanjo.noguera@xilinx.com) received a B.Sc. degree in computer science from the Autonomous University of Barcelona, Spain, in 1997, and a Ph.D. 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 Xilinx Research Labs, Dublin, Ireland. His interests include high-level system design, reconfigurable architectures, and next-generation wireless communications.

LINDA DOYLE (linda.doyle@tcd.ie) is a member of faculty in the School of Engineering, University of Dublin, Trinity College. She is currently the director of CTVR, the Telecommunications Research Centre. CTVR is a national research center that is 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. She is responsible for the direction of CTVR as well as running a large research group that is part of the center. Her research group focuses on cognitive radio, reconfigurable networks, spectrum management, and telecommunications and digital art.