Towards Software-Centric Listen-Before-Talk on Software-Defined Radios

Sebastian Bräuer, Anatolij Zubow, Falko Dressler
Technische Universität Berlin, Chair of Telecommunication Networks
Email: {braeuer,anatolij.zubow,dressler}@tu-berlin.de

Abstract*—Listen-Before-Talk (LBT) is an essential function of many MAC protocols and a key mechanism of sharing spectrum in an uncoordinated manner. However, this simple but effective concept remains a major challenge for Software-Defined Radios (SDRs). If performing the protocol stack on a host PC, due to their structure, SDRs have inherent latencies built in. These latencies increase their reaction time, the so-called *turnaround* time, significantly, compared to their conventional radio counterparts. Unfortunately, this means that they cannot comply with the LBT channel access procedures used in modern protocols like IEEE 802.11 or LTE Licensed-Assisted Access (LTE-LAA). Given the flexibility and rapid-prototyping capabilities of SDRs, these protocols could clearly benefit from such SDR-based implementations. In this paper, we fill this gap and present a design approach for SDRs supporting LBT. We particularly focus on the rather complicated timing issues. As a proof-of-concept, we showcase our LBT-enabled implementation of the *srsLTE* software stack for LTE.

***Index Terms*—Listen-Before-Talk, Software-Defined Radio, GNU Radio, USRP**

I. Introduction

Software-Defined Radios (SDRs) have proven to be a very useful tool in the hands of researchers and engineers alike. In such a radio system, significant parts of the transceiver, which traditionally would be implemented as dedicated circuitry, are instead handled digitally by a General Purpose Processor (GPP) [1]. This concept has the clear advantage that such a radio is much more flexible in its use. While a traditional radio is tied to the physical characteristics of the radio protocol it was built for, software radios can easily switch their protocol (also referred to as *waveform*) during their runtime through a simple software update. This makes them an ideal platform for specialized radio purposes, where possibly multiple waveforms are required in a single unit as well as for rapid prototyping of (new) protocol stacks [2]–[5]. A popular programming framework for such SDRs is GNU Radio [6].

Now, some protocols require action on the scale of microseconds for specific parts of the protocol stack. A typical example is Listen-Before-Talk (LBT), which is an essential function of many MAC protocols and a key mechanism of sharing spectrum in an uncoordinated manner. Examples include modern protocols like IEEE 802.11 [7] or LTE Licensed-Assisted Access (LTE-LAA) [8]. The LBT mechanism creates a unique problem for software-defined radios. In order to increase flexibility of the SDR, most (in the best case, all) of the protocol stack is implemented in software in a GPP, which typically runs on a host PC. This introduces additional latency in the signal and protocol processing and particularly increases the reaction time, the so-called *turnaround* time, significantly.

This additional latency has two main sources: First, processing delay is introduced by the GPP. In a traditional radio all signal processing task such as filtering, modulation, etc. are handled in parallel by specialized circuits. In an SDR, these computationally heavy tasks have to be handled by the GPP on a limited number of processor cores with additional overhead for parallelization. Second and most importantly, latency is added due to the necessary transport of data between the SDR components. Due to the fact that a powerful processor is required for signal processing, the GPP often cannot be co-located with the other radio components on the same circuit board. Typically, the GPP is connected via some high-bandwidth link such as Gigabit or 10-Gigabit Ethernet, PCIe, or USB 3.0. This requires additional buffering, packetization, and processing. The additional latency adds up to the order of milliseconds, depending on the specific hardware used.

This creates a unique problem for software-defined radios: How can we facilitate time-critical functions using software radios with all their advantages? One answer is to be bring the software processing closer to the radio hardware. The easier approach is to just bring the GPP physically closer to the radio hardware on the same SoC. An example would be the *USRP E300* device family by *Ettus* [9]. Unfortunately, this SoC is usually limited in its processing power. The second, powerful but substantially more complicated approach is to make use of Field Programmable Gate Arrays (FPGAs). Such an FPGA is programmed to handle most of the protocol stack, while the GPP only interacts on a high-level packet interface. An example is the Labview platform [10] by National Instruments. While this approach works, it introduces additional complexity and decreases flexibility.

In our opinion this is an unnecessary price to pay. In this paper, we present a novel LBT controller concept to bridge this gap. We follow the *split-functionality* approach by Nychis et al. [11]. The goal is to reduce the complexity of the hardware design and preserve the flexibility. Functions of the stack with loose timing restrictions should remain in the software domain, and only time-critical functions that need to be close to the radio frontend should be implemented in hardware. We particularly focus on one specific time-critical core function which is common among many protocols: *Listen-Before-Talk*. We present how LBT can be designed with a software-centric, split-functionality approach, while reducing the latency. Our hope is to provide a building block on which more software stacks for more waveforms can then be built to widen the range of applications for SDRs. To achieve this, we introduce a new modular component on the FPGA, which only facilitates LBT and is otherwise transparent and thus not specific for a particular protocol. Furthermore, we propose a generalized interface to control our component so that the component parameters are adjustable to the protocol requirements. As a proof-of-concept, we demonstrate a working prototype running a modified LBT-enabled LTE stack on an *USRP X310* SDR using the *RF-Network-on-a-Chip (RFNoC)* [12] SDR hardware development framework. Using the LTE-LAA protocol stack as an example, we demonstrate the capabilities of our novel LBT controller. Our system can be used by the research community as a general toolchain for rapid prototyping of (novel) protocol stacks using easy-to-use frameworks such as GNU Radio – the implementation will be made available as Open Source.

Our main contributions can be summarized as follows.

- We introduce a novel LBT controller to support the development of random access protocols on SDRs,
- we implemented our LBT controller in the RFNoC framework using Verilog and studied the resulting latencies for acting upon protocol primitives, and
- we showcase the impact of this work using the LTE-LAA protocol stack as an example.

II. Related Work

Since the problem of time-critical MAC layer functions, especially LBT, is very common in the SDR community, there are already some ways to tackle this issue. In general, they can be grouped into three categories based on where the MAC layer functions are implemented and how they are distributed within the SDR.

The first category are the *SoC-based* solutions. Here GPP, FPGA, and radio frontend are directly interconnected on the same board. The best known examples are the *Nutaq ZeptoSDR* [13] and the *USRP E3x0 Series* [9]. These solutions have the distinct advantage that the latency between the components is minimal. This also enables the use of traditional software development tools and testing methods. However, the CPUs on these embedded devices are very limited due to their embedded nature. That means that they quickly saturated with their signal processing tasks, especially for high bandwidth applications.

The second category are *FPGA-based* solutions. These avoid the latency problem by handling all low-layer processing close to the radio frontend on an FPGA. So instead of having a streaming interface for received samples, the GPP only interacts with the FPGA on a per-frame basis. Notable examples of this approach are OpenWifi [14] and the *Real-time LTE/WiFi Coexistence Testbed* [15] or the *Wireless open-Access Research Platform (WARP)* [16] platform. Such FPGA-based solutions are able to meet strict timing requirements. However, their problem is that they are inherently inflexible: Changing parts of their protocol stack requires remodelling the FPGA image, which requires specialized knowledge in a hardware description language like *Verilog* or *VHDL*.

The third category are so-called *split-functionality* solutions. The idea is to combine the advantages of the other two categories. Nychis et al. [11] identified a list of core functions, which are common among many protocols and time-critical: Precise scheduling, carrier sensing, backoff, fast packet recognition, and handling of so-called dependent packets (e.g., ACKs). Only those time-critical functions are supposed to be handled by an FPGA. All other functions with less strict time restrictions are executed by the GPP. This requires an extra control channel in the SDR interface to configure and control the added components. Bloessl et al. [17] realized such an approach to enable carrier sensing and backoff of IEEE 802.11p in the GNU Radio framework. Even though this approach is able to meet the deadlines for LBT, it is very specific to both an SDR platform (USRP N210) and a protocol stack (IEEE 802.11p). The same can be said for the CSMA and TDMA proof-of-concept protocols in [11].

Support for re-usability of hardware components among different SDRs is difficult due to ways of handling data internally. This issue has been addressed by the *RF-Network-on-a-Chip (RFNoC)* framework by Ettus [12]. RFNoC is a uniform internal streaming interface for all the FPGAs of the USRP SDR series that allows for easier development of functional blocks, which they call *Computation Engines*. The hope is that this encourages modularity and reuse of FPGA hardware designs. We think this framework contributes to more general split-functionality SDR platforms and use it for implementing our novel LBT controller.

Finally, we want to mention one particular solution, which does not fit into the listed categories: Microsoft's *SoRa platform* [18] is a specialized board connected via PCIe that allows for software-only waveform implementations on a regular multicore PC. Low latency and real-time accuracy are achieved through a custom driver that dedicates CPU cores for handling the radio board. However, the board depends on a custom driver only working on Windows XP and the system is also not easily extensible nor portable to other protocol stacks.

III. Problem Statement

The goal of this paper is to design a generalizable and modular Listen-Before-Talk function for Software-Defined Radios. In order to understand why this is a challenging task, we first have to re-consider the basic architecture of SDRs. We then explain in detail where a naïve approach would fail and outline the properties a solution to this problem should have.

A. SDR Architecture

Typically, an SDR is functionally split into three distinct parts: First, there is the so-called *radio frontend*, which comprises all analog parts, such as the antennas, analog filters, as well as the digital-to-analog/analog-to-digital converters. The radio frontend is directly connected to the second part, the *digital frontend*. It is responsible for rate conversion and digital filtering, converting the received samples from the desired

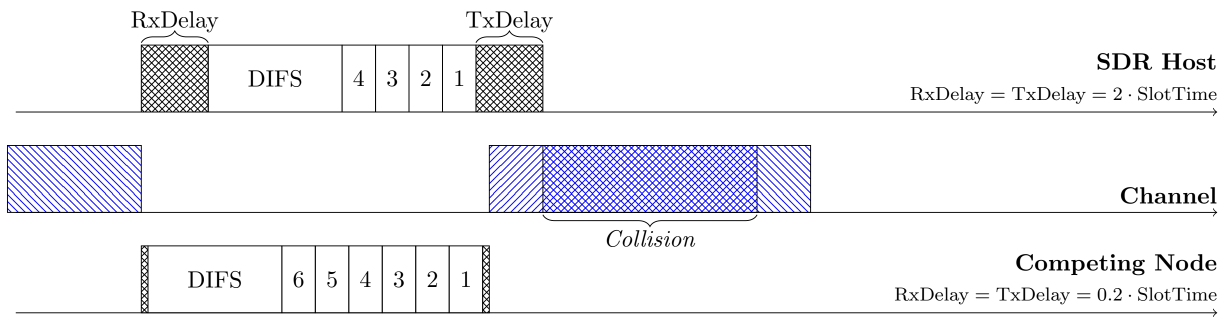

Figure 1. Exemplary timeline of naïve SDR implementation of IEEE 802.11 channel access colliding with a standard compliant competing node.

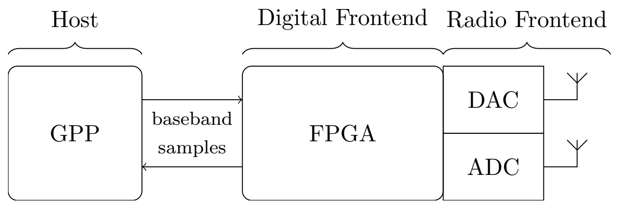

Figure 2. Generalized architecture of a typical SDR.

band into the baseband and vice-versa. Additionally, the digital frontend exerts control over parts of the radio frontend, e.g., to set oscillator frequencies. Usually, this function is co-located with the radio frontend and implemented via a DSP or an FPGA. The third major part is a processing unit that handles the flow of samples. It is responsible for baseband modulation/demodulation and most importantly all higher-layer processing. Often, this processing is performed on a GPP, which is physically separated from the frontend often even running on a separated host PC. An overview of this generalized architecture is shown in Figure 2.

The host interacts with the digital frontend via a streaming interface to and from which baseband samples are transported. This abstraction requires large buffers on both sides to account for the jitter introduced by the non-deterministic scheduling in the GPP and the network link. Additionally, processing of the samples in the actual SDR application are delayed by additional overhead from the operating system. This added delay is significant for time-critical applications. Jiao et al. [19] measured the round trip latency for the *USRP X310* using a PCIe link to the host PC to be 79 µs. Not every SDR follows exactly this structure, however, it is very common and frequently used in our research community.

B. Naïve LBT Implementation

Let us now consider how one would implement LBT conventionally on an SDR using the CSMA procedure of IEEE 802.11 as an example. If the host wants to transmit, it first has to start the receiver sample stream. The radio frontend would start sending samples to the GPP, based on which it can compute the energy over a given time period to determine the state of the channel. Because of the inherent delays in the SDR architecture, these samples would always represent the channel state several microseconds in the past.

Let us suppose the channel is busy at first but then the ongoing transmission ends. The host will observe this later than competing traditional nodes and, thus, start its backoff with a significant offset. This leads to the effect that a competing node with a higher slot number than the SDR host might finish its backoff earlier and, thus, acquire the channel before SDR can. Therefore, the SDR has a lower chance for channel access than the other nodes.

Depending on the exact timing, also a more destructive effect will take place. The delay between radio frontend and host is also present in the transmission stream. Thus, when the SDR decides to transmit there is a time window for a competing node to finish its backoff before any samples can hit the air. Hence, if a competing node chose a backoff slot that is very close to the one of the SDR, a collision might occur even though both stations chose different slots. This case is illustrated in Figure 1. The probability of such a collision rises with the length of the transmission delay and reduces spectrum efficiency.

C. Requirements

On of the key features of software radios is their flexibility in usage. Since they provide a programmable interface to the physical layer, it is possible to add filters, change modulation or even to switch or combine protocol stacks. Any LBT-capable SDR design should not interfere with these capabilities. Hence, any solution must be transparent to the host in the data path and preserve the streaming interface. Furthermore, the LBT function must be configurable in such a way that a broad range of protocols can use it with different parameters. It is also desirable that the configuration of the LBT function can be controlled by the host in software, for two reasons: First, it enables a reconfiguration of the device during the runtime, otherwise any hardware modification on the FPGA would commonly require power-cycling the device. Second, it reduces the complexity level for faster prototyping and less errors in the development.

IV. LBT Controller Design

We propose a new design for Listen-Before-Talk on Software-Defined Radio. For this design, we introduce a new

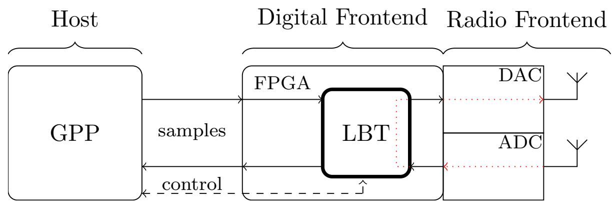

Figure 3. Proposed LBT controller design. Critical path for latency shown by the red-dotted line.

component to the digital frontend, the *LBT Controller*. Its purpose is to act as a baseband sample gate, which blocks the transmission whenever a transmission is not permissible under a given set of protocol parameters for the LBT procedure. As soon as the channel is available, the LBT Controller allows the samples to pass through to the radio frontend. Since the LBT Controller must be able to observe the channel to determine whether or not channel access can be acquired, it must be connected to the receiving baseband sample stream. The generalized composition is outlined in Figure 3. Via a secondary control channel the host can check after each transmission if the channel could be acquired or if the transmission was blocked. This design allows to shorten the critical path for the radio turnaround time by excluding the host completely from it while still preserving the baseband sample access.

A. Channel Access Modes

Depending on the protocol type, *blocking the transmission* has to have different semantics. For random channel access, the sample stream must be stopped and should continue as soon as channel access is acquired. However, for time scheduled channel access, this behaviour would result in a loss in time synchronisation. Therefor, we propose two different modes of operation for the LBT controller that can be switched at runtime.

First, a *Random Access Mode* that behaves as described above. The controller delays the samples until transmission is permitted. Second, a *Time-slotted Mode* that replaces each sample with zeros until the channel is available, also known as *nulling*. This results in a non-decodable partial frame that is still aligned to the time slots and thus serves as an arbitrary reservation signal. During this reservation time, the host must check a special purpose register to see whether the frame needs to be repeated. This mode of operation implicitly requires an arbitrary reservation signal to be allowed by the used protocol. However, since a time-slotted protocol without any channel reservation would always be pre-empted in an random access environment such as communication in the popular ISM band, we think this is a reasonable assumption. A prominent example to support this claim is the LTE Licensed-Assisted Access extension that operates in unlicensed spectrum with a time-slotted channel and combines the LBT procedure with a reservation signal [8].

B. CSMA Parametrization

We see different CSMA procedures in use by modern protocols. Hence, a generalized implementation should be configurable in order to express those differences. For this design, we decided to restrict CSMA operation to *p-persistent* CSMA. This still leaves several degrees of freedom but is suitable for most major protocols using CSMA, most specifically for IEEE 802.11 [7], LTE-LAA [8], and IEEE 802.15.4 [20].

We assume the following basic procedure: First, a station observes the channel for a deferment period T_d. If the channel is free, the station may proceed to send. If the channel is busy, the station will wait until the channel is free for at least T_d and then enter the backoff phase. During the backoff phase, the station observes the channel and decrements a backoff counter N for each empty backoff slot T_{slot}. If during backoff the channel is busy again, then the station will enter the deferment period again but persist the backoff counter for the next backoff. If the backoff counter is zero, the station may transmit. The backoff slot N, the length of T_d, and T_{slot} as well as the energy detection threshold for determining the channel state should be set via software during runtime.

V. Prototypical Implementation and Validation

To demonstrate the feasibility of our LBT controller design, we implemented a prototype based on the *Ettus USRP X310* platform. We chose this platform specifically for its support of the RFNoC framework [12].

A. RFNoC in a Nutshell

The objective of RFNoC is to hide device specific hardware details behind a common structure, which then can be used to provide hardware functions for all USRPs that support the framework. The backbone of this common structure is the so-called *crossbar*. This is essentially a switch that connects different user-defined blocks, which are called *Computation Engines (CEs)*. These connections can be modified at runtime, allowing for high flexibility in the processing, similar to a flowgraph in, e.g., *GNU Radio*.

There are two special-purpose CEs that have external connections beyond the crossbar. The first one is the *Radio* block, which interfaces with the radio frontend and can either be a sink or source in the flowgraph. The second one is the *DMA FIFO* block. This is a large sample buffer to which the host can send its samples. Its main purpose is to compensate the jitter from the host side and to serve as steady source of samples.

RFNoC also provides a set of default blocks, that accomplish common tasks. The most important task necessary is sample rate conversion, since the radio frontend only operates at the device clock rate of 200 MHz. Therefore, RFNoC provides the *DDC* and *DUC* blocks (Digital Down/Up Conversion), which filter and decimate the signal and vice versa.

Internally, RFNoC blocks exchange packets of samples, which carry meta-data such as End-Of-Burst tags or a timestamp. Additional tools allow to convert the packet interface into an *AXI4-Stream* interface, which allows developers to execute functions on a per-sample basis. Every CE is also connected to a control channel from which block specific registers can be read or written. An application always interacts with RFNoC blocks via the USRP Hardware Driver (UHD).

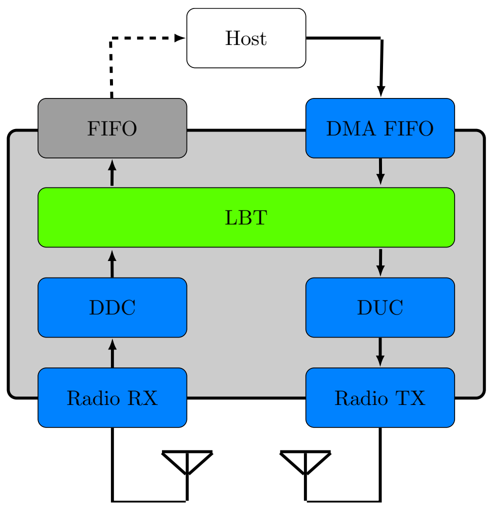

Figure 4. RFNoC flowgraph configuration for the LBT prototype.

B. LBT Block Design

We designed the LBT CE as a two input-two output block (cf. Figure 4).[1] The first input-output pair transports the downstream samples, while the second pair transports the upstream samples to the host. Technically, the second output is not strictly necessary, but having the same number of inputs as outputs simplifies the design because we can reuse more of the provided RFNoC tools.

The CSMA procedure is modelled as a finite state machine which is triggered by the first sample arriving at the transmission input in its idle state. The block maintains time synchronization by counting the number of samples since the beginning of the burst. Until the backoff is finished, the block will only output zero samples. The state machine is reset whenever the sample data is tagged with an End-of-Burst tag or the configurable maximum transmission time is exceeded. Each sample that is nulled during a transmission attempt is counted in a register, which can be polled by the host to determine whether or not the data must be resend by the host.

The LBT block can be configured via registers to set all necessary CSMA parameters such as the backoff slot, which should be generated randomly by the application in advance to every transmission. To use the block, we instantiate a flowgraph on the FPGA as seen in Figure 4. The DDC and DUC blocks are necessary since we can unfortunately not operate at the full rate of the FPGA. Each block has only two crossbar connections on which the inputs/outputs are multiplexed. Since the crossbar operates at 200 MHz, we cannot have a block with two 200 MHz connections directly connected to the radio blocks. The FIFO block only serves the purposes to circumvent a UHD bug that prevents two outputs from being in a different domain of the flowgraph (i.e., host domain vs. FPGA domain).

In order to demonstrate the advantages of our LBT Controller design, we integrated it with the LTE-LAA protocol

[1]https://git.tu-berlin.de/tkn/rfnoc-lbt

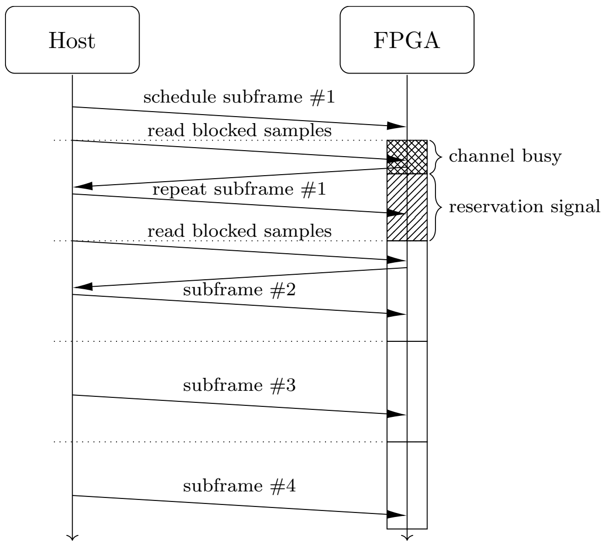

Figure 5. Example procedure for LTE-LAA downlink transmission in time-slotted mode.

stack for a set of initial experiments. In particular, we customized the *srsLTE* [3] software LTE stack.[2] We modified it to use our custom flowgraph on the FPGA to incorporate LBT by extending the driver-specific instructions for the interface to the UHD driver. Furthermore, we extended its downlink transmission sequence to enable retransmission of nulled LTE subframes and fixed the backoff slot to zero for a deterministic backoff. The transmission sequence from host to FPGA can be seen in Figure 5. The modifications are mostly transparent to the normal srsLTE functionality apart from checking whether or not channel acquisition was successful.

VI. Experimental Evaluation

We used our prototypical implementation of our LBT controller concept for an initial performance evaluation. For controlled interference, we used a signal generator loaded with a pre-computed LTE waveform, which transmits on a fixed duty-cycle to allow for gaps in the transmission. We placed our prototype USRP and the signal generator at a 2 m distance and placed a second USRP in the middle to observe the spectrum for post-analysis. We set the signal generator power level to 20 dBm to ensure that the energy detection would be triggered. To distinguish both transmissions in the post-analysis, we set the center frequencies slightly apart from each other at 2400 MHz and 2402 MHz, respectively.

A. Initial Results

The described measurement setup allowed us to measure the length of the transmission gaps in between the signal generator and the prototype transmitter over multiple transmission attempts and, thus, to infer the RxTx-turnaround time. In our collected traces, we did not find any instance where the prototype transmitted while a transmission from the signal

[2]https://git.tu-berlin.de/tkn/srslte-rfnoc

Table I
PACKETIZATION DELAY IN RFNOC FOR THE DEFAULT PACKET SIZE

Link	Link Rate	Delay
Rx Radio - DDC	200 MHz	5.10 µs
DDC - LBT	23.04 MHz	44.27 µs
LBT - DUC	23.04 MHz	44.27 µs
DUC - Tx Radio	200 MHz	5.10 µs
	Total:	98.74 µs

generator was on-going. Therefore, we can conclude that our prototype behaves correctly.

However, to our great surprise the observed gaps in the traces were much longer than anticipated. The measured gaps of 100 µs–145 µs seemingly contradict, for example, observations by Jiao et al. [19]. Furthermore, we observed that the gaps for any particular measurement run were consistent, i.e., always the same length, but they varied between two different runs of the experiment.

After some investigation, we were able to identify two main reasons for this long delay: The first one lies in the architecture of the RFNoC framework. Despite having a stream interface, which operates on a per-sample basis, the blocks exchange data via packets, i.e., sequences of samples. These data packets are transmitted at the sampling rate of the sample-emitting blocks, resulting in long packetization delays particularly at baseband sampling rates. By default, the packet size is set to 1020 samples per packet. For the critical path from RX-Radio to TX-Radio, this leads to a total delay of 98.74 µs as described in Table I.

The second source of delay in our prototype is the synchronization of the receiving and transmitting sample streams. Both streams are started separately through software, which results in a non-deterministic offset in the starting times. However, the LBT block only emits output samples when it has two valid samples at its inputs. This input synchronisation is necessary to avoid timing issues in the FPGA build. Thus, the block waits for the second input to receive a packet, causing a delay that is in between zero and the inter-packet arrival time. This depends again on the rate of the block input links. In our case that adds up to 0 µs–44.27 µs to the turnaround time. This delay changes every time the prototype is started but then stays consistent, once the streams run, which is consistent with the observed behavior.

Fortunately, both sources of delay can be addressed through software changes. First, we synchronized both sample streams, which can be achieved in UHD via so-called *timed-commands*, which will be executed at a specific clock time of the FPGA. This eliminates the synchronization delay. Second, we addressed the packetization delay by drastically reducing the packet sizes of the streams. We found that the minimal packet size is 90 samples per packet for the receive stream and 180 samples per packet for the transmitting stream. Any smaller packet sizes would lead to an unstable streams and packet underruns due to increased overhead. We also observed, that both packet sizes must be whole multiples of each other, otherwise new synchronization delay is introduced.

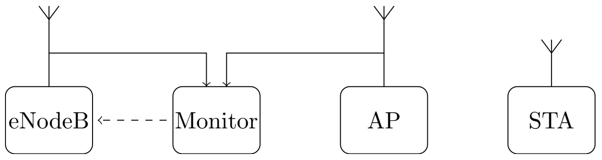

Figure 6. AP-eNodeB coexistence experiment setup.

With these optimizations, we repeated our initial experiment. This lead to an improved RxTx turnaround time of 19.2 µs. We can attributed 13.6 µs to packetization. The remaining 5.6 µs are unspecific. It is possible that this delay originates from the packet-to-stream conversion within the RFNoC blocks as described in [19].

B. Practical Lab Experiments

As a practical showcase, we evaluated a co-existence setup between our SDR-based LTE-LAA eNodeB prototype and a regular IEEE 802.11 Access Point (AP). A crucial point in such a scenario is the adaption of the contention window. Normally, the contention window of the eNodeB should be increased based on the number of UEs that acknowledge the transmission. However, we currently have no LTE-LAA capable UE at our disposal and are therefore missing a proper receiver. As a replacement, we set up a secondary USRP as a collision monitoring device. Using an RF power splitter with sufficient isolation across its outputs (20 dB), we connect each antenna port (AP and eNodeB) to separate inputs on the collision monitor. The setup is shown in Figure 6.

On the monitor device, we can capture the transmissions of both eNodeB and AP independently. An observed overlap in the transmission on the cable signals a collision on the air. The monitor is implemented using *GNU Radio* with a custom block that sends a UDP packet to the eNodeB upon a collision. For the AP and STA we used two *Linksys WRT3200ACM* set up for IEEE 802.11n with 20 MHz channel bandwidth. We use the 2.4 GHz ISM band, which is unused in our building for experimentation purposes. To generate Wi-Fi traffic, we use *iperf2* to saturate the channel with UDP traffic. For the eNodeB, we can estimate the achieved throughput by accumulating the non-nulled subframes and substracting the number of observed collisions.

The results are shown in Figure 7. The AP achieved an average throughput of around 42.6 Mbit/s, while the eNodeB only managed to send 19.4 Mbit/s, which is approximately a third of the maximum throughput for LTE-LAA in this single-antenna configuration. We also conducted a control experiment with a second AP replacing the eNodeB; here we observed an average throughput of 41.2 Mbit/s. Therefore, on average the impact of the LBT controller on a nearby AP is comparable to the impact of a regular, standard-compliant AP. As expected, we observed a relatively high collision rate of 0.146 collisions per LTE frame. This can be attributed to the non-compliant turnaround time of the LBT controller as shown earlier in Figure 1.

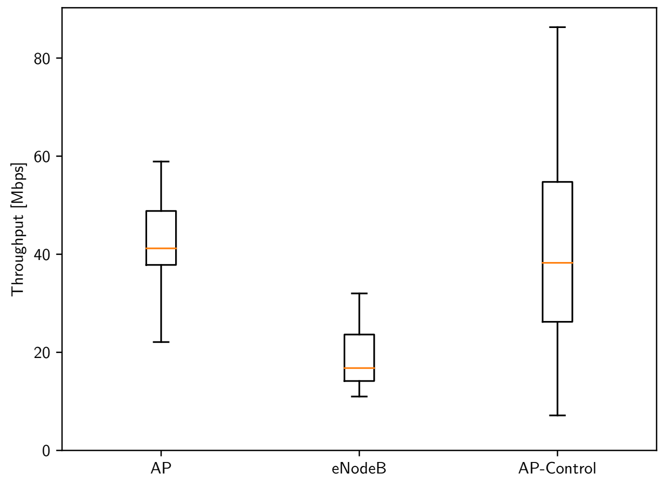

Figure 7. Box plot of the AP-eNodeB coexistence throughput measurement. Whiskers do not extend beyond 1.5 IQR.

C. Discussion

We need to stress that the throughput results from Section VI-B should not be considered for their absolute value. They serve only as a demonstration that our split-functionality approach is feasible and has the potential to match the performance of dedicated radio hardware. We also have to acknowledge that the achieved turnaround time needs further improvements for specific applications. To be within specification range of protocols like IEEE 802.11 or LTE-LAA, we would need a value less than 9 µs (the CSMA slot time). We see possible ways to improve the LBT Controller even further. Given that the packetization delay is the biggest contributor of delay, the key would be to reduce the critical path length even more. This could be achieved by removing the DDC and DUC blocks and integrating decimation, filtering, and upconversion into the LBT block itself, thereby removing to links in the RFNoC flowgraph. An increase in the FPGA clock rate could also lower overall delay. Thus, a further decrease in turnaround time of 10 µs–15 µs is within the range of possibilities. We therefore see our contributions as a starting point to enable future developments for SDR platforms in the unlicensed band.

VII. Conclusion

We presented a new approach for Listen-Before-Talk (LBT) on Software-Defined Radios (SDRs) that allows for rapid prototyping in software for protocol stacks which require some form of CSMA/CA. As proof-of-concept we present an LBT-enabled eNodeB prototype using *srsLTE* and X310 USRP. Compared to a software-only way implementation of LBT, we were able to reduce the critical turnaround time by a factor of four, down to 19.2 µs, by following the *split-functionality* concept proposed in [11]. To the best of our knowledge, this is the only LBT implementation capable of achieving such low turnaround times, while preserving a software-centric interface with direct access to the baseband samples. Furthermore, we laid out how this turnaround time can be improved even further, which would enable SDR software stacks for protocols like IEEE 802.11 or LTE-LAA.

References

[1] M. Dardaillon, K. Marquet, T. Risset, and A. Scherrer, “Software defined radio architecture survey for cognitive testbeds,” in *8th International Wireless Communications and Mobile Computing Conference (IWCMC 2012)*, Limassol, Cyprus: IEEE, Aug. 2012, pp. 189–194.

[2] Great Scott Gadgets. (2016). “Ubertooth One,” [Online]. Available: https://greatscottgadgets.com/ubertoothone/ (visited on 10/30/2020).

[3] Software Radio Systems. (Oct. 2020). “srsLTE - Your own mobile network,” [Online]. Available: https://www.srslte.com/ (visited on 10/30/2020).

[4] The Osmocom Project. (Sep. 2020). “rtl-sdr,” [Online]. Available: https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr (visited on 10/30/2020).

[5] Carles Fernández-Prades. (2020). “GNSS-SDR - An open source Global Navigation Satellite Systems software-defined receiver,” [Online]. Available: https://gnss-sdr.org/ (visited on 10/30/2020).

[6] GNU Radio project. (2020). “GNU Radio - The Free and Open Source Radio Ecosystem,” [Online]. Available: https://www.gnuradio.org/ (visited on 10/30/2020).

[7] IEEE, “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” IEEE, Std 802.11-2016, Dec. 2016.

[8] “LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures,” ETSI, Sophia Antipolis, France, Technical Specification ETSI TS 136 213, May 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/136200_136299/136213/13.12.00_60/ts_136213v131200p.pdf.

[9] Ettus Research. (Oct. 2020). “Ettus USRP E300 Embedded Family Hardware Resources,” [Online]. Available: https://kb.ettus.com/Ettus_USRP_E300_Embedded_Family_Hardware_Resources (visited on 10/30/2020).

[10] National Instruments Corporation. (2020). “What is LabVIEW?” [Online]. Available: https://www.ni.com/en-us/shop/labview.html (visited on 10/30/2020).

[11] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, “Enabling MAC Protocol Implementations on Software-Defined Radios,” in *6th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2009)*, Boston, MA: USENIX, Apr. 2009, pp. 91–105.

[12] Ettus Research. (Jul. 2019). “RFNoC,” [Online]. Available: https://kb.ettus.com/RFNoC (visited on 10/30/2020).

[13] Avada. (2013). “Nutaq ZeptoSDR Datasheet,” [Online]. Available: https://www.nutaq.com/sites/default/files/zeptoSDR-datasheet-lowres.pdf (visited on 10/30/2020).

[14] X. Jiao, W. Liu, M. Aslam, and I. Moerman, “openwifi: a free and open-source IEEE802.11 SDR implementation on SoC,” in *91st IEEE Vehicular Technology Conference (VTC 2020-Spring)*, Virtual Conference: IEEE, May 2020, pp. 1–2.

[15] “Real-time LTE/Wi-Fi Coexistence Testbed,” National Instruments, Whitepaper, Mar. 2020. [Online]. Available: https://www.ni.com/de-de/innovations/white-papers/16/real-time-lte-wi-fi-coexistence-testbed.html.

[16] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, “WARP: A Flexible Platform for Clean-Slate Wireless Medium Access Protocol Design,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 12, no. 1, pp. 56–58, Jan. 2008.

[17] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, “Performance Assessment of IEEE 802.11p with an Open Source SDR-based Prototype,” *IEEE Transactions on Mobile Computing (TMC)*, vol. 17, no. 5, pp. 1162–1175, May 2018.

[18] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, “Sora: High Performance Software Radio Using General Purpose Multi-core Processors,” *Communications of the ACM*, vol. 54, no. 1, pp. 99–107, Jan. 2011.

[19] X. Jiao, I. Moerman, W. Liu, and F. A. P. de Figueiredo, “Radio Hardware Virtualization for Coping with Dynamic Heterogeneous Wireless Environments,” in *International Conference on Cognitive Radio Oriented Wireless Networks (CrownCom 2018)*, Gent, Belgium: Springer, Sep. 2018, pp. 287–289.

[20] “Low-Rate Wireless Personal Area Networks (LR-WPANs),” IEEE, Std 802.15.4-2011, Jun. 2011.