Enabling Wideband, Mobile Spectrum Sensing through Onboard Heterogeneous Computing

Yilong Li, Yijing Zeng, Suman Banerjee
University of Wisconsin-Madison
{yilong,yijingzeng,suman}@cs.wisc.edu

ABSTRACT

We explore the design of a platform to support truly mobile and untethered, wideband spectrum sensing. First, the design of the platform needs to be physically small and lightweight. Next, we observe that wideband spectrum sensing (say >20 MHz at a time) can easily generate a large volume of PSD/IQ data, in uncompressed form (> 1 Gbps). Due to challenges of efficient offload of this large data volumes, we design a heterogeneous computing platform — using a combination of FPGA and CPU — built right on the spectrum sensor board, onto which various sophisticated compression algorithms, or wireless signal processing functions (even deep learning based ones) can be implemented. The FPGA is chosen to meet the real-time processing requirements of modern high-speeds wireless protocols, opening new opportunities. Finally, we provide easy connectivity to common mobile devices (currently Android phone) and a starting mobile app to enable easy programmability and control functions. Overall, our highly-integrated platform has the capability of sensing a wide frequency range of wireless signals with a high sampling rate and being controlled by a mobile phone via a USB OTG cable. We build a prototype of our system, and show through experiments that our device can support a bandwidth up to 56MHz and a wide frequency range from 70MHz to 6GHz for spectrum sensing, and run a deep learning model inference onboard for signal classification. We conclude by discussing the future challenges to realize large-scale spectrum sensing using our platform.

CCS CONCEPTS

• **Hardware** **Sensor devices and platforms**.

KEYWORDS

Mobile spectrum sensing, Heterogeneous computing

ACM Reference Format:
Yilong Li, Yijing Zeng, Suman Banerjee. 2021. Enabling Wideband, Mobile Spectrum Sensing through Onboard Heterogeneous Computing. In *The 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile 2021), February 24–26, 2021, Virtual, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3446382.3448651

HotMobile 2021, February 24–26, 2021, Virtual, United Kingdom

ACM ISBN 978-1-4503-8323-3/21/02…$15.00
https://doi.org/10.1145/3446382.3448651

1 INTRODUCTION

Distributed and wide-area, wideband spectrum sensing is a challenging problem. Traditionally, most efforts deploy a set of static spectrum sensors to gather Power Spectral Density (PSD) or IQ data, which are streamed to a connected desktop-grade compute node and eventually to cloud-hosted servers for analysis (e.g., Spectrum Observatory project [5]). Such static sensors naturally have a relatively limited coverage in the spatial domain. In contrast, when using mobile spectrum sensors, it is easy to see that substantial wide-area coverage can be efficiently obtained in time, space, and frequency domains that effectively complement static sensing [13]. In this work, we explore the design and use of mobile spectrum sensing platforms, where the sensors are low-cost, portable, and wideband, and can be placed in a flexible manner across a wide-area to gather a dense footprint.

Platforms common today: Most common low-cost spectrum sensing platforms use resource-limited spectrum sensing hardware (e.g. RTL-SDR, LimeSDR). Consequently, they usually have to be connected to a desktop-grade compute node, which provides necessary computing functions that are tied to spectrum sensing tasks. Wideband spectrum sensors capture a large volume of PSD or IQ data, leading to high throughput requirements (in excess of 1 Gbps) for any upload to cloud-hosted repositories. Therefore, most practical deployments of these spectrum sensors, when used for wideband sensing, rely on a co-located desktop-grade compute node to perform necessary manipulations and operations on the sensed raw data and. This, of course, is one reason why these spectrum sensors cannot operate in untethered ways. We note that a recent project, SparSDR [11] provides a spectrum data compression scheme to overcome some of these limitations, under the assumption that spectrum activity in the given band is sparse (which may not hold true for all spectrum bands, at all times, and locations), but we believe more can be done for wideband spectrum sensing using our proposed approach described in this paper.

Furthermore, these common platforms do not provide an ecosystem whereby they can be connected to a mobile device, draw power from it, and be adminstered and controlled by an app located on such a device. We believe such a structure would provide great flexibility in mobile wideband spectrum sensing.

Our design: With the development of integrated RF components and powerful processing units, we propose a new design for mobility, reconfigurability, and high performance for truly untethered, wideband spectrum sensing. Our platform uses an appropriate heterogeneous computing processor to address the aforementioned challenges. Different from existing commercialized platforms, we put wireless signal processing tasks in an onboard heterogeneous computing processor, Xilinx ZYNQ-7020, which combines an ARM-based processor with a programmable FPGA. Therefore, our sensing

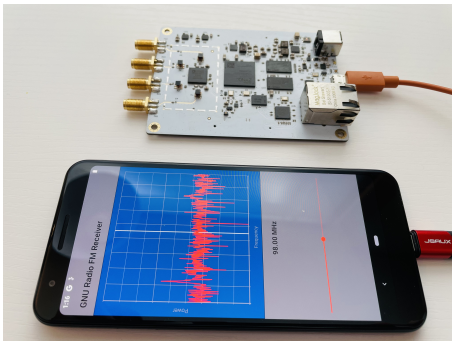


Figure 1: Our hardware connecting to a mobile phone. It has four antenna interfaces and two RF chains with the frequency range from 70MHz to 6GHz.

board enables the implementation of general-purpose analytics and hardware acceleration of wireless signal processing right on the sensor. In particular, with advanced needs of spectrum sensing functions today, it is also possible to deploy deep learning (DL) based functions, while integrating CPU, DSP, ASSP, and mixed signal functionality on a single device. The powerful heterogeneous computing capability also enables us to complete all necessary computations fully on our hardware to reduce the dependency of high throughput between the sensor platform and any offload sites (local compute nodes or cloud-hosted ones), and also removes the dependence of a capable co-located compute node.

To enable large-scale spectrum sensing based on common mobile devices, we also make our hardware compatible with the Android ecosystem by developing an Android driver compatible with the open-source toolkit project, GNU Radio. GNU Radio is a free and open-source software development toolkit that provides signal processing functions and wireless protocols implementations. Consequently, it is convenient for users to implement their own functionalities based on our driver or transplant applications developed by GNU Radio, and the users can control or reconfigure the sensors via a GUI on the mobile phone, as shown in Figure 1.

The structure of a spectrum sensing application using our platform is shown in Figure 2. The RF frontend uses an advanced highly-integrated transceiver with the capability of sensing a wide frequency range of wireless signals with a high sampling rate. The transceiver transfers digital and high-fidelity wireless samples to the digital processing controller via a parallel data bus provided by our FPGA. We remove all bottlenecks of the data path from the RF frontend to the wireless signal processing module. This enables our platform to sense and analyze a wide-range spectrum with a high sampling rate in real-time.

In addition, our platform makes extensive use of features of the onboard heterogeneous computing processor ZYNQ-7020 to accelerate deep learning applications. A deep learning model that classifies the signal types runs successfully on our platform to learn spectrum features. This provides new opportunities to bridge the gap between sensing hardware and deep learning applications.

There are still several challenges to enable large-scale spectrum sensing based on mobile phone users using our platform, which are described in the final section of this paper.

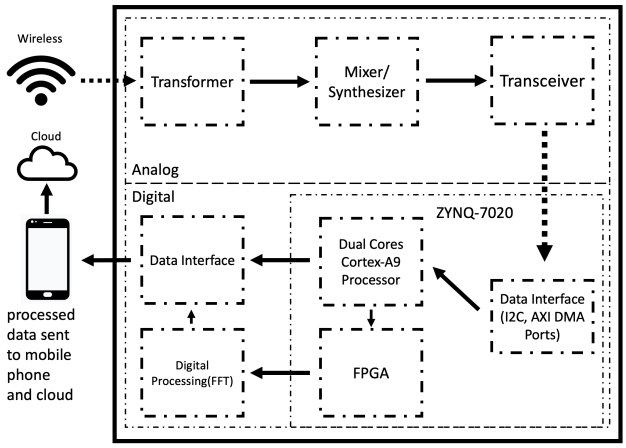


Figure 2: Overview of our prototype for spectrum sensing.

2 RELATED WORK

	RTL-SDR	LimeSDR	Ours
Freq. Range	22MHz-2.2GHz	100kHz-3.8GHz	70MHz-6GHz
RF Bandwidth	3.2 MHz	30.72 MHz	61.44 MHz
Sample Depth	8-bit	12-bit	12-bit
Sample Rate	3.2 MSPS	61.44 MSPS	61.44 MSPS
Duplex	No	Yes	Yes
Interface	USB 2.0	USB 3.0	USB OTG

Table 1: Comparison of existing sensing devices and our platform. RTL-SDR costs $20, LimeSDR costs $300, and ours are likely at the same range with LimeSDR.

Many existing commercialized spectrum sensors use desktop-grade machines to provide computation capability. The most common one is RTL-SDR [4], a USB dongle that can receive a very limited frequency range from 22 MHz up to 2.2 GHz. Mobile phones can also connect to an RTL-SDR dongle via an OTG interface.

LimeSDR [3], a popular platform, provides higher performance than RTL-SDR and uses an FPGA of Altera Cyclone IV EP4CE40F23. The biggest advantage of LimeSDR is that it leverages its own transceiver with a frequency range of 100 kHz to 3.8 GHz and a sampling rate of 30.72MHz. LimeSDR does not provide any support for a mobile phone connection, so it has very limited mobility. Moreover, it also needs computing capability from desktop-grade compute nodes. It is also currently limited by its USB interface bandwidth to deliver the maximal sampling rates possible.

Universal Software Radio Peripheral (USRP) [6] by Ettus Research is a traditional SDR series that is widely used and usually needs a connection to a PC host via USB or Ethernet. The PC host provides the computing capability for wireless sample processing. USRP B210 is a single-board 2x2 MIMO SDR and has a similar performance to LimeSDR. It depends on USRP Hardware Driver (UHD), which is deployed on PC host and writes firmware into hardware when it powers up.

TinySDR [10] is a newly presented SDR platform, which is tailored to IoT applications and provides Over-the-Air programming. It uses a low-cost transceiver integrated-chip AT86RF215 that has a very narrow frequency band. The frontend cannot capture wide range frequency wireless signals.

Snoopy [14] proposes the vision of using mobile phones for spectrum sensing. However, its prototype needs a bulky and costly frequency translator, which significantly constrains the mobility and the adoption of the sensor.

SparSDR [11] drops the requirement of high-throughput connection for data offloading by compressing the spectrum data. Nevertheless, it is based on the assumption that the captured signal is sparse, which is not always valid.

Different from the platforms above, we conduct a new design that uses a highly integrated transceiver to provide wide bandwidth sensing, support for a mobile phone connection, and sufficient computing capability onboard. The major difference between our hardware and previous works are summarized in Table 1. Although some commercialized hardware has comparable performance with our platform, we provide the flexibility that enables mobile connectivity.

3 SYSTEM DESIGN

In a typical programmable sensor's architecture, the analog RF frontend receives and/or transmits radio signals through an antenna. Protocol functionality and baseband processing are implemented in software usually in PC hosts, which provide substantial computational power. In spectrum sensing applications, the RF frontend's performance of the sensor is important. It needs a wideband sensing capability and a high resolution of sampled signals, but also results in a high data rate of IQ samples needs to be transferred to PC hosts for real-time processing. Therefore, a bottleneck exists. The bandwidth between different processing blocks limits the maximal data rate, and a high-throughput data interface between the RF frontend and the digital processing unit is needed. In this section, we present the design of our system and describe how we make design choices on the RF frontend and the digital backend respectively to overcome this bottleneck.

3.1 RF Frontend Design

Because all the signal processing is done in software, the RF frontend design in our sensing device can be rather generic. Our platform uses a custom-designed RF frontend that can sense a wide frequency band with a high sampling rate. The core of our design is a wideband transceiver, AD9361 from Analog Devices Inc. It receives signals from 70 MHz to 6.0 GHz and transmits signals from 47 MHz to 6.0 GHz range, with a tunable channel bandwidth from less than 200 kHz to 56MHz. Moreover, AD9361 provides four TX/RX channels and two RF chains, which enable 2×2 MIMO transmission. We, therefore, design an RF switch circuit so that the users can determine which TX/RX channel they would like to use. Given the maximal 56MHz channel bandwidth and 12-bit quantization, the RF frontend of our system generates a maximal data rate of 61.44Mbps, which needs to be supported by the interface to the digital backend.

3.2 Digital Backend Design

The RF frontend transfers a high volume of IQ samples data stream to the digital backend. The digital backend should be able to do signal processing on this high-throughput IQ samples data stream. Furthermore, high-throughput data interfaces between the hardware and the host (PC or mobile phone) and between the RF frontend and the digital backend are also necessary.

3.2.1 Main Processor. Our digital backend places the necessary computations and (de)modulation processes on a heterogeneous computing unit, Xilinx ZYNQ-7020, which is integrated with an ARM-based processor and an FPGA core. The 667MHz dual-core ARM Cortex-A9 processor of ZYNQ-7020 is integrated with a programmable logic (PL) on SoC. It provides a flexible design choice of offloading a small part of the computations, so we use it for system control and simple computation tasks. The 28nm Kintex-7 based FPGA of ZYNQ-7020 has hundreds of DSP units and can achieve up to 10T FLOP/s. As a result, we use it to support real-time wireless signal processing and high-throughput data transfer. The data transmission between PS (Processing System, ARM-based processor) and PL (Programmable Logic, FPGA module) goes through high-speed AXI interface on SoC which provides a large bandwidth. We achieve an aggregate bandwidth up to 3.8 GB/s full-duplex data transfer between the processor and memory. Overall, different from the platforms that use general purpose processors (GPPs) or individual microprocessors, the heterogeneous computing capacity provided by FPGA and ARM-based processor on a single chip ZYNQ-7020 enables high-throughput signal processing as well as flexible and robust system configuration.

3.2.2 Build the data path. Real-time data transfer depends on high-throughput interfaces. Our hardware can connect to both PC hosts and mobile phones for user interaction, although we mainly use mobile phones in practice. Data transfer from mobile/PC hosts to the digital control unit is based on kernel driver and interface controller. Users can use Android apps or Matlab/GNU Radio to send data and configure the SDR platform.

As shown in Fig. 3, the mobile or PC hosts uses USB interface to communicate with the hardware. Because ZYNQ-7020 SoC provides substantial computational power for signal processing and data streaming, our hardware does not depend on mobile/PC hosts' computational power and this greatly reduces the throughput needed for USB peripheral. We use a high-bandwidth USB 3.0 peripheral controller from Cypress, CYUSB3014, which controls data transmission and provides sufficient interaction speed with mobile/PC hosts. CYUSB3014 is also used for device verification when the device is powered up. In addition, USB controller can be re-used as a JTAG downloader to load the firmware to ZYNQ-7020 SoC. Currently, the firmware is not pre-loaded and is loaded from the host driver through controller when powering up.

Moreover, the data transfer between the RF frontend and the digital backend is charged by FPGA SPI/I2C, which can provide a substantial data streaming capability at data rates up to 6.25Gb/s (GTP) and 12.5Gb/s (GTX). Note that because the channel bandwidth of AD9361 is limited up to 56MHz and the sampling rate is 61.44MSPS, this data transferring throughput is not fully utilized.

3.3 Software

Fig. 4 shows the layered software architecture. We spend our major effort in developing both the firmware for our hardware and the driver for Android phones. The firmware is loaded into the ZYNQ-7020 co-processor and completes the operations of system control and

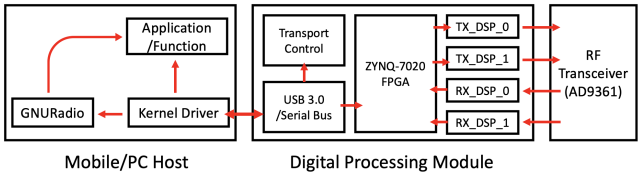


Figure 3: Digital backend design. Our platform provides an interface to both of mobile and PC host.

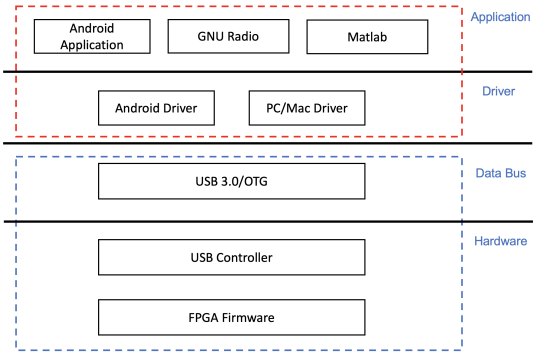


Figure 4: Software architecture.

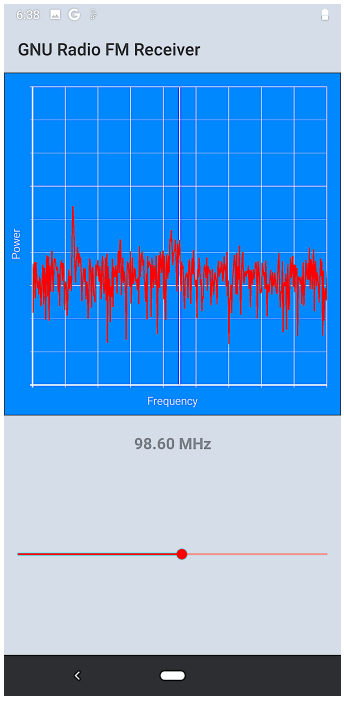


Figure 5: A Demo App based on GNU Radio is deployed successfully with our device.

synchronization of each hardware modules. The driver on Android phones supports developers to build Android apps that connect to our hardware device. Note that PC hosts are also supported but we omit for brevity.

3.3.1 FPGA Firmware. The firmware on FPGA is not pre-loaded before the board powers up. When the hardware device connects to a host, the driver first verifies the device and then loads the pre-compiled firmware into the FPGA's memory field through the USB controller. The DL models or other algorithms can be deployed on FPGA via Xilinx IDE after the driver is successfully booted. We are still improving the design and trying to make code deployment easier.

3.3.2 Android Driver. We develop an Android driver based on an open-sourced framework [8] which integrates USRP Hardware Driver (UHD) and GNU Radio so that our SDR platform can connect to Android phones via an USB OTG cable. Using the Android compatible GNU Radio framework, it is convenient for the community to develop Android apps that work on our hardware based on GNU Radio. Fig. 5 shows a demo Android app that is deployed on a Google Pixel 3a and displays the PSD of the spectrum obtained from our hardware device.

3.4 Power Supply & Consumption

There are two options of power supply for our platform. The first option is the USB power from mobile phone (or PC host). We believe this is most common one for our target users. The second option is the external power supply from power adapters or battery pack, connected through a dedicated onboard power connector. We support external power supply as the alternative power source for longer usage time or higher performance. The power interface is a standard connector which uses a voltage range from 5V - 12V and the input voltage will be regulated to 3.3V through a low-dropout regulator.

There are some power saving features supported by our hardware. The transmitter of our AD9361 transceiver frontend can be turned off during initialization and the receiver-only mode can save the power in spectrum sensing applications.

Our device consumes a power of 3W when it is powered through standard 5V voltage from USB and running at high performance. We can adjust the processing speed of PS (ARM-based processor) to further lower the power consumption. When the computation workload is light, the processor can operate at a lower frequency. In addition, if we disable the TX function in AD9361, the analog frontend will save a lot of power. If we enable these features in our hardware configuration, it can work at a power saving mode with a power consumption of 1.2W. If a battery pack containing four 18650 lithium-ion rechargeable batteries is used as the power source, it can support a whole day of consistent heavy usage of our device.

3.5 Comparison with Commercial Designs

There are a few commercially available SDR designs that contains the same main components with ours [1, 2]. They also use AD9361 as the transceiver and ZYNQ-7020 to provide some computational capabilities. However, there are several noteworthy differences. First, from hardware design's perspective, our platform can be powered and connected by mobile phone via USB, whereas they are designed to be connected by PC hosts. Second, from software design's perspective, we provide an Android driver such that the GNU Radio framework can be applied on top of it, which makes wide adoption in the research community possible; nevertheless, the commercial SDR platforms require the specific software from the manufacturer,

which cannot be easily extended with new functions. Last, we give the guideline that all the signal processing related computations, even complicated ones like DL model inference, should be placed on the onboard heterogeneous processor ZYNQ-7020; while the commercial SDR platforms still largely depend on the computation capability from PC hosts. Overall, we believe our platform supports mobility easily and has better reconfigurability and flexibility than these commercially available designs.

4 FEASIBILITY STUDIES

In this section, we showcase the performance of our platform in spectrum sensing application. We deploy several devices in a campus building to see if our platform meets the requirements of spectrum sensing. Furthermore, we show that our platform hardware successfully runs a DL model for wireless signal classification. Finally, we evaluate the usage time of our device when powering through USB.

4.1 Capturing a Bluetooth Signal

The Bluetooth signal hops in the band from 2.4 to 2.456GHz, which is divided into 80 channels. The frequency hopping is in a random fashion and at a rate of 1600 times per second, which makes capturing a single pulse very difficult because it needs high sampling rate of sensing hardware. Fig. 6 shows how the Bluetooth signal captured by our device hops to a different channel. Note that in this experiment, the frontend's sampling rate is set to maximum of 61.44Mbps.

4.2 Sensing GHz Wi-Fi Signals

Existing commercial Wi-Fi works in 2.4GHz and 5GHz band. 2.4GHz Wi-Fi signal occupies a 22MHz bandwidth and 5GHz Wi-Fi commonly occupies a 20MHz or 40MHz bandwidth channel. Sensing 5GHz Wi-Fi signals needs a hardware that has a wide bandwidth in both the frontend and the digital backend. Our RF frontend is based on AD9361, which can operate a tunable channel bandwidth up to 56MHz. Fig. 7 shows the captured 40MHz bandwidth 5GHz Wi-Fi signal when a computer is transferring a large file to a cloud server. Our wide bandwidth frontend and digital backend can capture the whole 5GHz W-Fi signal.

4.3 DL Onboard Inference

DL models have gained the attention from the researchers in the wireless community in the recent years. While GPUs offer parallelism for DL models, FPGA also has a great potential in DL model inference acceleration. We select signal classification as the example spectrum-related application that leverages DL models, and the long short term memory (LSTM) model proposed in [12] is the state-of-the-art model. Therefore, we use this model to show the possibility of running DL model inference on our FPGA. The LSTM model takes the averaged FFT of the captured wireless signal, calculated by the FPGA, as the input and tries to classify whether the captured signals are Wi-Fi or LTE signals. The LSTM hardware implementation is based on previous work [9]. Multiply Accumulate (MAC) unit are used to compute the matrix-vector multiplications and non-linear functions in LSTM computations. The corresponding gate hardware design is presented in Fig. 8. Direct Memory Access (DMA) ports are used to stream data in and out. We set the running frequency of ZYNQ-7020 as 142MHz and put all the computations on the PL (Programmable Logic or FPGA) rather than the Cortex-A9 processor in order to parallelize the computations, while the Cortex-A9 is still able to do basic data transfer and system control. The experiments are conducted with input size of both 128-point and 1024-point FFT data. We measure the throughput of FFT data segments, the execution time, and the overall system performance in terms of operations per second (op/s). Execution time on FPGA includes the time spent on data transferring and the time taken for computations in programmable logic for 1300 segments. Table 2 shows the performance metrics of DL model inference on our FPGA taking 128-point and 1024-point FFT data as input respectively. When ZYNQ-7020 operates at 142 MHz, LSTM taking 1024-point FFT data as the input can be computed at 235 M-ops/s by using simultaneously 4 AXI DMA ports [9]. The bottleneck when computing 128-point FFT data is that the memory bandwidth achieves the upper bound full-duplex memory transfer bandwidth of our hardware. This can be improved in the future hardware design.

Table 2: The performance metrics of LSTM model inference on FPGA

Metrics	1024-Point FFT input	128-Point FFT input
Throughput	1423 segments/s	2754 segments/s
Exec. Time	0.9s	0.47s
Performance	235M op/s	1.78G op/s

4.4 Power Consumption

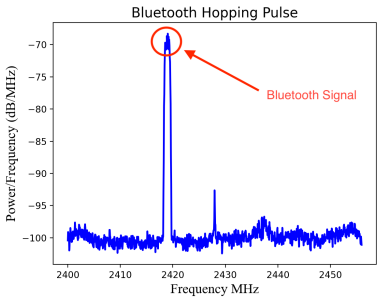
We also evaluate the power consumption of our hardware when it is powered by a Google Pixel 3a mobile phone through the microUSB interface. Google Pixel 3a has a 3000 mAh Li-Ion non-removable battery which can achieve 7 hours of battery life with just 15 minutes of fast charging. In the experiment, our platform can support 3 hours and 22 minutes of usage for spectrum sensing when Google Pixel 3a is the only power source, which matches the theoretical result 3000 mAh $\cdot$ 3.3 V/3 W = 3.3 hours. Note that the battery in Google Pixel 3a provides power to both the hardware device and the smart phone itself. If we use a battery box with four 3400mAh 18650 lithium-ion rechargeable batteries to power the hardware device only, it can support 3 days of consistent spectrum sensing at power saving mode.

5 CHALLENGES AHEAD

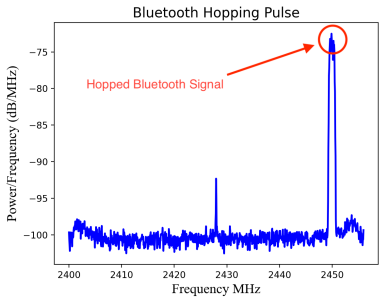
In this section, we discuss the technical challenges of wider adoption of our platform for fine-grained spectrum sensing. There are other challenges/future directions to explore, but we would like to work on the followings as the next stage of work since they are the most critical ones to boost adoption. With wider adoption of our platform, we also hope the community can bring up more interesting applications based on our platform, regardless of whether it is spectrum sensing related.

5.1 FPGA Programming & Computation Allocation

In order to have the ability of wide adoption, the sensing platform must be easy to program and support various wireless protocols and signal processing functions, ideally through an open-source



(a) A Bluetooth signal.



(b) The hopped Bluetooth signal.

Figure 6: A hopping Bluetooth signal captured by our device.

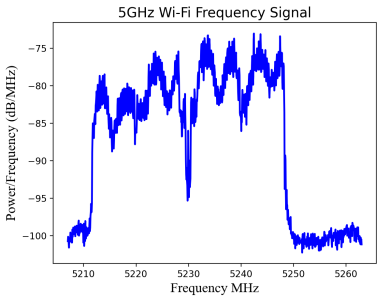


Figure 7: A 5GHz Wi-Fi signal captured by our device.

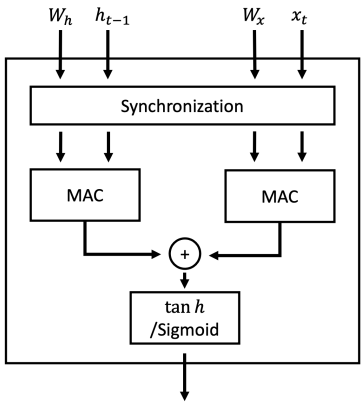


Figure 8: The hardware implementation of LSTM gates on our FPGA.

code base shared by a large community. However, currently, in order to run computations on the FPGA, one needs to write HDL code and load it to the FPGA via Xilinx Vivado [7], which is still somewhat cumbersome since it needs an additional Ethernet port for debugging. It would be much more convenient if there is a generator or tool chain to build developers' own hardware implementation, which translates high-level languages like python or C++ directly into HDL to realize the same signal processing/ML functions. We will also open source our firmware and driver when our hardware design is more optimized and finalized, from which we hope to motivate more research effort in the community using our hardware.

Moreover, when the FPGA is running some computationally intensive tasks, it would be ideal to leverage the computational resources and offload some workload at the mobile phone as well. This requires a dynamic computation scheduling mechanism between the FPGA and the mobile phone. Mobile phone's computational ability, load variations, energy consumption, etc. should all be considered when devising this mechanism.

5.2 Large-Scale Sensing by Mobile Users

With a mobile phone connection rather than a PC host connection, we believe our platform serves as an important step towards large-scale spectrum sensing by mobile users. We envision that it will be an integral part of fine-grained spectrum sensing in frequency, spatial, and temporal domains. However, this vision also brings several new challenges. For example, the privacy of mobile users' trajectories/locations can be a significant obstacle towards wide adoption among mobile users, because these meta-data are collected by the platform and usually transferred along with the spectrum readings to a centralized cloud to do further spectrum-related analysis. In addition to the trajectories/locations of mobile users, the captured spectrum data can also be sensitive since the platform may capture the communication data of the mobile users themselves. This may require a method to further differentiate whether the captured spectrum data contain users' communication data, if a centralized data analysis framework is used. Nevertheless, we foresee that a decentralized data analysis framework is inevitably necessary for the future, and Federated Learning, an ML paradigm that preserves data locality, can be a good potential solution. Note that Federated Learning requires on-device model training. This means a data scheduling mechanism is needed to decide when to transfer which part of the data between the mobile phone and the FPGA so that the best efficiency is achieved.

5.3 TX Mode as a Coordinator

At the initial adoption phase when the number of devices is small, we can turn one of the devices into the master of the sensor network and use it to coordinate the whole data collection process within the sensor network. This can be done by leveraging the TX function of our device and one only needs to choose the communication protocol and coordination procedure. Note that given the mobility nature of our sensor, one need to ensure that the communication range is large enough so that the master can cover all other sensors or handle the cases when this condition is not met. Moreover, the mobility constraint and the energy consumption of the master can be different from the other nodes.

ACKNOWLEDGEMENT

We thank our shepherd Junehwa Song and the anonymous reviewers for their feedback. All authors are supported in part by the following awards from US National Science Foundation: CNS-1647152 and CNS-1629833.

REFERENCES

[1] Commercial SDR design 1. https://www.aliexpress.com/i/33043152501.html.

[2] Commercial SDR design 2. https://www.aliexpress.com/i/4000890098857.html.
[3] Lime SDR. https://limemicro.com/products/boards/limesdr/.
[4] RTL-SDR. https://www.rtl-sdr.com/.
[5] Spectrum observatory. http://spectrum-observatory.cloudapp.net/.
[6] USRP. https://https://www.ettus.com/products/.
[7] Vivado. https://www.xilinx.com/products/design-tools/vivado.html/.
[8] B. Bloessl et al. Hardware-Accelerated Real-Time Stream Data Processing on Android with GNU Radio. In *ACM WiNTECH'20*, London, UK, September 2020. ACM.
[9] A. Chang et al. Recurrent neural networks hardware implementation on fpga. *arXiv preprint arXiv:1511.05552*, 2015.
[10] M. Hessar et al. Tinysdr: Low-power sdr platform for over-the-air programmable iot testbeds. In *USENIX NSDI'20*, pages 1031–1046, 2020.
[11] M. Khazraee et al. Sparsdr: Sparsity-proportional backhaul and compute for sdrs. In *ACM Mobisys'19*, pages 391–403, 2019.
[12] S. Rajendran et al. Deep learning models for wireless signal classification with distributed low-cost spectrum sensors. *IEEE TCCN*, 4(3):433–445, 2018.
[13] Y. Zeng et al. A framework for analyzing spectrum characteristics in large spatio-temporal scales. In *ACM MobiCom'19*, pages 1–16, 2019.
[14] T. Zhang et al. A wireless spectrum analyzer in your pocket. In *ACM HotMobile'15*, pages 69–74, 2015.