A Split MAC Approach for SDR Platforms

Paolo Di Francesco, Séamas McGettrick, Uchenna K. Anyanwu, James C. O'Sullivan, Allen B. MacKenzie, and Luiz A. DaSilva

Abstract—Implementation of carrier sensing-based medium access control (MAC) protocols on inexpensive reconfigurable radio platforms has proven challenging due to long and unpredictable delays associated with both signal processing on a general purpose processor (GPP) and the interface between the radio frequency (RF) front-end and the GPP. This paper describes the development and implementation of a split-functionality architecture for a contention-based carrier sensing MAC, in which some of the functions reside on an field-programmable gate array (FPGA) and others reside in the GPP. We provide an FPGA-based implementation of a carrier sensing block and develop two versions of a carrier sense multiple access (CSMA) MAC protocol based upon this block. We experimentally test the performance of the resulting protocols in a multihop environment in terms of end-to-end throughput and required frame retransmissions. We cross-validate these results with a network simulator with modules modified to reflect the mean and variance of delays measured in components of the real software-defined radio system.

Index Terms—SDR, FPGA, Split Architecture, CSMA

1 Introduction

The term Cognitive Radio (CR) applies to a wide variety of systems, ranging from transceivers that use simple techniques to gather information about the wireless environment and have basic decision-making capabilities, to systems that have multi-sensory features and are capable of sophisticated analysis, learning, and decision-making. Cognitive radios can be viewed as radio nodes that are aware of the context in which they are operating and can reconfigure themselves to best fit the current conditions in the medium.

A cognitive radio should be able to sense activity in the channel in which it is operating and reconfigure itself based on the results. To achieve this reconfiguration, cognitive radios are often built on software defined radios (SDRs). Flexible SDR platforms have been incorporated into a wide range of wireless communications technology, spanning from satellite communications to sensor networks. They allow the reconfiguration of waveforms, frequency, and modulation schemes in order to improve communication performance. Traditional SDR platforms are capable of performing most, if not all of their signal processing tasks on a general purpose processor (GPP). The hardware radio front-end employed usually performs only minimal tasks. For example, products in the Universal Software Radio Peripheral (USRP) family [1] are widely used by the SDR research community. They are inexpensive devices that provide the basic functions required for baseband signal processing. The combination of minimal radio hardware and appropriate software packages (e.g. GNU Radio, Iris, Sora) offers great opportunities for researchers to carry out cognitive radio experiments at a relatively low cost.

Having cognitive radio nodes reconfigurable via software makes it possible to analyze the radio environment and to adjust the system parameters to a particular operational situation quickly and without redesigning hardware. However, the flexibility achieved with the software introduces high delays due to the nature of the off-the-shelf computer processing the radio waveforms. To date, SDR and CR experiments have primarily focused on environments where a single link is established or, at most, two competing links coexist in an interference channel. Higher layer issues have been mostly neglected or analyzed using unrealistic assumptions due to the limitations introduced by the aforementioned delays.

In this paper we propose a new architecture for a carrier sense multiple access (CSMA) MAC, with MAC functionality split between a field-programmable gate array (FPGA) and the GPP. This approach provides an improved Rx/Tx turnaround response time for MAC implementations. We carry out time-critical functions in the FPGA, while non-time-critical functions remain implemented in software running on a GPP [2]. This paper also extends our previous random access MAC implementation on the USRP N210, a non-embedded SDR platform [3], to the E100, an embedded SDR platform. This work makes the following contributions:

(i) We propose an architecture for a CSMA MAC on a radio frequency (RF) front-end with limited computing capabilities, introducing a split-functionality implementation of a random access MAC where

- *P. Di Francesco, S. McGettrick, J. C. O'Sullivan, and L. A. DaSilva are with CTVR / The Telecommunications Centre, University of Dublin, Trinity College, Ireland.*
  E-mail: pdifranc@tcd.ie
- *U. K Anyanwu, A. B. MacKenzie, and L. A. DaSilva are with Wireless @ Virginia Tech, Blacksburg, Virginia, USA.*

This work has been partially funded by European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 258301 (CREW project) and by the Science Foundation Ireland (SFI) under grant No. 10/CE/I1853. This research was also supported by a Bradley Fellowship from Virginia Techs Bradley Department of Electrical and Computer Engineering and made possible by an endowment from the Harry Lynde Bradley Foundation.

some of the functions reside on an FPGA and others reside in the GPP. We develop a carrier sense block and a random backoff block in an FPGA, closer to the RF front-end. The implementation of carrier sensing and backoff mechanisms on the FPGA is agnostic to the choice of software platforms.

(ii) We incorporate our implemented carrier sensing and backoff FPGA blocks into two versions of a CSMA MAC protocol. The first version is a random access MAC with carrier sensing and explicit acknowledgements, similar to the IEEE 802.11 MAC. The second version employs implicit acknowledgements and is tailored for operation in a mutihop wireless environment.

(iii) We cross-validate all results between prototype-based experimentation and network simulations using OMNeT++.

This paper is structured as follows. In section 2, we describe the proposed architecture, whose functionality is split between an FPGA and a GPP. In section 3, we discuss the implementation of the software and hardware modules, also summarizing the details of the prototype we used. Section 4 presents the tests and results. In section 5 we discuss the related work on MAC protocol implementation in SDR platforms. Finally, section 6 summarizes our main conclusions and discusses areas for future work.

2 Random Access MAC Protocols for SDR Platforms

There are a number of considerations to take into account when designing a MAC protocol for cognitive radio nodes on SDR. A designer must first consider the limitations of the available hardware platforms, and the needs of the radio functions at the physical (PHY) and MAC layers. With this information the designer can make informed decisions about how to implement the radio protocol stack. In this section we do this by first discussing the limitations of the available platforms. We then look at the requirements of the radio functions to implement an effective MAC. Finally, we propose our solution to address these limitations and describe the proposed radio.

2.1 Platform Considerations

SDR platforms used by research groups for cognitive radio experimentations usually rely on inexpensive RF front-ends connected to a PC, where most of the radio chain actually runs in software in a GPP. Generally speaking, those RF front-ends are equipped with an FPGA chip on board that carries out minimal operations on the incoming and outgoing signals. This configuration allows the radio node to be inexpensive and extremely flexible. Fig. 1(a) shows a typical architecture for this configuration.

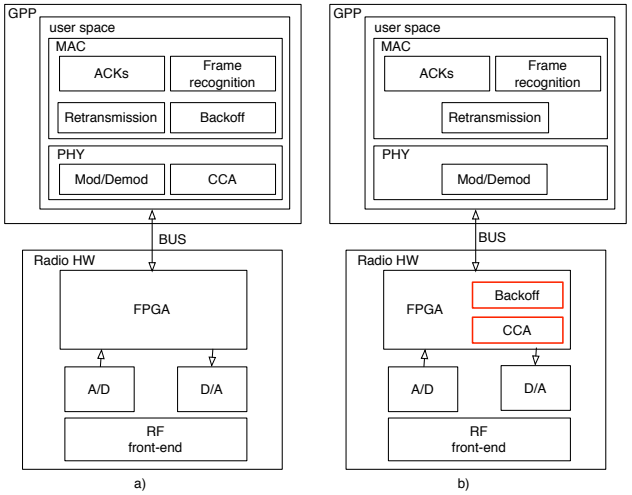

Fig. 1. a) Full Software architecture for a typical Radio Hardware (FPGA) + GPP (host user space) configuration. b) Split Functionality architecture proposed.

However, enabling networking experimentation on affordable SDR platforms presents a challenge at the MAC layer, where the issues of implementing protocols with reasonable performance have been well summarized in the literature [2] [4] [5], and were alluded to in our earlier work [3].

The flexibility achieved by SDRs does not lead to high-performance flexible MAC implementation. There may be significant and variable latency between the signal processing elements and the physical radio front-end. This long and variable latency results in a "blind-spot" in which assessments of the channel state may be of indeterminate age, and hence possibly stale [4]. Fig. 2(a) and (b) show a qualitative comparison of the frame processing delays between a conventional hardware network interface card (NIC) and an SDR system. The conventional NIC has a fixed pre-determined delay and thus a fixed pre-determined "blind-spot". The SDR on the other hand has a longer, more variable frame processing delay and thus a longer more variable "blind-spot". This leads to unpredictable performance of the MAC protocol. It particularly affects the turnaround time, which is made up of MAC layer processing time, non negligible data transfer time to and from the radio front-end and modulation/demodulation time. This unpredictable turnaround time leads to difficulty in predictably scheduling frames for transmission by the SDR, thus reducing the performance of many MAC protocols. For example, Puschmann et al. [6] have shown that fully software MAC implementations cannot meet the requirements of protocol parameters such as short inter frame spacing (SFIS) and distributed inter frame space (DIFS) needed in the IEEE 802.11 family. Even less demanding standards such as IEEE 802.15.4 for Wireless Personal Area Network (PAN), with a larger receive-transmit turnaround time, cannot be met when using a

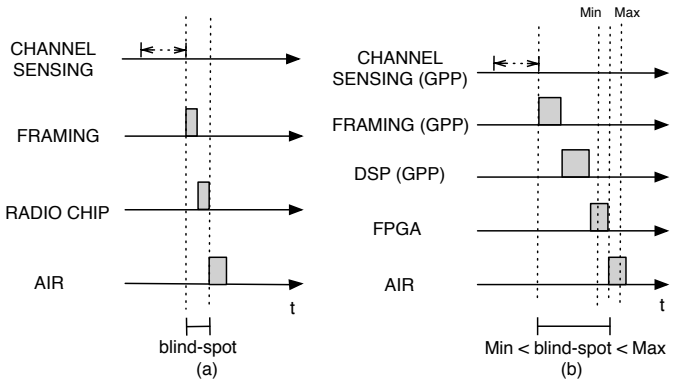

Fig. 2. Carrier Sensing to transmission processing delays in conventional NIC (a) and SDR (b).

Function	Low Latency	Low Complexity	Flexibility	Coupled
Clear Channel Assessment	✓	✓	✗	-
Backoff	✗	✓	✗	Clear Channel Assessment
Mod/ Demod	✓	✗	✓	-
Frame Recognition	✓	✓	✓	Mod/ Demod
Retransmission	✗	✓	✗	Mod/ Demod
ACKs Tx	✓	✓	✗	Mod/ Demod

TABLE 1
Radio function requirements for a CSMA MAC protocol.

MAC solution fully implemented in software [4].

2.2 Radio Considerations

In this paper we are particularly interested in contention-based MAC protocols. These protocols often use a carrier sense mechanism, which is a fundamental part of most wireless networking stacks (e.g. in wireless LAN and sensor networks), to detect other transmissions. This mechanism may use simple energy detection (received signal strength) or use matched filtering to detect incoming frames. The implementation of a CSMA MAC radio node can be roughly split into six main functions. These are: clear channel assessment (CCA), backoff, modulation/demodulation, frame recognition, retransmission, and ACKs transmission (see Fig. 1(a)).

Each of these functions has different requirements with regards to latency, computational complexity, flexibility, and order of execution.

The CCA function gives the node the ability to determine if the channel is idle by checking for the presence of other signals on the channel before transmitting. To avoid the "blind-spot" problem described earlier, the CCA function requires low latency. However, the function has low complexity, requires low flexibility, and does not require input from any other module.

The backoff function schedules transmissions on a random basis when multiple nodes try to access the medium simultaneously. It attempts to avoid two or more nodes accessing the channel at the same time when the channel is sensed idle. The backoff function is tightly coupled to the CCA, since it requires data from the CCA unit to function.

The modulation/demodulation functions are computationally complex modules, since they perform most of the digital signal processing (DSP) operations on the signal. They require high flexibility, to be used in many different configurations.

The frame recognition function gives the node the ability to detect if an incoming frame is relevant to the local node and eventually to decide the action to be taken by the MAC layer.

The retransmission function is the mechanism dedicated to retransmit lost frames.

Finally, ACKs transmission is the operation of generating acknowledgment frames in response to the reception of a DATA frame.

Frame recognition, retransmission and ACKs transmission cannot be implemented before the modulation/demodulation units, and as such should remain tightly coupled to those modules.

We summarize these requirements in Table 1.

2.3 Proposed Solution

In the previous two sections we have looked at the requirements of a CSMA MAC radio node and the limitations of platforms that use a minimal RF front-end. The large latency introduced by using an SDR behind a minimal RF front-end makes it difficult to achieve the low latency requirements of the carrier sense module in the radio node. To circumvent this problem we propose and implement a split-functionality hardware-software architecture for MAC protocols, judiciously placing signal processing functions over multiple computing platforms, such as an FPGA and a GPP. In Fig. 1(b) we depict how the proposed split functionality architecture would look in a typical FPGA + GPP configuration.

The goal is to identify MAC functions that must be moved as close as possible to the RF front-end, and functions that can remain in the host to maintain flexibility and ease of development.

Since the performance of the CCA is crucial to the MAC, we should place the CCA adjacent to the radio hardware. Moreover, the radio hardware must be able to store the incoming frame from the MAC layer ready to be transmitted in a buffer, and, based on the CCA result, quickly deactivate the carrier sensing and transmit the frame without waiting for further communications from the GPP. Without this storage system, a CCA in hardware would lead to little gain in terms of delay reduction.

The backoff is another important function in contention-based MAC protocols. Since it requires communication with the CCA, we implement it in on the FPGA too.

The modulation/demodulation modules are crucial enabling components for a MAC protocol. A tradeoff needs to be made between the flexibility of the module and the performance of the unit. The result of this tradeoff is largely determined by the hardware and software platforms used. As we are interested in highly flexible SDR we choose to leave the modulation/demodulation modules in the GPP, which is more conducive than hardware to the implementation of many different configurations.

Other functions, such as frame recognition, retransmission, and ACKs transmission handling remain on the GPP. These functions require fully demodulated frames, and since in our design we choose to leave the demodulation on the GPP, the natural place for these functions is to remain on the GPP.

2.4 Advantages and Drawbacks of Splitting the Functionalities between SW and HW

There are several advantages associated with implementing carrier sense and backoff in the FPGA. Firstly, the FPGA-based modules can be used with any software defined radio, as they do not require any specialized software modules. Secondly, the carrier sense blind-spot is significantly decreased, since the latency from sensing to sending frames is greatly reduced, as compared to a full software implementation of CSMA. This decreases the chance of collisions with other nodes' transmissions.

Splitting the MAC between the FPGA and software implementation also has some drawbacks, as the software MAC does not know whether a frame has been sent or whether it is just waiting in the TX buffer. In the software-only version of the design, the time necessary to send the frame through the FPGA logic is fixed due to the deterministic nature of FPGA signal processing. The MAC estimates this delay and includes it when determining the duration of the retransmission counter (i.e. the time the MAC should wait for an ACK before assuming the frame is lost and retransmitting the frame). However, in the software/hardware system the MAC unit can no longer assume that the frame is sent when it is transferred to the FPGA and so does not know when to start its transmission timeout timer. If the MAC starts the timer immediately, and the carrier sense unit stops the frame from being transmitted, the MAC unit might assume the frame has been lost and thus retransmit it. This would lead to unnecessary frames being sent on the channel and a lower throughput.

The integration of the software and the hardware implementation into a single communicating unit can be addressed by creating bidirectional control signals from the software to the hardware and vice versa.

3 Implementation

In this section, we present the detailed system designs for both the MAC implementation on a GPP and the carrier sense and backoff modules on the FPGA. Where possible we have kept this discussion general, as the proposed design can be targeted at any combination of software radio platform and RF front-end with user-programmable FPGA. The section concludes with details of the software radio platform and RF front-end used in these experiments.

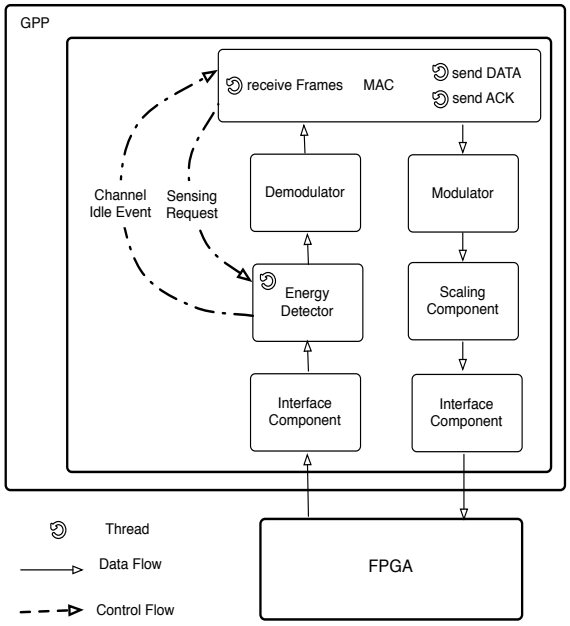

Fig. 3. Software-only carrier sensing MAC architecture in SDR.

3.1 Carrier Sense MAC Software Design

We implement two variations of the software MAC. The first is a software-only approach to carrier sensing, which is used as a reference design. The second software design implements the software/hardware split functionality carrier sensing MAC.

Fig. 3 shows a block diagram of a software-only carrier sensing MAC. When the MAC component has a frame to transmit, it sends a request - *Sensing Request* - to the energy detector component. Once the channel is free the energy detector reports back to the MAC with a *Channel Idle Event*. Now the MAC is ready to send the frame in the transmitting chain. A similar architectural approach is taken in [5].

Fig. 4 shows a block diagram of the software implemented for the software/hardware split functionality design. Since we offload some of the components (e.g. the Energy Detector) and logic to the FPGA, a number of changes need to be made to the software architecture. This decreases the workload carried out by the SDR platform running on the GPP and also simplifies the software, reducing potential software interrupt latencies.

In order to address the biggest limitation of the split-architecture (see section 2.4), we can exploit the existing *Frame Transmitted* feedback message that most radio front-ends provide. We make a small modification to the

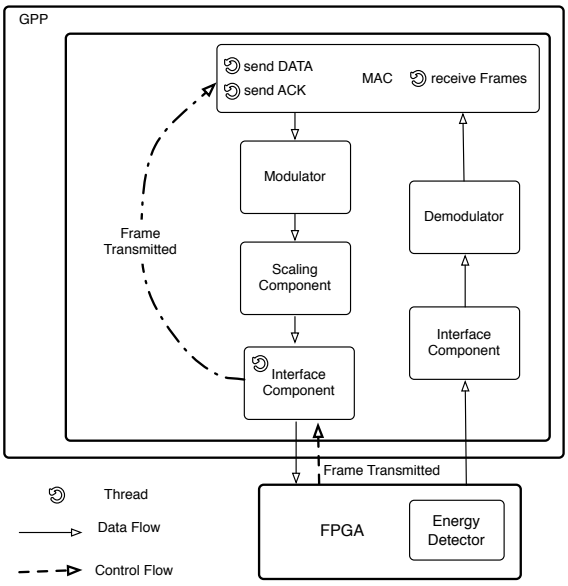

Fig. 4. Split carrier sensing MAC architecture with feedback in SDR.

MAC to utilize this information to start the retransmission timeout timer. Thus, we initiate the retransmission timeout timer once the frame has been sent across the channel, instead of when it leaves the MAC layer. In this way, the software and hardware work in tandem to provide access to the channel. It is also important to reiterate that, although the software and hardware communicate in this experiment, the FPGA hardware is agnostic to the software being run on the GPP.

3.2 Carrier Sense MAC Hardware Design

In section 2 we discussed the various aspects of the SDR MAC and we concluded that moving latency-sensitive blocks like carrier sense and backoff to the FPGA could increase the performance of an SDR system. In this section, we provide details of our implementation of these blocks.

3.2.1 Carrier Sense Module

The Carrier Sense block in the FPGA uses a simple energy-detection strategy. The block computes the average signal power for a set of samples and then compares this power with a programmable threshold value. If the received signal strength is greater than the threshold, then we send a signal indicating the carrier is present to the TX controller. We modify the state machine in the TX controller to ensure that the carrier present signal is low before sending frames to the RF front-end. In this way, our carrier sense module ensures the channel is free before transmitting.

Fig. 5 shows a block diagram of the carrier sense module as implemented on the FPGA. The data path can be split into four separate processes. These processes are the control registers, the magnitude calculation, the

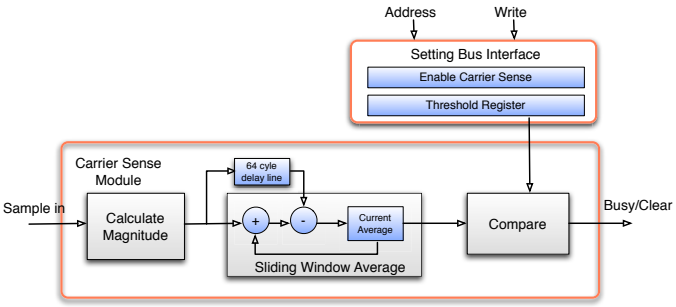

Fig. 5. Block Diagram of the proposed carrier sense module.

sliding window average, and the threshold compare. We use the control registers to enable/disable the carrier sense unit and to set the threshold value for the carrier sense unit. These registers are connected to a settings bus and can be written from the software. When the carrier sense module is disabled, it does not affect the rest of the FPGA hardware, and thus the carrier sense module can permanently remain on the FPGA without interfering with non-carrier sensing experiments. The FPGA-based energy detector performs the same set of operations implemented in the software-based energy detector (i.e. magnitude calculation, sliding window average, and threshold comparison).

The remaining three hardware components implement the sliding window carrier sensing module (see Fig. 5). We collect the data from the receive chain and the magnitude of the current sample is calculated. We add this magnitude to the current average and we simultaneously place it at the top of a delay queue. We subtract the data on the bottom of the delay queue from the average, as they have exited the moving average window. Finally we compare the average to a programmable threshold value and we set the busy/clear signal.

To allow the carrier sense module to pause/resume transmissions based on the state of the channel, a small change must be made to the TX path control state machine. The exact details of this modification vary depending on the radio front-end used but most radio front-ends use a radio transport protocol (RTP) [9] which has been designed to provide a standard and consistent way to transport data throughout complex heterogeneous systems. These protocols provide an intermediate frequency (IF) data frame that contains a header followed by the data payload. These frames pass status information such as RF and IF frequencies, bandwidth, internal delays, sampling rates, A/D overflow, and user-defined parameters between the GPP and FPGA.

To illustrate the process of integrating the carrier sense modules into an RTP we take the example of the VITA radio transport protocol [10].

Fig. 6 shows in black the original design for the VITA TX controller state machine. It consists of six states. The Idle state is the default state for when the TX chain is not in use.

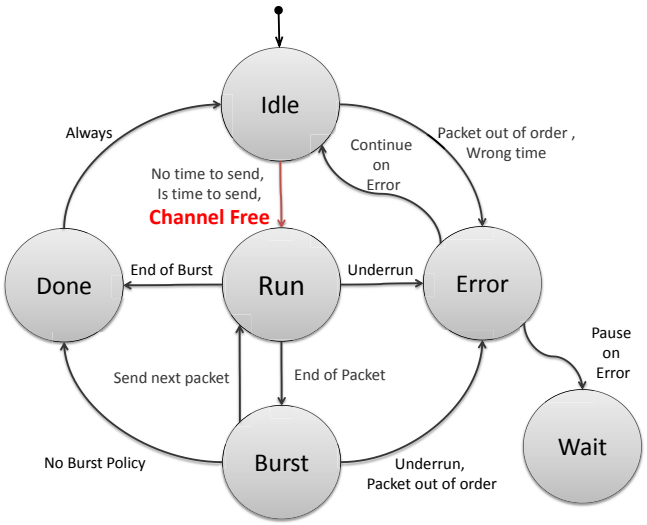

Fig. 6. Vita TX Controller State machine. Edits required to include carrier sense data are shown in red.

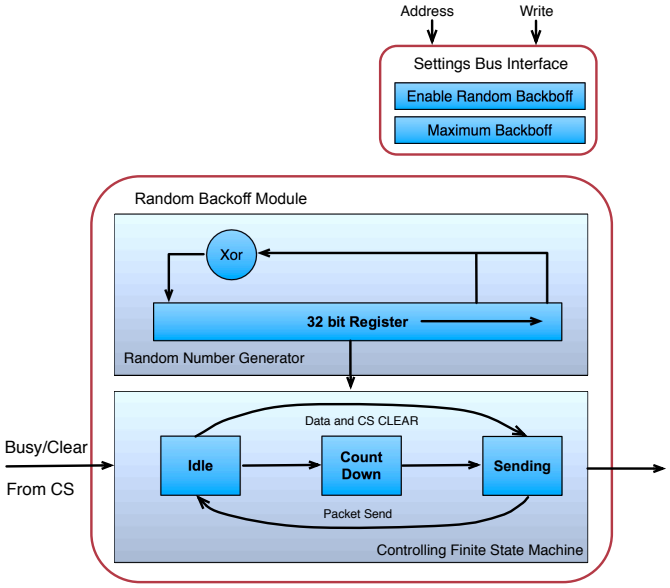

Fig. 7. Block diagram of the random number generator and finite state machine used in the random backoff hardware module.

The changes we make to this state machine are minimal. Now, instead of transitioning from the Idle state to the Run state when data is valid and when the timing conditions are right, we also check that the channel is free. If all of these conditions are true then the state machine can transition to the Run state.

3.2.2 Backoff Module Design

In addition to adding the basic carrier sensing functionality to the FPGA logic, we also design and implement a random backoff module on the FPGA, as shown in Fig. 7. The backoff module sits between the carrier sense module and the TX controller. We connect the carrier present signal from the carrier sense module and the TX queue status signal to the backoff module. We use the output of the backoff module in the TX controller to ensure that it is safe to send data on the channel. The random backoff module consists of a random number generator in the form of a Logic Feedback Shift Register (LFSR) and a controlling finite state machine shown in Fig. 7. The backoff module also contains two settings registers. The first register allows the backoff hardware to be enabled or disabled based on the users' preferences. When disabled, the backoff module relays the carrier present signal from the carrier sense module to the TX controller. The second settings register allows the user to set a maximum backoff value (in the form of a mask) to control the length of the contention window.

When the TX controller receives a frame from the higher layers, a "data in queue" signal is asserted. We pass this signal to the backoff finite state machine. If the carrier sense module measures the channel as free, the backoff module gives the TX controller permission to send the frame and transitions into the sending state. The backoff FSM remains in the sending state until it receives the "frame sent" signal from the TX controller.

If the carrier sense module finds the channel is busy when a frame is placed in the send queue, the backoff module transitions into the countdown state. As the FSM transitions to the countdown state, we set a countdown timer to the current output of the random number generator. The node must then wait for the countdown timer to become zero before we allow the transition into the sending state and send the queued frame. On every received sample in which the carrier sense module assesses the channel to be free, we decrement the countdown timer.

3.3 Carrier Sense MAC Prototype

In this work we use the USRP E100 as the radio front-end/processing platform and Iris [13], [14] as the software radio platform for our proof-of-concept implementation. In this section we give a quick overview of the USRP E100 and Iris, along with a discussion of why we choose these platforms.

It should be noted that the on-FPGA elements of the proposed design are agnostic to the choice of attached software platform. An FPGA hardware that preserves all the USRP Hardware Driver (UHD)[1] semantics will therefore work with other software radio platforms, for instance, in a GNU Radio [15] system.

3.3.1 USRP E100 Overview

The USRP family products [1] are the most popular example of a minimal RF front-end that relies on a PC for baseband signal processing. USRPs allow research groups to develop large testbeds with a reasonable

1. The VITA was chosen as the underlying transport protocol for UHD-enabled. The goal of UHD software is to provide a host driver and an API for Ettus Research products [1].

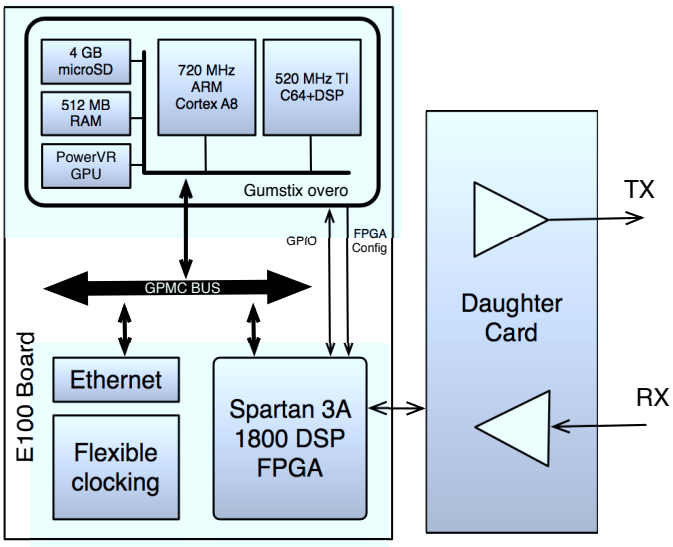

Fig. 8. A block diagram of the E100 USRP.

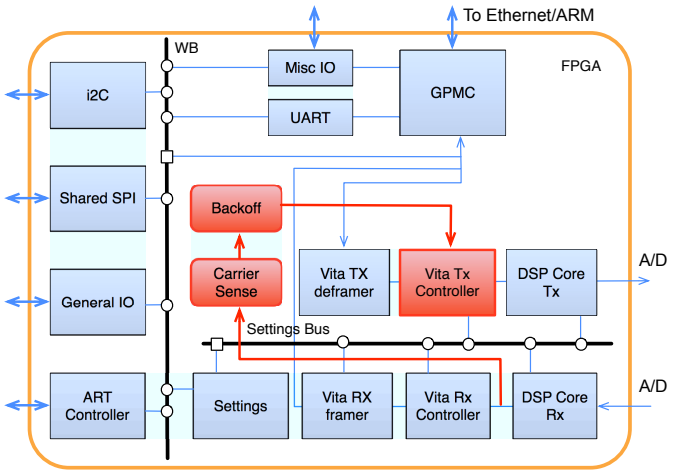

Fig. 9. An overview of the modules on the E100 USRP as shipped (grey) and the integration of the carrier sense and backoff blocks into the FPGA architecture (red).

budget (e.g. the ORBIT radio grid testbed at Rutgers University [11], CORNET at Virginia Tech [12]).

We select the USRP E100 as our hardware platform based on available FPGA space, programmability, ease of use, and cost. Moreover, the USRP E100, with its relatively low-power processor and tightly-coupled FPGA, represents a first step towards practical, real-world cognitive radio platforms. That is to say that the computational resources available on the USRP E100 are indicative of what a practical mobile cognitive radio may have available to it. However, this design can be easily extended to other platforms using the USRP + GPP configuration (e.g. USRP N210 + PC).

The USRP E100 is an embedded standalone software-defined radio platform. Fig. 8 gives an overview of the system. The E100 incorporates a Gumstix Overo board, FPGA, digital-analog converters (DACs), analog-digital converters (ADCs) and a daughterboard interface.

The Gumstix Overo board runs a full embedded distribution of Linux which enables development and deployment of software radio systems without the need for an external host computer.

An important feature of the USRP E100 is that the embedded processor is interfaced to the Xilinx FPGA with a direct memory interface. This feature is advantageous in radio designs where MAC functionality is split between the ARM GPP and the baseband FPGA hardware, as it allows fast access to baseband samples streamed from the FPGA and increased opportunities to control the data stream on the FPGA.

It is clear from Fig. 8 that the radio front-end is connected directly to the FPGA and that all data transmitted or received by the radio front-end must pass through the FPGA. The FPGA is therefore ideally placed to implement latency-sensitive processes, like sensing whether a channel is occupied and controlling transmissions to avoid collisions. To implement this system it is necessary to integrate customized FPGA blocks into the existing FPGA hardware design. As shipped, the FPGA on the USRP E100 contains a number of generic communication/debugging modules to aid in software radio design, and the RX and TX data chains for the radio front-end, see Fig. 9. The figure also shows the placement of additional modules (backoff, carrier sense) that are part of our split hardware/software solution.

Our hardware (shown in red) can be easily connected in parallel with the receive chain where it processes the RX data to determine channel occupancy. As discussed earlier, the output of the carrier sense module connects to the TX controller where it can be used to pause/resume data transmissions.

3.3.2 *Iris*

Iris [13] is an open source SDR platform and it interfaces with Ettus USRP RF front-end hardware, to allow for affordable experimentation [1].

The key elements of Iris can be listed in three groups: the components, the engines and the controllers. The components are blocks that perform DSP operations (e.g. modulator, demodulator, energy detector) and they are characterised by a set of user defined input/output ports used to connect them with other components. The engines encapsulate one or more components and execute them. They define how data are passed between components and how a specific part of the flow-graph is executed. There exist two types of engines in Iris: PHY and Stack. The PHY engine executes each of its child components and it is usually used for PHY layer operations. The Stack engine supports bidirectional data-flow and allows the user to spawn additional threads. It is intended to support upper-layer operations such as MAC, but the entire protocol stack can be defined and executed within the engine. Finally, the controllers manage and have a complete view of the radio. They can

reconfigure component parameters at run-time. They are usually triggered by events sent out by components, but they can also trigger commands which can be received by other components, to allow easy inter-component communication. In our design, the controllers are responsible to deliver commands and events from the MAC component to PHY components (e.g. *Sensing request* in Fig. 3) and vice versa (e.g. *Channel Idle Event* and *Frame Transmitted* in Fig. 3 and Fig. 4 respectively).

The Iris platform allows for quick and easy implementation of multi-layer radio chains (e.g. PHY and MAC), while other widely used platforms such as GNU Radio are more suited for only PHY layer experimentations.

4 Test and Results

In order to evaluate the impact of our split-functionality, FPGA/software implementation of carrier sense and backoff modules in a network of SDR nodes, we devise two sets of experiments. In the first experiment we compare the split MAC architecture performance to the software-only MAC architecture in a prototype experimental two-hop scenario. We also compare two different versions of carrier sensing MAC, one of them tailored to multihop communications. We then replicate the split software-hardware setup in a network simulator to cross-validate our findings and to further expand our analysis to larger networks. In the second experiment we evaluate the effects of the backoff module on the performance of the same two-hop network.

4.1 Carrier Sense Results

Our two-hop network testbed consists of three USRP E100 nodes with Iris running on the ARM processor and the XCVR2450 daughterboard [1] as a RF front-end transceiver. The testbed uses a host machine running a local NTP server (Fig. 10) for synchronization and monitoring purposes. In order to assess the performance of the system, we look at two metrics: the total number of retransmissions for a DATA frame and the end-to-end throughput. The retransmissions metric is the sum of the number of retransmissions required at the source node and the intermediate node to successfully transmit the data to the destination. The throughput is calculated as the total number of bytes sent from the source node to the destination divided by the time required to successfully send the data across the network.

Each experimental run involves 100 frames successfully transmitted from the source node to the destination node through one intermediate node. We limit our measurements to 100 frames, since we have observed experimentally that the system has reached steady-state performance at this point. In this scenario, the source and the destination are within interference range of each other. In this setting, the channel contention is due to intra-flow interference created by the DATA/ACK frame exchange between nodes. Table 2 and Table 3 present the parameters used for the radio experimental setup. It is worth noting how, even though the HW allows data rates up to 8 MSps [1], we use a lower data rate in order to avoid the loss of samples from the FPGA to the OMAP due to overflow events. Overflow events, in this case, are consequence of the poor floating point performance on the OMAP [16].

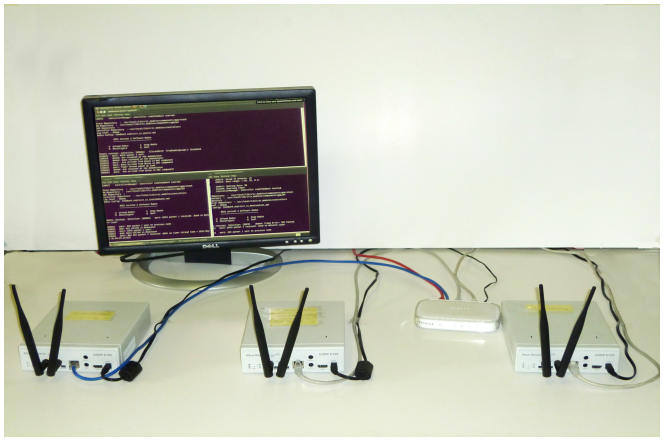

Fig. 10. Experiment setting in the CTVR wireless lab.

Parameter	Value
Frame size	600 B
Center Frequency	5.009 GHz
Sampling Rate	180 KSps
Bandwidth	180 KHz
Modulation Scheme	QPSK
Active data subcarriers	192 of 256
FFT size	256
UHD buffer size in RX	4196 samples
ARQ timeout	200 ms

TABLE 2
Table of RF parameters used

4.1.1 Software Architecture vs Split Architecture

In this first experiment, we compare the software-only architecture with the split architecture. We evaluate two different MAC protocols that we have already partially studied [3]. We refer to them as *Explicit ACK* with Carrier Sensing and *Implicit ACK* with Carrier Sensing. The Explicit ACK mechanism requires that all transmissions be acknowledged using a specific ACK frame. The Implicit ACK strategy encodes the acknowledgements into the frames being forwarded. We refer interested readers to [3] for more details about the protocols.

SDR operations and the wireless environment introduce a high degree of unpredictability on the system's

Component	Version
UHD / E100 FPGA	003.003.000
Iris	2

TABLE 3
HW & SW version used in the experiments.

behavior. For that reason, we repeat the experiments 50 times for each configuration and report the results in a box-and-whisker diagram. For the rest of the article, when referring to a box-and-whisker diagram, the whiskers below and above show the minimum and maximum value, respectively, the bottom and the top of the box represent the 1^{st} and 3^{rd} Quartile respectively, and the horizontal line inside the box represents the median. Outliers are excluded from calculation of min/max and are shown as small crosses. An outlier is defined as the value of an observation which is less than 1.5×1^{st} Quartile or greater than 1.5×3^{rd} Quartile.

In this first set of experiments we disable the FPGA-based backoff hardware and each DATA frame must be acknowledged in order to assess a successful transmission. In Fig. 11 we report the total number of retransmissions, and in Fig. 12 we report the end-to-end throughput of the system.

The results obtained suggest that a MAC protocol based on sensing uniquely in software fails to achieve reasonable timing requirements and results in poor performance (see section 2.1). With this approach, by the time the MAC issues the request for sensing, receives the command to transmit, modulates the frame, and finally transmits it over the air, the assessment of the channel state is outdated, resulting in a wide blind-spot. In Table 4 we report the measured delay between the moment at which the channel state switches from busy to idle and the time the node actually accesses the channel. We generate the blocking signal using a *Rohde & Schwarz* Vector Signal Generator, and we used a *Rohde & Schwarz* Real-Time Spectrum Analyzer to measure the Rx/Tx turnaround time. For convenience, in the rest of this article when presenting tables we report mean, median, and standard deviation, represented as $\bar{x}$, $\tilde{x}$, and s respectively. In the software-only architecture the frame is sent over the air with 54ms delay (on average), increasing the likelihood that the information about the channel is stale, hence increasing the chance of a collision. On the other hand, the split hardware/software architecture results in a 100-fold decrease in the turnaround delay. In this way the frame accesses the channel using more up-to-date information about the channel state. However, retransmissions can still take place due to either errors in correction of the carrier offset (rare) or timeout expiration in the MAC due to over-the-air collisions. As we can observe from Fig. 12, the split architecture delivers a throughput increase (on average) compared to the software-only architecture of 160% and 113% for the *Explicit ACK* case and the *Implicit ACK* case, respectively. In Fig. 13 we show the actual exchange of a sequence of frames over the air in the case of the *Explicit ACK*, captured using the *Rohde & Schwarz* FSVR Spectrum Analyzer.

As we stated in section 4.1, our experiments use a lower data rate than the nominal allowed by the hardware. This limitation is due to the limited processing capabilities of the ARM processor. We found that when

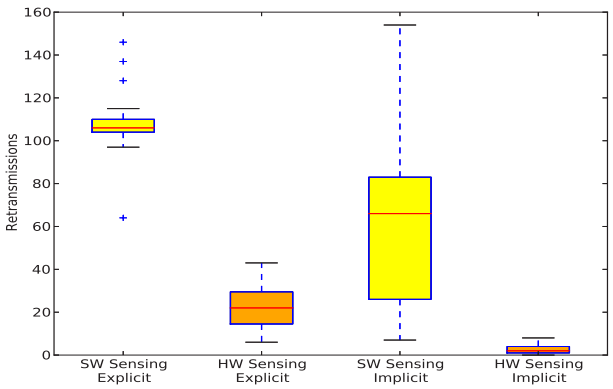

Fig. 11. Retransmissions.

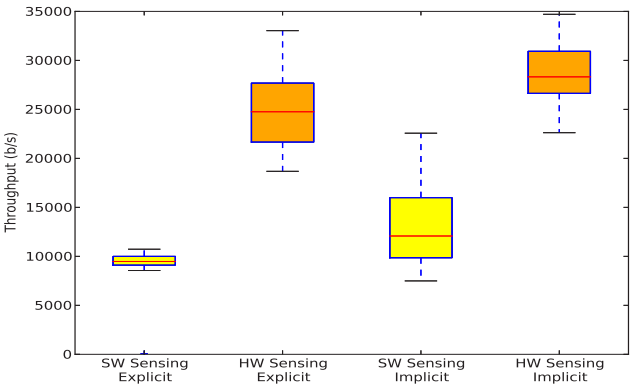

Fig. 12. End-to-end throughput.

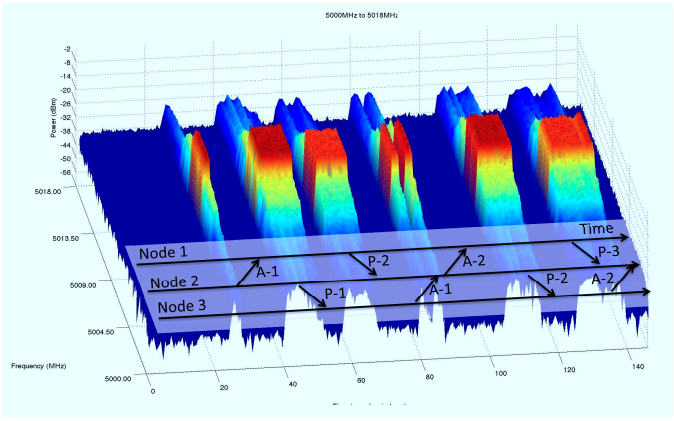

Fig. 13. Frame sequence exchange in the *CSMA with Explicit ACK MAC*. There are no collisions in this transmission. The P-3 data frame and A-2 acknowledgement appear on the channel in very close succession; however, the carrier sensing module ensures no collisions occur.

Rx/Tx turnaround time	$\bar{x}$ $[ms]$	$\tilde{x}$ $[ms]$	s $[ms]$
Sensing on GPP	54.54	45.34	11.01
Sensing on FPGA	0.46	0.45	0.01

TABLE 4
Carrier Sense Rx/Tx turnaround time at 180 KSps.

Logic Utilization	Total	Available	Utilization
Tot. No. Slice Registers	16,245 (15,007)	33,280	48% (45%)
No. of occupied Slices	13,464 (12,068)	16,640	80% (72%)
Tot. No. of 4 input LUTs	22,461 (20,901)	33,280	67% (62%)

TABLE 5
Carrier Sense module utilization in Xilinx FPGA (w/o CS in parenthesis)

implementing a software-only transceiver node using Iris on the ARM processor we could only achieve a data rate of 180 KSps. However, when we implement the split architecture, we can increase the data rate up to 250 KSps without risk of overflows, since we offload some of the components to the FPGA. However, to have a fair comparison between the two architectures, we limit the data rate for the split architecture to 180 KSps.

Table 5 is a summary of the FPGA resource utilization with both the basic UHD modules supplied with the USRP and the proposed carrier sense module. The table also shows, in parentheses, the FPGA resource utilization of only the UHD hardware as supplied on the E100. The carrier sense module implementation and the supplied modules occupy 80% of the total slices, which consist of registers and look-up tables, which means there is still considerable space available in the FPGA fabric to extend our carrier sense module or provide other functions.

4.1.2 Simulations

Simulations play an important role during protocol design and refinement phases. They also provide an easy way to perform larger scale experiments that otherwise would not be feasible to accomplish on prototype testbeds. However, many simulation results are never validated against real-world or experimental scenarios. We sought to use our experimental results to cross validate our simulations to enforce our understanding or our results and to enable for the testing of our protocol in situations unobtainable in our lab. We use the OMNeT++ network simulator. We set up a two-hop network using OMNeT++'s standard libraries and we modify existing modules in order to replicate our SDR implementation of CS, including the impact of modules running in the GPP and those running in the FPGA. It is well known that a conventional SDR node based on the USRP suffers of *unpredictable turnaround* time [3] when performing networking experiments. Unfortunately, as we have experienced, these effects are more prominent in embedded systems.

In order to model the unpredictable turnaround time of an SDR node, we first characterize the time required to process the signal, and the time needed to pass data to each component. Then we modify the existing Network, MAC, and Physical layer modules in OMNeT++ to introduce the modeled delay.

In Table 6 we report the time spent in each component in the receiver and transmitter chain, differentiating when possible whether the entry refers to the time necessary to process a DATA frame or an ACK.

We calculate the processing time by taking a timestamp before and after the `process()` function in each component[2].

The results shown in Table 6 are peculiar to our prototype; however, the methodology can be reused to obtain a similar set of results using a different software platform (e.g. GNU Radio).

To further characterize the radio components we collect data on other MAC functions (e.g. framing and deframing in Table 7) that are common to both the fully-software and split architecture MAC. Moreover we calculate the time taken by the different engines to pass data among themselves (see Table 8). To complete this task we use the same methodology used for Table 6, setting a timestamp right before the frame is released by the engine and setting another timestamp when the subsequent engine detects the frame. Finally, we measure the time needed to *request an event* and *issue a command* (see Table 9).

Several observations can be drawn from the results. Firstly, we observe that the receiver chain is the critical part in an SDR node. While the transmitter chain works only when the MAC sends down a frame, the receiver chain has to process samples coming from the FPGA constantly. When the UHDRx buffer is full with samples coming from the FPGA, it sends these samples to the next radio component, where they are processed in Iris (in our case the demodulator in Fig. 3 and Fig. 4). Secondly, the different frames' dimensions do not affect the processing time on the demodulator. The reason for that resides on equalization and carrier phase recovery functions, which are independent of the data size.

Lastly, the data being passed from the *PHY Engine* to the *Stack Engine* during the receiving phase (see Table 8) introduces a high variation in the system. The reasons for that stem from a combination of the current Iris framework and the thread scheduling in the operating system handling the USRP E100. The thread running the *PHY Engine* in the receiver chain gains greater priority with respect to the one running the *Stack Engine* because it constantly receives samples to be analyzed from the hardware. This leads to an unfair scheduling of the resources between the threads involved in Iris, where, even if the *Stack component* has data sitting in the buffer ready to be processed, it waits for the *PHY Engine* to complete its operations.

We repeat the simulations 50 times for each MAC configuration. As it is the case with the experiments, the retransmissions highly influence the throughput. The simulations show a decreased number of retransmissions and an increased throughput of around 8% compared to the laboratory experiments, as we show in Fig. 14 and

2. See the Example component source code provided in [14].

Component	Frame	$\bar{x}[ms]$	$\tilde{x}[ms]$	$s[ms]$
Modulator	DATA	2.58	2.57	0.06
	ACK	0.45	0.45	0.03
Signal Scaler	DATA	0.53	0.53	0.03
	ACK	0.14	0.14	0.02
UhdTx	DATA	0.34	0.33	0.03
	ACK	0.16	0.15	0.03
UhdRx	raw data	0.20	0.19	0.02
Demodulator	DATA	7.45	7.23	2.58
	ACK	6.53	7.21	2.43
Energy Detector*	raw data	1.82	1.45	1.01

* Only considered in the software-only architecture.

TABLE 6
Processing time measurements in Iris - PHY layer components used in software-only and split software-hardware architecture.

MAC Function	Frame	$\bar{x}[ms]$	$\tilde{x}[ms]$	$s[ms]$
Framing	DATA	4.45	0.42	31.78
	ACK	3.23	0.42	25.33
Deframing	DATA	0.09	0.09	0.01
	ACK	0.03	0.03	0.01

TABLE 7
MAC function profiling in Iris.

Data Passing Time	Frame	$\bar{x}[ms]$	$\tilde{x}[ms]$	$s[ms]$
Stack→PHY	DATA	3.26	0.52	5.94
	ACK	3.75	0.23	8.43
PHY→Stack	DATA	27.99	23.98	26.72
	ACK	23.16	10.18	31.15

TABLE 8
Time required to pass data from different engines (PHY and Stack) in both the software-only and the split software/hardware architecture.

Event/Command Passing	$\bar{x}[ms]$	$\tilde{x}[ms]$	$s[ms]$
Sensing Event→Sensing Command*	9.28	1.53	18.01
Ch. Idle Event→Ch. Idle Command*	3.42	0.64	11.71
Frame TX Event→Frame TX Command**	0.72	0.69	0.05

* Only in fully-software architecture.
** Only in split-architecture.

TABLE 9
Events to Commands processing time.

Fig. 15. It is likely that this difference between simulated and experimental results is due to the assumption we make in the simulations of no fading effects in the wireless channel.

Once we have cross validated the hardware and simulation results we expand our test scenario using simulation to include more hops. In Fig. 16 we evaluate the fully-software architecture and the split-architecture for both Explicit ACK and Implicit ACK from a point-to-point link to a maximum of 5 hops. We observe that the split-architecture always delivers considerably higher throughput compared to the fully-software architecture. As we increase the number of hops, the throughput decreases in both architectures. In particular we observe a 40% throughput decrease from a point-to-point to a 2-hop network for the case of Explicit ACK and 32% decrease for the Implicit ACK case in the split-architecture. On the other hand, for the same case in the fully-software architecture, there is a throughput decease of around 74% and 69% for the Explicit ACK and Implicit ACK case respectively.

As we add more hops, the throughput decreases due to increased contention in the channel.

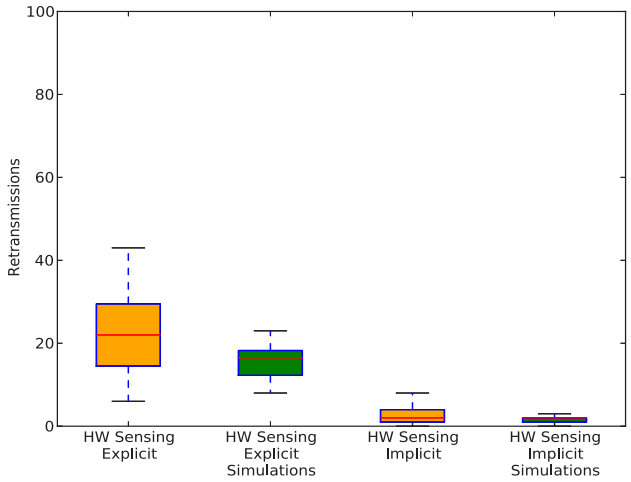

Fig. 14. Retransmissions.

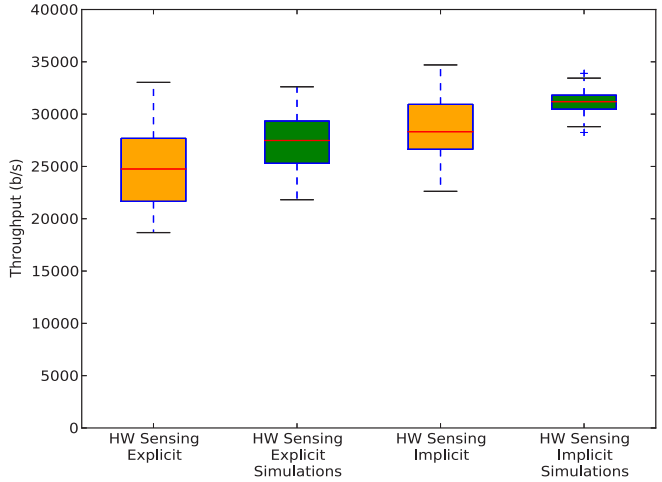

Fig. 15. End-to-end throughput.

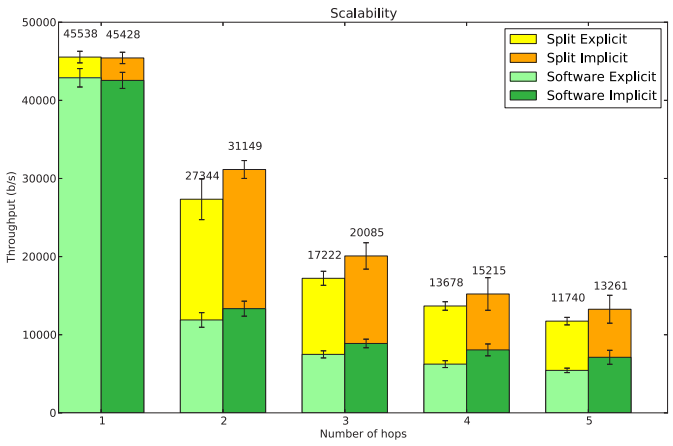

Fig. 16. Throughput when increasing number of hops.

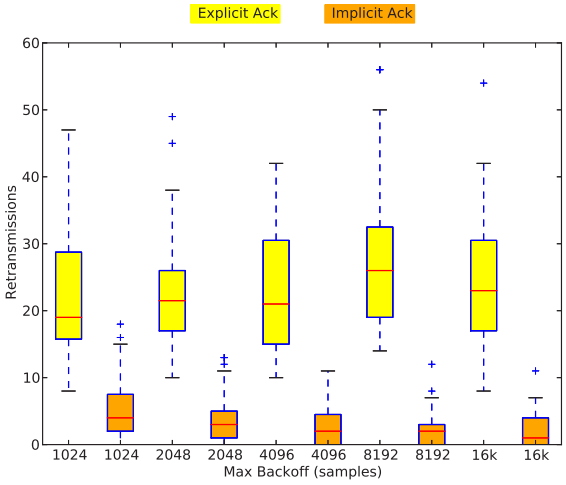

Fig. 17. Retransmissions of various maximum backoff times.

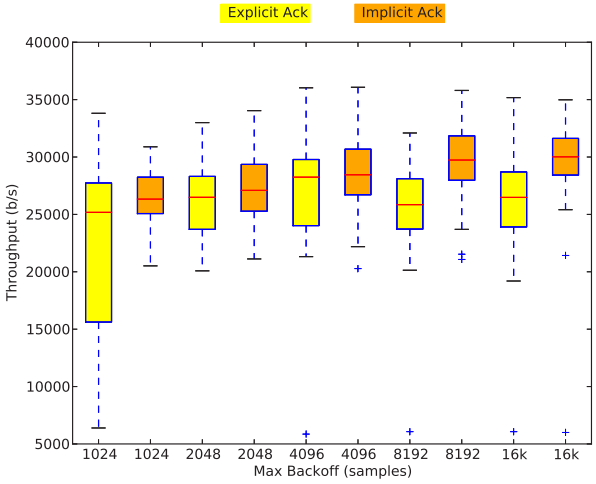

Fig. 18. Throughput of various maximum backoff times.

4.2 Backoff Results

The second set of experiments evaluates the effect of the random backoff hardware module on the retransmission rate and throughput of the original two-hop network. In these experiments we increase the maximum backoff time for each of the nodes from 1024 samples to 16384 samples in steps of powers of two. The actual backoff time is a random number between 0 and the maximum backoff time. If the channel is free when a frame arrives for transmission, it is transmitted straight away. However, if the channel is busy, the frame waiting to be sent must wait for the channel to be free for at least the random backoff time selected before the node is allowed to transmit.

We repeat the same experiment 50 times for each configuration of maximum backoff time.

Fig. 17 shows the number of retransmissions and Fig. 18 shows the throughput in a two-hop network for different maximum backoff times.

When we increase the maximum backoff time between 1024 samples and 16K samples, we observe a decreasing trend in the number of retransmissions for the Implicit ACK strategy. This translates into an increase in throughput of about 12% when the maximum backoff time is increased from 1024 samples to 8192 samples (see Fig. 18). However, when we further increase the maximum backoff to 16K samples, the number of retransmissions stabilizes compared to the previous case (8192 samples) and we observe a slight degradation in the throughput because of the longer congestion window considered. For the Explicit ACK case we observe more retransmissions compared to the Implicit ACK. However, we observe an increase in throughput of about 22% when the maximum backoff time is increased from 1024 samples to 4096 samples. When we further increase the maximum backoff, we observe a decrease in throughput of about 4%.

To conclude, a random backoff component adds some variation to the systems, which is evident from the more elongated box and whisker plots.

5 Related Work

In the following we survey research endeavors to enable MAC development on software-centric and hardware-centric SDR platforms. These endeavors have exposed the communication and processing limitations in SDR platforms.

Hunter et al. take a hardware-centric approach to decreasing MAC protocol delays, targeting the WARP platform [7] [18]. Their implementation places PHY layer functionality on the fabric of the platform's FPGA, and exposes some parameters to MAC implementations running on the embedded PowerPC GPPs. In this manner turnaround time and throughput are improved while permitting some protocol flexibility. This reconfigurability can be achieved through partial reconfiguration, when supported by the FPGA. Nonetheless, the on-FPGA implementation of the PHY limits overall system flexibility when compared to the split hardware-software approach we propose, in which only turnaround-time-

sensitive portions of the overall PHY and MAC mechanisms are implemented on-FPGA. The characteristics of the PHY waveform on the WARP cannot be altered beyond the range of pre-provided parameters built into the FPGA implementations, whereas our system benefits from a more flexible software-based implementation.

Kuo et al. [20] have built an embedded, low power, and portable SDR platform called μSDR. They identify some of the same issues we have raised in this paper. They have implemented the full receiver chain on an FPGA, which moves away from the flexibility needed by many SDR/cognitive radios.

In [19] the authors acknowledged the high delays observed when using inexpensive SDR equipment. The solution proposed by the authors employed a hardware-assisted CCA mechanism, showing the positive impact on the Rx/Tx turnaround time compared to a fully-software solution. However, they did not provide a hardware frame storing system. With our solution, the Rx/Tx turnaround time can be further improved [6] and has a positive impact on the frame error rate too.

The SDR philosophy introduced the concept of developing loosely coupled functional blocks that can be reused and recombined to reduce costs of deployment and to improve the overall system flexibility. In [21] the authors introduce the concept of *Flexible Link-Layer*, where they partition the functions of the link-layer protocol into multiple reusable block-functions. We embrace a similar philosophy, developing MAC functions as blocks, with the difference that now the blocks are not implemented exclusively in software, but some of them are implemented in hardware (e.g. Carrier Sense and backoff).

To the best of our knowledge, the research reported in this paper is the only work that has addressed MAC performance on an embedded SDR platform, through real-world simulations and prototype implementation.

6 Conclusions & Future work

In this paper, we have described the implementation of random access, carrier sensing MAC protocols on inexpensive reconfigurable radio platforms. We have designed this implementation using a split-functionality approach, implementing the most delay sensitive functions, including carrier sensing and associated frame-transmission logic, on the FPGA, close to the RF front-end. We have combined this FPGA-based implementation with appropriate modifications to the GPP-based MAC functions of a software radio built with the Iris software platform, though the FPGA-modifications themselves are software platform agnostic.

We performed mutihop experiments to assess the performance of our carrier sensing MAC in a realistic environment, measuring the throughput and number of retransmissions in a two-hop linear network. We showed that a split software-hardware carrier sensing significantly improved the performance of the MAC protocols, as compared to a software-only carrier sensing implementation. As a matter of fact, the carrier sensing implemented in the GPP is unproductive because of the blind-spot caused by the large receive-transmit turnaround time. We also cross-validated our experimental results using a network simulator (OMNeT++), which we modified to reflect the details of our MAC implementation as well as the component delay times seen in our experimental system.

Embedded SDR platforms posses limited computational capabilities. For this reason in [6] we have studied the impact of the proposed architecture on a more classic SDR configuration (e.g. USRP N210 + PC), together with strategies (e.g. frame pre-fetching) already employed in commercially available wireless cards and how it is possible to reduce the difference with the time constraints imposed by modern communication protocols.

In future work, we plan to further enhance our carrier sensing MAC protocol implementation via more sophisticated frame-dependent logic and more complex network topologies. We also plan additional tests to monitor the energy consumption of the device with the split architecture using the NITOS EMF Framework [22].

The FPGA image used for this work is available at [23].

References

[1] "Ettus Research LLC." [Online]. Available: http://www.ettus.com/.

[2] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC protocol Implementations on Software Defined Radios," in *USENIX symposium on Networked systems design and implementation (NSDI)*, 2009; pp. 91–105, DOI: 10.1.1.150.4119.

[3] J. C. O' Sullivan, P. Di Francesco, U. K. Anyanwu, L. DaSilva, and A. MacKenzie, "Mutihop MAC Implementations for Affordable SDR Hardware," in *IEEE New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, 2011; pp. 632–636, DOI: 10.1109/DYSPAN.2011.5936259.

[4] T. Schmid, O. Sekkat, and M. B. Srivastava, "An Experimental Stady of Network Performance Impact of Increased Latency in Software Defined Radios," in *ACM 2^{nd} Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WinTECH)*, 2007; pp. 59–66, DOI: 10.1145/1287767.1287779.

[5] A. Puschmann, M. A. Kalil, and A. Mitschele-Thiel, "A Flexible CSMA based MAC protocol for Software Defined Radios," *Journal of RF-Engineering and Telecommunications (Frequenz)*, 2012; **6(9-10)**, pp. 261–268, DOI: 10.1515/freq-2012-0048.

[6] A. Puschmann, P. Di Francesco, M. A. Kalil, L. A. DaSilva, and A. Mitschele-Thiel, "Enhancing the Performance of Random Access MAC Protocols for Low-cost SDRs," in *ACM 8^{th} Workshop on Wireless Network Testbeds Experimental Evaluation and Characterization (WinTECH)*, 2013; pp. 9–16, DOI: 10.1145/2505469.2505481.

[7] P. Murphy, A. Sabharwal, and B. Aazhang, "Design of WARP: A wireless open-access research platform," in *European Signal Processing Conference*, 2006; pp. 1804–1824.

[8] J. M. Fifield " A Software Defined OFDM Modulator," Master Thesis, 2006 [Online]. Available: http://systems.cs.colorado.edu/ fifield/ms-thesis.pdf.

[9] J. Malsbury, and M. Ettus, "Simplifying FPGA Design with A Novel Network-on-Chip Architecture," in *2^{nd} Workshop on Software Radio Implementation Forum (SRIF)*, 2013; pp. 45–52, DOI: 10.1145/2491246.2491251.

[10] R. Normoyle, and P. Mesibo, "The VITA radio transport as a framework for Software Definable Radio architectures," in *SDR '08 Technical Conference and Product Exposition*, 2008.

[11] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *IEEE Wireless Communications and Networking Conference*, 2005; pp. 1664–1669, DOI: 10.1109/WCNC.2005.1424763.
[12] T. R. Newman, S. M. S. Hasan, D. DePoy, T. Bose, and J. H. Reed, "Designing and deploying a building-wide cognitive radio network testbed," *IEEE Communications Magazine*, 2010; **48(9)**, pp. 106–112, DOI: 10.1109/MCOM.2010.5560594.
[13] P. D. Sutton, J. Lötze, 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 Communications Magazine*, 2010; **48(9)**, pp. 114–122, DOI: 10.1109/MCOM.2010.5560595.
[14] Software Radio Systems Ltd., [Online]. Available: http://www.softwareradiosystems.com/.
[15] Free Software Foundation, Inc., "GNU Radio The GNU Software Radio." [Online]. Available: http://www.gnu.org/software/gnuradio/.
[16] P. Balister, 2011. *High Performance Interface between the OMAP3 and an FPGA*, [Online]. Available: http://elinux.org/images/7/7b/Omap3-fpga.pdf.
[17] A. Puschmann, M. A. R. Kalil, and A. Mitschele-Thiel, "Implementation and Evaluation of a Practical SDR Testbed," in *Conference on Cognitive Radio and Advanced Spectrum Management (CogART)*, 2011; No. 15, DOI: 10.1145/2093256.2093271.
[18] C. Hunter, J. Camp, P. Murphy, A. Sabharwal, C. Dick, "A Flexible Framework for Wireless Medium Access Protocols," in *IEEE Asilomar Conference on Signals and Systems*, 2006; pp. 2046–2050.
[19] A. Puschmann, and M. A. Kalil, "The Impact of a Dedicated Sensing Engine on a SDR Implementation of the CSMA Protocol," in *10th International Symposium on Wireless Communication Systems (ISWCS)*, 2013; pp. 1–5.
[20] Y. S. Kuo, P. Pannuto, T. Schmid, and P. Dutta, "Reconfiguring the Software Radio to Improve power, Price, and Portability," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2012; pp. 267–280 DOI: 10.1145/2426656.2426683.
[21] A. Puschmann, M. A. Kalil, and A. Mitschele-Thiel, "A Component-based Approach for Constructing Flexible Link-Layer Protocols," in *8th Conference on Cognitive Radio Oriented Wireless Networks (CROWNCOM)*, 2013; pp. 244–249, DOI: 10.1109/CROWNCom.2013.6636825.
[22] S. Keranidis, G. Kazdaridis, V. Passas, T. Korakis, I. Koutsopoulos, and L. Tassiulas "Online Energy Consumption Monitoring of Wireless Testbed Infrastructure through the NITOS EMF Framework," in *ACM 8th Workshop on Wireless Network Testbeds Experimental Evaluation and Characterization (WinTECH)*, 2013; pp. 73–80, DOI: 10.1145/2505469.2505478.
[23] Available: https://github.com/mcgettrs/E100_carrier_sense.

Paolo Di Francesco Paolo Di Francesco received the B.S. and M.S. degree in telecommunications engineering from University of Bologna in 2008 and 2011 respectively. He is currently pursuing a PhD. degree in CTVR at Trinity College Dublin. His research interests include Software Defined Radios, embedded systems, wireless networks architectures, resource sharing techniques, and big data analysis.

Séamas McGettrick Séamas McGettrick is a Research Fellow in CTVR, The Telecommunications Research centre , Trinity College Dublin. He is interested in reconfigurable hardware/software co-design and how it can be used to solve complex real world problems. He is currently researching and prototyping protocols for next generation wireless networks and Passive Optical Networks.

Uchenna K. Anyanwu Uchenna K. Anyanwu completed his Masters in Computer Engineering at Virginia Tech in 2012. After honing his technical acumen from his Alma mater, Virginia Tech, and gaining research experience in CTVR at Trinity College Dublin, he obtained a position as a software engineer at a major defense contractor in the United States. His current role is to implement satellite terminal protocols in software and hardware. His technical experience and interest is in developing high-speed, low-latency control layer protocols in FPGA and x86 processors.

James C. O' Sullivan James C. O' Sullivan was awarded the B.A.I. degree in Electronic and Computer Engineering from Trinity College Dublin in 2007, and the Ph.D. degree in Electronic Engineering from Trinity College Dublin in 2012. His research was in the area of MAC protocol design and analysis for Cognitive Radio, and he is particularly interested in the practical application of CR MAC Protocols, a move towards a more modular and adaptable view of MAC protocols in general, and the use of software-defined radio to facilitate both. He is currently a software developer, working in large-scale data processing and latency-sensitive systems.

Allen B. MacKenzie Allen B. MacKenzie received his bachelor's degree in Electrical Engineering and Mathematics from Vanderbilt University in 1999. In 2003 he earned his Ph.D. in electrical engineering at Cornell University and joined the faculty of the Bradley Department of Electrical and Computer Engineering at Virginia Tech, where he is now an associate professor. During the 2012-2013 academic year, he was an E. T. S. Walton Visiting Professor at Trinity College Dublin. Prof. MacKenzie's research focuses on wireless communications systems and networks. His current research interests include cognitive radio and cognitive network algorithms, architectures, and protocols and the analysis of such systems and networks using game theory. His past and current research sponsors include the National Science Foundation, Science Foundation Ireland, the Defense Advanced Research Projects Agency, and the National Institute of Justice. Prof. MacKenzie is a senior member of the IEEE and a member of the ASEE and the ACM. Prof. MacKenzie is an area editor of the *IEEE Transactions on Communications* and an associate editor of the *IEEE Transactions on Mobile Computing*. He is the author of more than 45 refereed conference and journal papers and the co-author of the book *Game Theory for Wireless Engineers*.

Luiz A. DaSilva Luiz A. DaSilva holds the Stokes Professorship in Telecommunications in the Department of Electronic and Electrical Engineering at Trinity College Dublin. He is also a Professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech, USA. His research focuses on distributed and adaptive resource management in wireless networks, and in particular cognitive radio networks, dynamic spectrum access, and the application of game theory to wireless networks. Prof. DaSilva is currently a principal investigator on research projects funded by the National Science Foundation in the United States, the Science Foundation Ireland, and the European Commission under Framework Programme 7. He is a co-principal investigator of CTVR, the Telecommunications Research Centre in Ireland. He has co-authored two books on wireless communications and in 2006 was named a College of Engineering Faculty Fellow at Virginia Tech.