S(oC)DR: A Low Cost, SoC-Based Software Defined Radio Platform

Ajay Thakkar
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, New Jersey
athakka5@stevens.edu

Bernard Yett
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, New Jersey
byett@stevens.edu

I. Introduction

Modern RF systems require portability and computational power to solve complex problems in dynamic environments. Applications like remote sensing and tactical communications require self sufficient RF systems that can operate on the edge while employing modern algorithms such as spectrum sensing and signal classification. Creating such devices usually requires custom hardware and software that are tailored to the application, which is both a costly and non-reusable solution [1]. Technology such as Software Defined Radios (SDR) and System-on-Chips (SoC) can be leveraged in these applications to form fast, reprogrammable, portable, and cost effective systems [2].

SDRs are flexible RF systems that allow for digital reconfigurability to change its operational parameters while running. Their novelty lies in the interface between a General-Purpose Processor (GPP) and an RF front end, allowing the user to digitally control the front end's operating frequency, gain, sampling rate, etc. This GPP also allows for a personal computer to communicate with the system to send commands and retrieve return data for further processing [2]. Software such as GNU Radio and MATLAB include packages to communicate with popular SDRs as well as signal processing and visualization blocks that can be tailored to a wide variety of RF tasks. This SDR ecosystem has been used extensively in modern research for prototyping RF systems, like Hu et al. utilizing the Pluto SDR platform as well as MATLAB's deep learning toolbox to implement a Convolutional Neural Network (CNN) for signal classification of the Pluto SDR's return signal [3]. However, since many of these studies use a personal computer for their data processing which have a large form factor and utilize a lot of power, these systems are not viable for field testing or deployment.

To that end, many of these same signal processing applications such as CNNs have been implemented on programmable logic platforms like SoCs. SoCs contain both Programmable Logic (PL) and a Processing System (PS), essentially integrating both FPGA fabric and general purpose processors on the same chip. The processors can execute sequential logic and control many peripherals that are baked into the chip, such as dedicated UART hardware and GPIO pins. The FPGA fabric executes parallel logic which can implement tasks such as convolutions or Fast Fourier Transforms much faster than sequential processors. For example, Amin et al. were able to implement a CNN for signal classification on programmable logic, achieving an accuracy of 99.98% at 84139 frames per second with only 4.4W of power consumption, showing how aforementioned SDR research can utilize SoCs to lower power consumption and increase portability while still retaining the same processing power as a personal computer [4]. To that end, the focus of this paper is showing a real example of using an SoC with an SDR, and making this technology more accessible to research and commercial applications.

The novel contributions of this paper can be summarized as follows:

- Generate a PetaLinux image that can be deployed on the ZYBO SoC Development Board with necessary libraries, packages, and SDR utilities
- Create a hardware platform in Vivado that can transfer HackRF I/Q data to and from the PS
- Compile a custom interface to be used in the Linux image which can control the HackRF, interface with the hardware platform, and communicate the data over a UDP server
- Write an upstream program that can connect to the UDP server on the ZYBO, collect and visualize data, and send commands for controlling the HackRF

This paper is organized as follows: Section 2 provides a formally defined problem statement that motivates the work done for this project. Section 3 presents the work done in the project, including block diagrams, photos of the lab setup, and the hardware and software that was created for the project.

II. Problem Statement

SoCs and SDRs have already intersected in the commercial and research sectors. In the research realm, compact and low power systems that leverage software defined radio and SoCs already exist. The CROWN project funded by the European Defense Agency is a 2-18 GHz capable system that features Xilinx RFSoCs and is labeled as a small-scale, multi-functional RF system. However, due to existing research in this space frequently using custom hardware for its application or being tied to government funding, this technology is rarely

available for purchase for others to develop and test on [1]. In the commercial sector, Ettus Research creates Universal Software Radio Peripheral (USRP) SDRs which includes an SoC to allow for low-power, edge applications. For example, the USRP E310 is a 70 MHz to 6 GHz software defined radio that includes a Xilinx Zynq-based SoC, and is labeled as a "portable stand-alone software defined radio platform designed for field deployment". While this shows the existance of commercially available SoC/SDR platforms, the cost of these systems range from 5000 all the way up to 20,000 dollars for various Ettus products, limiting the availability for researchers to prototype and test on [5]. This paper endeavours to provide an easily accessible and low cost SoC/SDR development platform to help progress capabilities in portable, self-sufficient RF systems.

For the scope of this project, the viability of this low cost development platform will be shown through certain key functionalities that existing solutions have. Most RF applications require the ability to retrieve raw data from the SDR, tune parameters such as gain, center frequency, and sampling rate in real-time, implement user defined signal processing chains onto oncoming data, and transmit data over a network [1], [2]. By integrating these features into an accessible and cost-effective SoC/SDR platform called S(oC)DR, this project aims to lower the barrier to entry for researchers, students, and engineers looking to develop and test self-sufficient and portable RF systems.

III. S(oC)DR Development

A. Hardware Choices

To create a low cost SoC/SDR platform, the first design decision to make is what hardware to base the platform off of. Designing custom hardware is a part of why existing solutions cost so much, since the labor, manufacturing, and debugging of custom hardware is a very complex task. On top of this, the designer will need to provide support for their custom piece of hardware since there is no other reference for users to look at. To solve this issue, this project will use existing and separate SoC and SDR development boards. These development boards are already tested, have existing community support, and hold all the functionality that a single, custom piece of hardware could offer. Therefore, it was decided as a proof-of-concept to develop an interface between the Zybo Zynq SoC development board and the HackRF One SDR platform.

The HackRF One is an open source SDR platform developed by Great Scott Gadgets. It is a half-duplex transceiver that operates from 1 MHz to 6 GHz, meaning it can both receive data and transmit data in this band. The hardware is centered around an AD9361 RF transceiver chip that handles analog tasks, and an ARM Cortex-M4 microcontroller for digital tasks. It is powered and controlled via USB, and can be interfaced with using the "libhackrf" API that is maintained by the developers. Using this API, the user can adjust parameters like center frequency, bandwidth, gain, etc. The HackRF will operate at these user defined parameters, and provide baseband data of the signal serially over USB [6], [7].

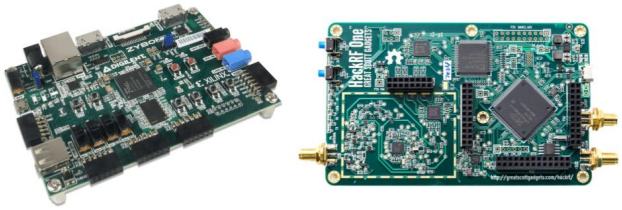

Fig. 1. Zybo (Left), HackRF One (Right)

The ZYBO is a Zynq-based SoC development board designed by Digilent. The board is centered around a Xilinx Zynq-7000 series SoC, which features a dual-core ARM Cortex-A9 processor and programmable FPGA fabric. The Zybo also includes on-board peripherals like DDR memory, USB, HDMI, Ethernet, GPIO, etc. The programmable logic is programmed via Hardware Design Languages (HDL) in the Xilinx Vivado Suite, and the processing system is programmed in Embedded C using the Xilinx Vitis Suite. The two can communicate via an AXI interconnect, allowing for seamless integration of the processor and FPGA fabric [8].

B. System Overview

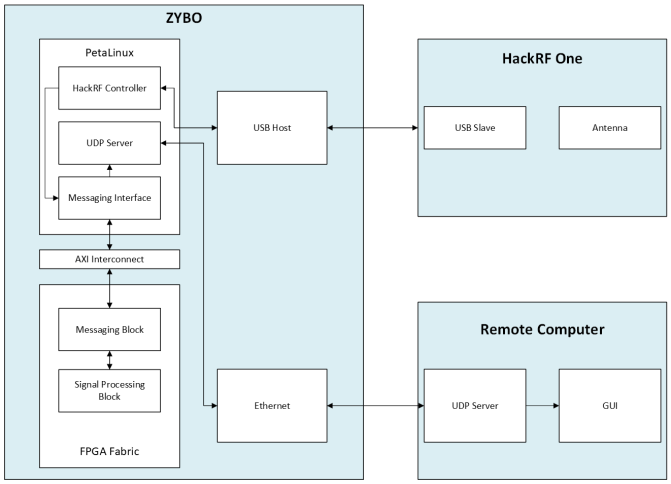

Fig. 2. System Diagram of S(oC)DR

To use the HackRF One and Zybo development boards to create a unified SoC and SDR development platform, a custom interface needs to be made for the Zybo that can control and communicate with the HackRF, use the data in the FPGA fabric for processing, and communicate this data to a computer for visualization and remote controlling. The implementation of this interface is laid out in Figure 2, which shows the blocks and connections necessary to fulfill these features. The Zybo features a USB On-The-Go (OTG) peripheral which can act as an embedded USB host that can both power and control the HackRF One. It also features an Ethernet port with a

MAC controller which can be used as a method to send data over a network to a remote machine. Internally, an embedded Linux kernel can be ran on the Zybo's processor and dedicated DDR memory for abstracting the device drivers needed to operate these peripherals. It will also make developing custom software that can interface with these peripherals much easier than having to write bare metal software. The software that will reside in the Linux kernel will be a HackRF Controller, a UDP server, and a messaging interface. The HackRF controller will use the libhackrf API to establish a stream of data from the HackRF into the program, as well as allow for a simple way to control the HackRF's operating parameters. The data stream in the controller will be given to a messaging interface, which will be able to communicate with the FPGA fabric via an AXI interconnect that is baked into the SoC. The data received from the FPGA fabric after processing will then be sent to a UDP server, which will be able to interface with the Ethernet peripheral to send frames of signal data over the network to a client machine for further use. It can also accept data from a client that will be sent to the HackRF Controller to remotely control the HackRF. The FPGA fabric will have a similar messaging block for taking in and formatting data from the AXI interconnect. This messaging block will be packaged as an IP block with a generic and reusable interface so that users can design their own custom signal processing chains in HDL that can easily interface with the messaging block and abstract the complexities of communicating with the Linux kernel over AXI [7], [8].

The deliverables for this project will be a Linux image that is pre-loaded with all the aforementioned software, as well as an editable Vivado project with the custom HDL that the user can edit to include their own signal processing chains. This will abstract all the complexities of having to interface between the Zybo and HackRF One development boards, and yield an extremely cost effective and powerful SoC and SDR development platform.

C. Development Environment

The development environment consists of an Ubuntu 22.04 operating system, PetaLinux 2017.4 which uses the Yocto Project Version 2.2, and Vivado 2017.2. The actual hardware and lab setup is shown in Figure 3, with a Zybo connected to a HackRF One via USB and an Ethernet cable connecting it to a local router. The Zybo is powered with a dedicated power supply to provide enough amperage to power the HackRF One. Another USB cable is used between the Ubutnu machine and the Zybo to remote into the Zybo's embedded Linux shell. A wide-band dipole antenna is used with the HackRF One to capture data along its entire band.

D. Linux Image

To deploy an embedded Linux kernel onto the Zybo, Xilinx has provided a toolkit called PetaLinux to help with the process for Xilinx specific targets, and its operation can be seen in Figure 4. Under the hood of PetaLinux is an open source build framework called Yocto, which uses other open

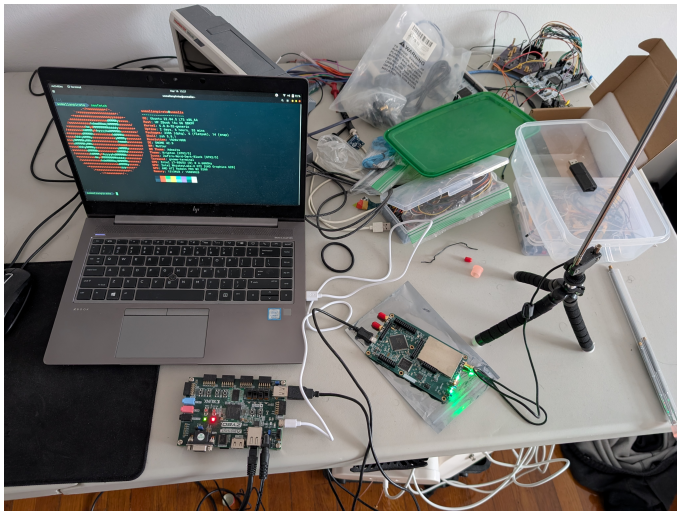

Fig. 3. Lab Setup

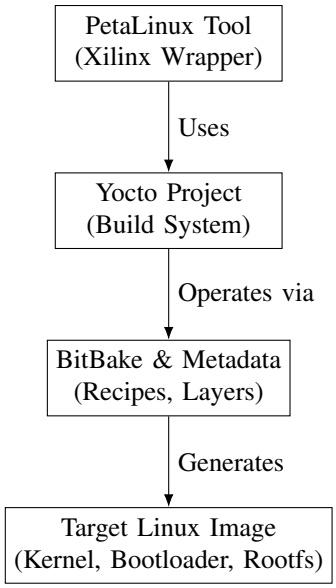

Fig. 4. PetaLinux and Yocto

source tools like BitBake, poky, and OpenEmbedded to make it easy to build custom Linux images for a variety of targets. Yocto is a layered architecture, meaning users can define custom layers for each library, piece of software, or device driver they want to add in the Linux image to allow for extremely granular control over the size and operation of the resulting Linux image. PetaLinux uses the Yocto project, and provides custom layers for Xilinx specific hardware like the Zynq-7000 series which is used in the Zybo. It also allows for a seamless integration between the Linux image and the FPGA fabric on Xilinx SoCs. After synthesizing a hardware platform for the FPGA fabric, an XSA file from the build artifacts can be used in PetaLinux to make sure the Linux image can interface with the hardware platform [9].

Digilent, the manufacturer of the Zybo, provides an example

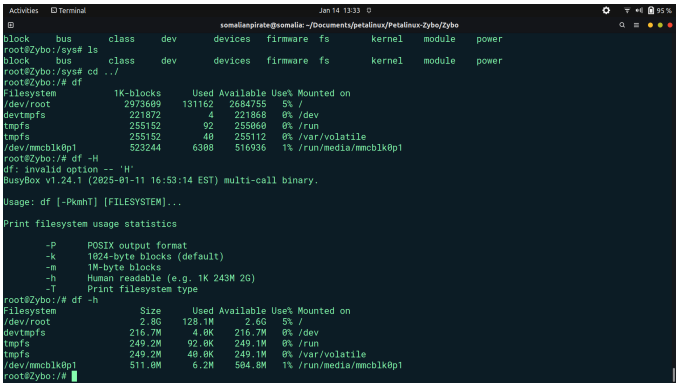

Fig. 5. Zybo Linux Shell

PetaLinux project and hardware platform that have pre-written device drivers and interfaces to use the board peripherals without any extra labor. Since this is open source, this is the starting point for this project in creating a Linux image and hardware platform that will include everything needed for an SoC and SDR platform. However, this example project was made in 2017, and has old dependencies and version conflicts that make it very hard to configure and build. After installing all the required dependencies, troubleshooting build errors, and making changes to Yocto layers, the example PetaLinux project was able to build and deploy onto the Zybo. In the final configuration of the PetaLinux project, an SD card is used to store the Linux kernel and the root filesystem to allow the DDR memory on the board to be completely dedicated to application memory. This will allow for a scalable size of storage space to store large chunks of signal data from the SDR, and as much memory as possible to run custom software. As can be seen in Figure 5, remoting into the Zybo's Linux shell shows the proper configuration and build of an embedded Linux image targetted for the Zybo development board [10].

E. Meta-SDR and USB OTG Peripheral

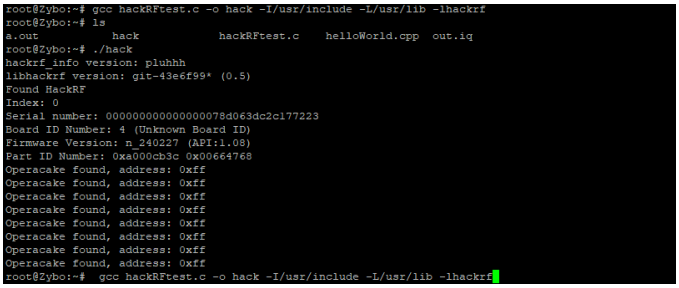

Fig. 6. HackRF Info Tool from Meta-SDR

Making custom software for an embedded ARM processor requires special compilation for the target, as most widely used compilers do not support embedded ARM architectures. Therefore, to use the HackRF libraries and other SDR tools this requires specific compilation of the libraries, and an inclusion of these compiled libraries into the root filesystem of the embedded linux environment. Thankfully, the OpenEmbedded community hosts open source Yocto layers that execute these steps for the developer in the Yocto build system. Meta-SDR is an OpenEmbedded layer which includes recipes, or scripts, for many popular SDR tools including the HackRF libraries. After tweaking the Meta-SDR layer to include a newer version of the HackRF library and modifying Yocto include files, the HackRF libraries and utilities were included into the root filesystem of the PetaLinux build [9], [11]. To test their functionality, the source code for the "hackrf_info" utility was copied into its own file on the embedded Linux target. The HackRF libraries in the root filesystem were linked during compilation, and the "hackrf_info" utility compiled and ran successfully as can be seen in Figure 6.

Since these HackRF utilities are now usable on the Zybo, this allowed for testing to be conducted on whether the HackRF can be powered and recognized from the USB OTG peripheral on the Zybo. USB On-The-GO (OTG) is a standard which allows embedded devices like smartphones or SoC development boards to act as USB hosts, essentially mimicing a standard desktop. The connection between the Zybo and HackRF using the USB OTG peripheral can be seen in Figure 3. A note in the peripheral's operation is that the HackRF requires up to 440 mA of current depending on its operation, meaning this peripheral must be able to supply this current at any given time. The Zybo reference manual states that the USB OTG peripheral can provide up to 500 mA of current through the peripheral only if the Zybo is powered from a wall supply. The implication of this is that the final system must utilize a bigger battery pack with an electrical outlet to enable this amount of current draw from the HackRF in a field environment. To test the data communication over this USB OTG peripheral, the "hack_info" utility compiled and ran previously in Figure 6 shows metadata being read from the HackRF such as serial number, firmware version, etc., showing a successful connection between the Linux kernel, the USB peripheral, and the HackRF [6], [8].

F. Future Work

Existing work created a working Linux environment with all the tools and libraries necessary for interfacing with the HackRF. Next, the software discussed in Section III-B must be written and tested, including the HackRF Controller, UDP server, and Messaging Interface. As well this, an effort should be made to edit the Digilent Vivado project that is used in the current PetaLinux build to include a custom Messaging IP Block.

REFERENCES

[1] J. C. M. Fernandez, A. C. Garcia, J. C. Soriano, J. L. G. de la Haba, M. Brandfass, J. Rauscher, J. Y. Dupuy, I. Le Roy-Naneix, P. Marquardt, S. Durst, P. Brouard, C. Martel, M. Thorsell, M. Sakalas, A. Nanni, T. Boman, M. Bartocci, and R. Van Dijk, "Crown - final results on a european multifunction aesa system demonstrator," in *2024 IEEE International Symposium on Phased Array Systems and Technology (ARRAY)*, pp. 1–8, 2024.

[2] D. M. Molla, H. Badis, L. George, and M. Berbineau, "Software defined radio platforms for wireless technologies," *IEEE Access*, vol. 10, pp. 26203–26229, 2022.

[3] L. Hu, H. Jiang, R. Lu, and C. Liu, "Signal classification in real-time based on sdr using convolutional neural network," in *2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, vol. 2, pp. 893–898, 2021.
[4] R. A. Amin, M. S. Amran Hossain, L. T. Schmid, V. Wiese, and R. Obermaisser, "Power efficient real-time traffic signal classification for autonomous driving using fpgas," in *2024 6th International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, pp. 1–5, 2024.
[5] Ettus Research, "Usrp e310," 2025. Accessed: March 9, 2025.
[6] Great Scott Gadgets, "Hackrf documentation." https://hackrf.readthedocs.io/en/latest/, n.d. Accessed: 2025-03-23.
[7] Great Scott Gadgets, "Hackrf - software defined radio." https://github.com/greatscottgadgets/hackrf/tree/master, n.d. Accessed: 2025-03-23.
[8] Digilent, Inc., "Zybo reference manual." https://digilent.com/reference/programmable-logic/zybo/reference-manual, n.d. Accessed: 2025-03-23.
[9] AMD, Inc., *PetaLinux Tools Reference Guide (UG1144)*, 2023. Accessed: 2025-03-23.
[10] Digilent, Inc., "Petalinux-zybo repository." https://github.com/Digilent/Petalinux-Zybo/tree/master, n.d. Accessed: 2025-03-23.
[11] OpenEmbedded Project, "meta-sdr layer — openembedded layer index." https://layers.openembedded.org/layerindex/branch/master/layer/meta-sdr/, n.d. Accessed: 2025-03-23.