

RadioStream: A Wideband Multi-Antenna SDR Platform

Adel Heidari^{1*}, Jeeva Keshav Sattianarayanan^{2*}, Rohith Reddy Vennam¹, Aaron Tartz¹, Agrim Gupta¹, Kishore Rajendran¹, Raviteja Devara², Lakshman Bhaskaran², Radha Krishna Ganti², Dinesh Bharadia^{1*}

¹University of California, San Diego ²Indian Institute of Technology Madras

{adheidari, rvennam, atartz, agg003, kirajendran, dineshb}@ucsd.edu, jeevakeshavs@smail.iitm.ac.in, rganti@ee.iitm.ac.in, {raviteja, lakshman}@wiseriitm.in

Abstract

Software-defined radios are central to wireless systems, but host-centric streaming becomes unreliable at high bandwidths, with many antennas, and when time-granular control is required. The root difficulty is a timing-domain mismatch between the FPGA/RF front-end's deterministic, clock-driven timeline and the host's asynchronous, OS-scheduled execution. The traditional stacks resolve this by forcing the host to keep pace with the radio. At wideband rates, this transfer results in underflows, overflows, and late packets. We present RadioStream, a radio-centric, slot-synchronous SDR architecture in which the host and radio coordinate using slots (a pre-defined unit of time) rather than individual sample transfers. The FPGA owns slot timing, time-critical radio actions, and exports slot boundaries as slot-tick Ethernet packets, while the host runs asynchronously behind slot-aligned buffers over a lightweight fronthaul with a look-ahead transfer for control and data path. The DPDK-based host path integrates directly with applications like GNU Radio, Python, and MATLAB on the other end. On a 100 Gbps interface, RadioStream sustains 90 Gbps concurrently on uplink and downlink with no packet drops, streams 2 GHz of aggregated RF bandwidth, bounds slot-tick jitter below 600 ns at the 99th percentile, and applies per-slot control, reaching the RF output in under 500 ns. We demonstrate these capabilities using communication, mmWave beam-sweeping, and sub-6 GHz multi-antenna sensing experiments.

CCS Concepts

• **Networks** → **Network experimentation**; • **Computer systems organization** → **Real-time systems**; • **Hardware** → **Signal processing systems**.

*The first two authors contributed equally to this work.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

MobiCom '26, October 26–30, 2026, Austin, TX, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2505-0/26/10.

<https://doi.org/10.1145/3795866.3844452>

Keywords

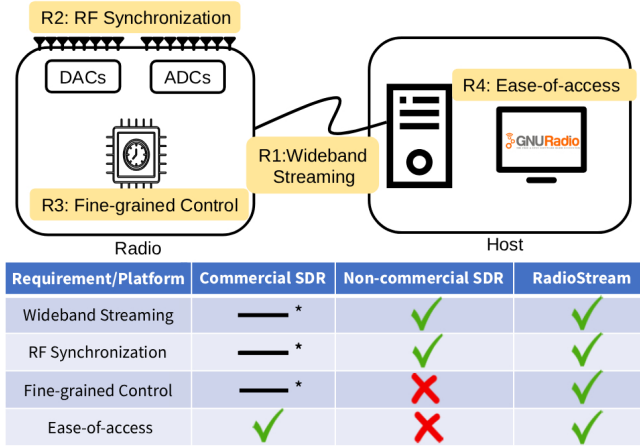
Software-defined radio, wideband radios, multi-antenna systems, MIMO, phase synchronization, real-time signal processing, wireless testbed

1 Introduction

Software-defined radios (SDRs) are no longer confined to research labs; they now underpin many deployed wireless systems. Emerging applications, however, place new demands on these platforms. Bandwidth-intensive applications such as Augmented and Virtual Reality (AR/VR), together with massive device connectivity, are driving 6G designs that exploit wide bandwidths in the upper mid-band and deploy massive antenna arrays with hundreds to thousands of elements [7, 29, 30, 45, 52]. Applications in Integrated sensing and communication (ISAC) further require tight time, frequency, and phase synchronization across antennas to support coherent multi-antenna sensing alongside communication on shared hardware [41, 55]. In addition, the growing adoption of open and programmable radio architectures, such as Open RAN (O-RAN), increases the need for flexible and fine-grained control over radio front-ends [39]. Applications such as beam sweeping, coordinated multi-antenna transmission, and adaptive jamming and anti-jamming operations [32] require this control at slot-level granularity.

The applications above place new demands on the SDR architecture. An SDR consists of a radio front-end, generally an FPGA coupled to RF data converters, that transmits and receives the over-the-air waveform and connects via a fronthaul link to a host, which is a general-purpose computer that runs the signal processing in software. These applications stress this link in four ways, as summarized in Fig. 1. Because these applications combine wide bandwidth with large antenna arrays, the raw IQ that must cross the fronthaul grows with both dimensions, reaching tens of gigabits per second. The link must therefore sustain continuous **wideband streaming (R1)** across all channels, so that processing can remain in flexible host software rather than being hardcoded on the FPGA. Applications like precise

Paper website: <https://wscng.ucsd.edu/radiostream/>



*only for high-end models

Figure 1: Figure showing requirements: (1) wideband streaming, (2) RF synchronization, (3) fine-grained slot-level control, and (4) ease of programmability. Traditional solutions only support 1-2 features, whereas RadioStream meets all four requirements.

localization and the joint sensing-and-communication further require **RF synchronization (R2)**: coherent alignment across channels in time, frequency, and phase, anchored to a single clock that originates at the radio and is shared with the host. That same shared clock must also support **fine-grained, slot-level control (R3)** of data direction, frequency, phase, and active antennas, timed precisely enough for coordinated multi-antenna transmission, beam sweeping, and jamming/anti-jamming operations to take effect at exact moments over the air. Finally, the platform must offer **ease of access (R4)** so that existing signal-processing frameworks, such as GNU Radio, can be reused with minimal effort.

Nearly all of these requirements converge on a single point: the connection between the radio and the host. This one link has to sustain wideband IQ streaming, keep both sides on the same clock, carry control commands, and deliver samples to external applications on time, all simultaneously and with minimal added overhead. The difficulty is compounded because the two ends of the link run on fundamentally different timing models. The FPGA and RF front-end operate on a deterministic, clock-driven timeline locked to the RF sampling clock, where every operation occurs at a known instant. The host, by contrast, runs in a non-deterministic, software-driven environment shaped by operating-system scheduling, interrupts, and queueing delay, so the instant at which it issues a command, consumes a sample, or hands that sample to an application is inherently variable.

Existing platforms place most of this burden on the host, which is why we call them *host-centric*: every reference to the radio, from reading its clock to timing a command, originates at the host, making the system as sensitive and fragile as the non-deterministic environment in which the host runs.

Commercial host-centric SDRs such as NI USRPs and Analog Devices Pluto [49] handle these tasks in host software through stacks such as UHD and GNU Radio [11]. NI USRPs, for example, regulate the sample stream with a credit-based flow-control scheme, where the FPGA grants the host a fixed number of credits for its small hardware buffer, and these credits are replenished by a continuous stream of credit-return messages. As Fig. 1 illustrates, this ties the pipeline’s pace to how quickly the host responds to those credits, so even an ordinary context switch, caused by OS scheduling, interrupts, or queueing delay, can leave the buffer running dry. This coordination traffic competes with the IQ stream itself, causing underflows and overflows that cap usable bandwidth at high rates. Commands timed for specific slots can also arrive late, forcing the radio to drop them or apply them out of sync, breaking system determinism.

To address these challenges, we propose **Radiostream**, a wideband multi-antenna SDR platform built on RFSoc in which the radio, not the raw sample stream, becomes the unit of coordination across the host–radio boundary. A slot is a pre-defined unit of time. The FPGA creates and exports time to the host at periodic slot intervals. The host then configures *future* slots against that timeline, and the FPGA applies all radio-visible updates only at slot boundaries. Control and data are both identified by the slot number. Slot-aligned buffering also absorbs transport delay and host jitter, making the system more robust. This contract lets the host remain fully asynchronous while preserving deterministic radio behavior, unifying host–radio timing, wideband streaming, and fine-grained control under a single abstraction. Radiostream builds this contract on the RFSoc architecture, tightly integrating the radio front-end with FPGA processing to sustain wideband capture and provide precise synchronization across antennas.

To realize this contract at wideband rates, we design a lightweight transport between the host and the RFSoc that carries both IQ samples and control information as explicit, slot-addressable ethernet packets. This transport minimizes control overhead and sustains high-throughput streaming even at large bandwidths across multiple antennas. Radiostream also offers precise phase synchronization across the radio front-end through tight clock synchronization. To make the platform accessible for rapid experimentation, Radiostream exposes a lightweight shared-memory interface that allows external applications to interact with the host controller, abstracting the interface to the radio. These interfaces integrate with existing signal-processing frameworks such as GNU Radio seamlessly. They can also be used directly from environments such as Python or MATLAB, enabling rapid prototyping for wireless research.

We implement Radiostream on XCZU48DR Xilinx RFSoc radio platform connected to a Linux host over a 100 Gbps

Ethernet link. Specifically, we prototype the architecture on two RFSoc boards. The first is the commercially available RFSoc4x2 platform, which provides a compact SDR system with integrated RF data converters and high-speed networking support. The second is a custom-built RFSoc platform designed for scalability, supporting up to 8T8R operation and equipped with two 100 Gbps Ethernet interfaces to sustain higher aggregate throughput. They allow us to evaluate RadioStream across both readily available hardware and a more scalable RFSoc deployment. Our evaluation shows that the system supports a baseband aggregated rate of 2 GHz, maintains slot-level timing alignment with 500 ns granularity, and achieves sustained fronthaul throughput of 90 Gbps using standard Ethernet packets. We further show that the platform integrates with external applications and GNU Radio-style processing while supporting wideband streaming, dynamic control, and multi front-end experimental workflows.

2 Background on RFSoc

The RF System-on-Chip (RFSoc) platform [6], developed by AMD, integrates programmable logic, embedded processors, and high-performance RF data converters within a single device. The integrated direct-sampling RF data converters make RFSoc a compelling platform for wideband RF and real-time streaming applications. As a result, RFSoc devices (i.e., ZCU111, ZCU208, RFSoc4x2, RFSoc2x2) have been widely adopted across diverse domains such as massive MIMO base stations [1, 23], phased-array radar systems [1, 36, 56], quantum computing [9, 17, 28, 31, 48, 50], satellite communications [43, 53], emerging terahertz sensing and imaging systems [54], and high-speed wireline communication [35], where high-bandwidth MIMO, low latency, and tight integration are critical.

2.1 Standalone RFSoc Architectures

Many existing RFSoc-based systems rely on relatively static, standalone architectures. Several prior works generate RF signals using precomputed waveforms stored in on-chip memory or rely on the embedded ARM processor to orchestrate control and waveform updates at runtime [16, 19–21, 34, 44]. SPEAR [20, 21] offers Python programmability on the ZCU216 up to 1.2 GHz but on a single channel with buffered (non-sustained) capture. In contrast, MIMORPH [34] processes up to 2 GHz aggregated across 4–8 channels in real time but fixes the PHY in the FPGA fabric and does not integrate with a more capable external host. Because of the SoC’s limited memory bandwidth and compute, this approach caps data throughput and introduces additional latency and timing variability, making deterministic real-time operation difficult. RFSoc’s tightly controlled timing has also been exploited outside communications, for qubit readout and control (QICK [50], QiCells [28], Qibosoq [17]), radar [36, 56], satellite [43, 53], and THz sensing [54], but

these are standalone control systems with low-rate (control-only) host links, confirming the platform’s deterministic-timing potential without addressing wideband host streaming. Consequently, these implementations operate with fixed waveforms, coarse control mechanisms, or tightly coupled architectures, limiting the flexibility required for dynamic wideband applications such as adaptive beamforming, multi-user MIMO, and integrated sensing and communication.

2.2 Host-Streaming RFSoc Architectures

To overcome these limitations, recent SDR designs on RFSoc platforms use high-speed serial interfaces, such as SFP+ and QSFP links to stream data between the FPGA and an external host with significantly larger memory and more computational resources. NI Ettus USRP radios [3, 4], among the most popular SDRs, and evaluation boards like the ZCU111 are built on the same Zynq UltraScale+ RFSoc. They rely on the UHD software stack and RFNoC framework [14] to stream data between the FPGA and the host over UDP/IP. In these architectures, software manages timing using host-side references rather than hardware counters synchronized to the RFDC clock, introducing timing jitter that increases with host load and reduces determinism in real-time experiments. Additionally, UHD’s multi-threaded transport model consumes multiple host CPU cores for packetization, flow control, UDP/IP stack processing, and timing management, limiting resources available for baseband processing. At the same time, RFNoC’s AXI-Stream and CHDR interface requirements constrain architectural flexibility for custom RAN pipelines.

Other works introduce custom FPGA streaming stacks in which the FPGA and radio act as secondary devices to the host [18, 27, 38, 42, 47]. Helix [42] and [18] demonstrate streaming platforms on ZCU208 and ZCU111 capable of supporting up to 417 MHz bandwidth, but they are limited to a single RF channel and do not provide slot-based control. [47] reaches an 80 Gbps host link on the RFSoc4x2 via UDP but experiences packet drops and provides neither an efficient multi-channel data-flow design nor downlink control from the host. [27], which is the closest to our work, introduces a radio-centric design enabling high-throughput data transport but does not support *burst sampling* in the receive path or provide simple interfaces for user applications. In contrast, RadioStream introduces a wideband MIMO SDR architecture with an optimized host–FPGA streaming framework that provides slot-level granular control, enabling deterministic real-time operation while sustaining high-throughput multi-channel RF data streams.

3 System Design

In this Section, we describe the RadioStream architecture that addresses the fundamental challenges: i) Host and radio

Table 1: Comparison of RadioStream with representative RFSoc-based SDR platforms.

Platform	Architecture	RF channels	Agg. RF BW	Host link / rate	Sust. streaming	Slot ctrl.	App. interface
SPEAR / SPEAR+ [20, 21]	Standalone (ARM/PYNQ)	1	1.2 GHz	1–10 G (buffered)			PYNQ/Python
MIMORPH [34]	Radio-centric (fixed PHY)	4–8	2 GHz (on-FPGA)	10 G (ctrl.)			MATLAB config
QICK [50]	Standalone (quantum ctrl.)	8	multi-GHz DAC	1 G (ctrl. only)			PYNQ/Python
USRP X410/X440 [3, 4]	Host-centric (UHD/RFX)C	4–8	up to 1.6 GHz/ch [check]	2×100 GbE	Partial (DPDK-UHD)	ns-level (no data) (SW timing)	UHD/GNU Radio
Helix [42], [18]	Host-centric custom stack	1	417 MHz	100 G-class			GNU Radio (limited)
100G offload [47]	Host-centric (UDP capture)	multi	—	100 G (80 Gbps, lossy)	(drops)		PYNQ/GNU Radio
[27]	Radio-centric	multi	wideband	100 G	(no Rx burst)		Custom (none)
RadioStream	Radio-centric, slot-synchronous	8T8R	2 GHz	100 GbE, 90 Gbps		(0.5 ms slots)	GNU Radio/Python/MATLAB

time synchronization, ii) wideband streaming, and iii) Fine-grained radio control introduced in Section 1.

RadioStream allows a non-deterministic host to drive a deterministic wideband radio without sacrificing transport efficiency or fine-grained control. This is a common problem that arises in modern SDRs, as they sit at the boundary between two incompatible execution domains. On the radio side, deterministic clocks drive the design of data-processing and control modules, while on the host side, operating system schedulers and interrupts drive software processing. So, the initial synchronization established at the start of the session, or less-frequent synchronization between the host and SDR, doesn't hold over the interval, and delays in the control commands reaching the RFSoc lead to packet drops.

RadioStream resolves this tension with a slot-synchronous host–radio contract, where the slot, not the raw sample stream, is the unit of coordination across the boundary. The FPGA generates and exports slot time; the host programs future slots on that timeline, and the FPGA applies all radio-visible updates only at slot boundaries. Control and data are both tagged with their slot number, and slot-aligned buffering absorbs transport delay and host jitter. Together, these mechanisms allow an asynchronous host to operate without sacrificing deterministic radio behavior.

Fig. 2 describes the complete architecture of the RadioStream system and the components needed to achieve the above principles, comprising two tightly coupled components: the RFSoc and the Host Server. On the host side, a custom DPDK-based framework controls high-speed fronthaul transport with the RFSoc and exposes a shared-memory interface to external applications such as GNU Radio and higher-layer radio software. On the device side, the RFSoc implements deterministic timing, slot-aligned IQ sample buffering, and runtime radio control while interfacing directly with the ADC and DAC channels. Together, these two components achieve the core design goals introduced in Section 1: the FPGA remains the timing master, the fronthaul stays simple and efficient, and control is expressed explicitly at fine timing granularity.

Subsection 3.1 introduces the data flow in the uplink (Radio-to-Host) and downlink (Host-to-Radio), highlighting the main modules in the RFSoc and the host. Further subsections describe the features of the RadioStream architecture in detail.

3.1 Architecture of System Design

The FPGA serves as the primary timing source. In contrast, the control (for timing, RF parameters, antenna configuration) is done by the host application at a granular level without sacrificing any functionality. The following four FPGA modules play a critical role in achieving this:

Slot tick generation: The *slot_tick_generator* IP generates a pulse for every finite duration, and we term this duration a slot. This module runs an internal counter on the positive edge of each cycle of a clock frequency synchronized to the RF sampling clock. This module then generates a slot tick for every slot duration comprising an SFN (System Frame Number), which iterates from 0–1023, and a Slot Number, which iterates from 0–19. A slot tick is generated every 0.5 ms ($122880 \text{ cycles} \div 245.76 \text{ MHz}$) with an incrementing slot number. Here, 0.5 ms is the slot duration, and the 245.76 MHz clock used is frequency-synchronized to the RF sampling clock. The SFN is incremented once every 20 slots. The SFN is a coarse identifier, and the Slot Number is a fine-grained identifier for each slot. The notion of Slot Number and SFN is inspired by the 5G NR protocol, where timing is critical, particularly in 5G TDD systems. The slot is the minimum duration for which the configuration parameters must remain constant; the slot duration cannot be changed at runtime and can only be changed at compile time.

Data Plane - Uplink Chain: The uplink data is received at the antennas, sampled with ADCs, and decimated by the *Xilinx RF Data Converter IP* at 10X, then further decimated by 1X, 2X, or 4X based on the chosen channel bandwidth. For a 122.88 MHz channel bandwidth, the RF signal is sampled at 4.9152 Gsps, decimated by 10X in the *Xilinx RF Data Converter IP* and further decimated by 4X to produce IQ samples at 122.88 Msps, with each IQ sample represented by a 4-byte word. This translates to $122.88 \text{ Msps} \times 32 \text{ bits} = 3.93 \text{ Gbps}$ of IQ data throughput in the fronthaul link.

The *rx_gate* IP performs two functions: i) permits the received samples during the Uplink slot and drops otherwise, ii) concatenates the received IQ samples from a maximum of 8 RF streams and lets the *slot_buffer* store them in the DDR memory. The stored IQ samples are read in the next slot. Associated Ethernet headers, slot numbers, and SFN identifiers are appended, and the samples are packetized to an MTU of 9000 bytes and sent to the 100G Ethernet Subsystem by the *ethernet_packetizer* IP for transmission over the QSFP28 optical link.

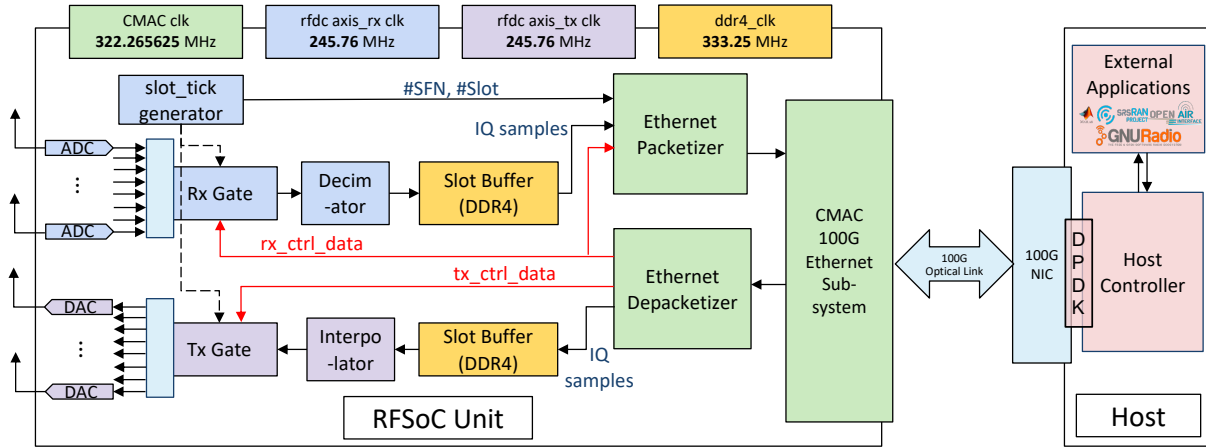


Figure 2: SDR Architecture with RFSoc, Host Unit(X86 based Server) and 100G link; RFSoc includes slot_tick generator for time synchronization, Gating modules for over-the-air time alignment, Decimators and Interpolators to vary channel bandwidth, DDR4 based Slot Buffers, Packetizer to fragment and add headers to slot_tick and Uplink IQ samples, Depacketizer to combine the ethernet frames and demultiplex control and downlink IQ samples; Host Server consists of Host controller application that interfaces with i) the RFSoc for high-speed fronthaul transport with a custom DPDK-based framework and ii) the external application via a shared-memory region.

Data Plane - Downlink Chain: The received frames are depacketized in the *ethernet_depaketizer* IP, and the IQ samples are stored in DDR memory, with SFN and Slot Number serving as identifiers that map to the memory addresses.

The *tx_gate* IP performs two functions similar to the *rx_gate* IP: i) backpressure the samples and align the transmission to the designated slot, i.e., hold the samples until the slot identifier matches the slot tick generated by the slot_tick generator, and ii) deconcatenate the IQ samples corresponding to each antenna port. The digital samples are then interpolated in two stages, similar to the decimator, and upsampled by the *Xilinx RF Data Converter IP* for transmission.

Control Plane: The control plane delivers slot-level radio configuration information from the Host to the RFSoc deterministically and time-aligned. Control exchange is tightly coupled with the slot-based timing mechanism described earlier. The depacketizer module extracts the control-plane configuration packets from the Host and distributes them across all IPs in the FPGA to process the IQ samples in the scheduled slot.

Host: The Host PC consists of two main components to establish a streaming interface and is described as follows:

Host controller: This application loads the per-slot configuration from a predefined array. It sends control packets and downlink Ethernet frames for downlink slots over the DPDK-enabled port for low-latency, high-throughput transfer. It also receives the corresponding frames from the FPGA for the uplink slots. The host controller manages two interfaces: i) a DPDK-enabled 100G interface for fronthaul communication with the RFSoc, and ii) a shared-memory region for

data transmission between the controller and the external application.

External Application: This is any application that runs on the same host machine, generates data for downlink transmission, and receives data from the uplink reception. Later sections of the paper show different applications, such as GNU Radio, that can interface with the host controller.

Note that both the host controller and the external application are assigned to different physical cores of the PC for efficient implementation. The host controller will require two CPU cores: one dedicated to downlink data generation and the other to uplink data reception.

3.2 Custom Fronthaul Interface

The fronthaul interface between the Host and the RFSoc is implemented as a custom high-speed Ethernet link via QSFP28 ports. On the FPGA side, the Xilinx CMAC (100G High Speed Ethernet Subsystem) IP handles packet transmission and reception, along with custom packetizer and depacketizer logic. On the host side, a custom DPDK application bypasses the kernel networking stack. It interfaces directly with the NIC queues in user space to maximize throughput and reduce software overhead.

Three types of packets are exchanged over the fronthaul:

- **Slot-tick packets**, which are fixed 64-byte packets carrying the current slot number and system frame number from the FPGA to the Host.
- **Data packets**, which transport downlink or uplink IQ samples and are packetized using a large MTU to reduce transport overhead.
- **Control packets**, which are fixed-size packets sent from the Host to the FPGA and carry slot-scoped radio

configurations such as MIMO state, data direction, and NCO parameters.

The three packet types serve separate functions. Timing is exported through slot ticks, IQ traffic is moved efficiently through large data packets, and control is separated so that runtime radio updates do not have to be inferred indirectly from the data stream.

3.3 Timing Management

Because the FPGA acts as the timing authority rather than the Host, accurate time synchronization underpins the entire architecture. In our setup, the RFDC clocking fabric derives the timing used for slot generation, and the *slot_tick_generator* RTL block maintains the current slot number and system frame number (SFN). For the current implementation, we use a slot duration of 0.5 ms, and the slot tick is generated once per slot as a 64-byte Ethernet packet sent from the FPGA to the Host.

Each slot tick contains two primary identifiers: (1) System Frame Number (SFN), a 10-bit field that iterates from 0 to 1023, marking a duration of 10 ms, and (2) Slot Number, a 5-bit field that iterates from 0 to 19, marking a granular timing of 0.5 ms. These slot numbers and SFNs are used as identifiers in control packets to configure the corresponding data packets. These slot tick packets are the highest-priority Ethernet packets flowing from the FPGA to the host PC.

On the host side, the DPDK controller decodes each slot tick and uses it to decide what to do: prepare downlink samples, consume uplink samples, or issue slot-specific control metadata. On the FPGA side, the same slot boundary drives the ADC and DAC gating logic. This design keeps radio timing deterministic without requiring host execution in lock-step with the FPGA clock.

3.4 RF synchronization

Timing adjustments for transceiver delays: The ADCs and DACs introduce fixed delays of $\Delta_{ADC} = 340$ ns and $\Delta_{DAC} = 280$ ns between their digital interfaces and the antenna, characterized empirically via oscilloscope measurements from the digital trigger to the RF port. To align transmission and reception with the slot tick, we advance the transmit path by Δ_{DAC} and delay the receive path by Δ_{ADC} in the *tx_gate* and *rx_gate* RTL modules.

Phase synchronization across transceivers: Coherent multi-antenna operation requires every RF channel to share a fixed phase relationship, even across RF-DC tiles, which otherwise drift due to independent clock and FIFO skew. RadioStream enables the RFSoc's multi-tile synchronization (MTS) feature [5] to align tile clocks and datapath latencies to a common reference, so channels on different tiles behave as one clock domain.

3.5 Control Plane and Radio Configuration

The control plane deterministically delivers radio configuration information from the Host to the FPGA. To tolerate transport delay and timing jitter along the control path, we transmit a single 64-Byte control packet with a two-slot look-ahead so that the configuration for slot n arrives at the FPGA in advance, i.e., at slot $n - 2$. For our chosen slot duration of 0.5 ms, the maximum control-packet execution delay is two slots (1 ms). The received control parameters are stored in a control buffer and applied only when the corresponding slot boundary is reached, as indicated by the slot-tick signal provided to the respective IP blocks (i.e., RFDC, *rx_gate*, and *tx_gate*).

Each control packet carries the slot number and System Frame Number (SFN), together with the slot-scoped radio parameters. The slot number specifies the target slot derived from the slot tick received from the FPGA, while the SFN identifies the corresponding target system frame. The packet also specifies the antenna configuration, including the number of active antennas and the data transfer direction: uplink (FPGA to host), downlink (host to FPGA), or duplex (both). The packet also includes the NCO frequency, which defines the numerically controlled oscillator frequency used for digital upconversion and downconversion, and the NCO phase, which provides the phase control information needed to maintain phase continuity across slots.

By sending explicit control packets rather than overloading the data stream with implicit semantics, the platform enables rapid per-slot updates to data direction, active layers, and RF parameters while maintaining a simple and deterministic host-FPGA contract.

3.6 Buffer Management for Uplink and Downlink IQ Samples

Slot-aligned buffering is used on both the Host and the FPGA to decouple fronthaul transport from real-time RF operation. All IQ data packets are tagged with the corresponding slot number, SFN, and transfer direction so that data and control can be associated unambiguously on both ends of the link, even when packet arrival times vary.

For the downlink path, IQ samples for slot n are prepared on the Host in advance and sent with a two-slot look-ahead. For our chosen slot duration of 0.5 ms, the maximum delay in the transmission of the data payload is two slots (1 ms). The FPGA depacketizes the received samples, stores them in DDR, and releases them to the DAC path only when the slot identifier matches the current slot tick. For the uplink path, the FPGA captures and buffers IQ samples during the active slot, packetizes them at the slot boundary, and forwards them to the Host as a coherent slot-sized data unit.

This buffered transfer is essential to bridge the deterministic RF domain and the non-deterministic host domain. In

the current implementation, at least four slots of buffering are needed to accommodate two uplink and two downlink slots safely. For 8 channels of 122.88 MHz IQ data sampled at 122.88 Msps with 16-bit I and 16-bit Q samples, this corresponds to approximately 64 Mb of buffering, which motivates the use of external PL-DDR rather than on-chip BRAM or URAM. The *FIFO_MIG_Controller* RTL block implements buffer management, including fragmented DDR transfers, address generation, and read/write tracking.

Buffering at least two slots on the uplink and downlink is a system requirement for pipelined data transfer and to absorb scheduling jitter on the Host. Buffer sizes can be reduced for low-latency applications by decreasing slot duration.

3.7 Host Software and External Applications

The host software stack consists of two components: the Host controller and the external application. The DPDK controller handles port initialization, queue setup, MTU configuration, packet parsing, and slot-aware packetization/depacketization. Because the platform supports external applications rather than a single fixed PHY, the DPDK controller exposes shared-memory regions for downlink and uplink traffic. These regions provide an internal user-space bridge between application logic and the fronthaul endpoint.

DPDK-enabled host application: The kernel normally controls data transfer through a NIC, arbitrating across multiple processes, routing through the network stack, and handling firewall processing. These mechanisms introduce non-deterministic latency that becomes a bottleneck at the packet rates required for wideband IQ transport. The host-side DPDK application bypasses the kernel entirely, giving the application direct responsibility for queue and buffer management via polling. The implementation operates as follows.

The port is configured with a single RX and TX queue, with ring sizes of 4096 descriptors each, backed by an mbuf pool of 16384 entries with a per-core cache of 512 mbufs. The MTU is set to 9000 bytes for jumbo frame support, and promiscuous mode is enabled to receive all frames on the point-to-point link regardless of destination MAC. A pre-built 14-byte Ethernet header template with a custom EtherType (0xAEFE) is used to avoid header construction overhead.

The main loop calls `rte_eth_rx_burst()` in a tight polling loop with a burst size of 64 packets. Each received packet is classified by the Payload type field in the packet header: slot-tick packets (0X0004), uplink IQ data packets (0X0001), and downlink control packets are handled in separate branches within the same burst processing loop. Upon receiving a slot-tick, the Host decodes the (SFN, slot) tuple from the payload, determines the scheduled direction for slot $n+2$ from a pre-loaded slot schedule array, and immediately constructs and transmits a 64-byte control packet

carrying the direction mask, MIMO configuration, sample count, and NCO frequency for the target slot. For downlink slots, IQ samples are read from the external application from the downlink shared-memory ring, then fragmented into 8896-byte payload packets using pre-allocated mbufs reused across slots to avoid allocation overhead. Uplink IQ payloads are accumulated from successive `rte_eth_rx_burst()` calls into a contiguous buffer and published to the external application in the uplink shared-memory ring once the slot's expected byte count is reached.

Shared-memory handoff: The uplink and downlink regions are each organized as a ring of slot-sized buffers, one slot deep, mapped into both the host controller and the external application. The producer writes a completed slot into the ring and then publishes its index, which the consumer observes before reading that slot. Because the index is published only after the payload is written, a consumer that sees a slot marked ready is guaranteed to see it complete. The ring depth also serves as the flow-control window, bounding how far the producer may run ahead of the consumer. The handoff, therefore, involves no system calls and no copy through a kernel buffer.

External applications: External applications interface with the DPDK host application through shared memory. Both processes map the same region, which carries data from the external application to the DPDK host application for downlink slots and in the reverse direction for uplink slots. Because the transfer involves no kernel mediation or system calls, it preserves a clean separation between application logic and fronthaul implementation while still supporting high throughput. This exposed interface can be used to connect GNU Radio, srsRAN, OAI, or custom DSP pipelines.

We implement this interface for GNU Radio by creating custom out-of-tree (OOT) downlink sink and uplink source blocks that connect directly to the DPDK controller through the shared-memory regions described above. These blocks map the regions directly and link no DPDK libraries, so applications need no DPDK integration to use the platform. The downlink sink block consumes IQ samples through the GNU Radio scheduler, accumulates them into slot-sized frames, and commits each completed frame to a circular buffer. A dedicated thread pinned to a user-selected CPU core drains this buffer and writes each frame into the shared-memory ring for the DPDK controller, which packetizes and forwards the data to the FPGA. Conversely, the uplink source block receives slot-sized frames from the DPDK controller, buffers them locally, and drains them over successive scheduler calls. This design allows baseband processing to evolve independently of the fronthaul transport logic.

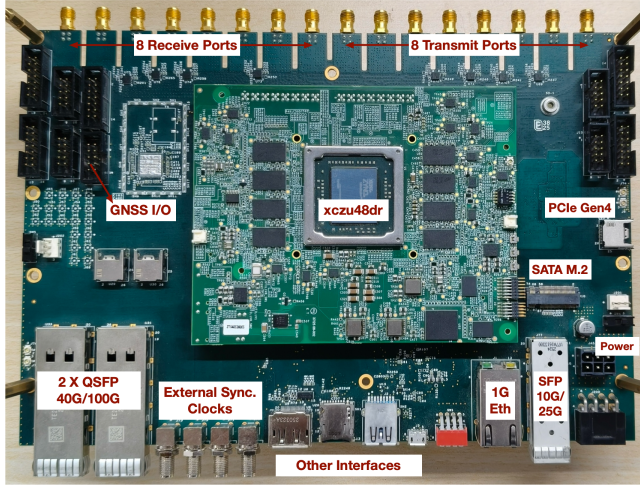


Figure 3: Picture of the actual hardware showing the SOM module in the center and the RF board containing all peripherals; the SOM Module and the RF Board are connected via FMC+ connectors.

3.8 Configurable Parameters

The system exposes a set of parameters that can be configured either statically at bitstream generation time or dynamically at runtime on a per-slot basis. Static parameters are baked into the FPGA fabric and define the radio’s fundamental operating constraints, such as slot duration. Dynamic parameters are delivered to the FPGA via the per-slot control packet from the Host, allowing the baseband software to reconfigure the radio slot by slot without interrupting the streaming pipeline.

Together, these mechanisms directly address three challenges introduced in Section 1. Static slot duration and clock-anchored slot tick generation ensure deterministic radio timing and immunity to host scheduling variability, satisfying the strict timing requirement. The streamlined three-packet fronthaul with jumbo-frame data payloads keeps protocol overhead minimal and sustains high link utilization, satisfying the simplicity and performance requirements. The dynamic parameters delivered per slot via the control packet allow updates to data direction, layer count, NCO frequency, and phase at the finest granularity the radio supports, satisfying the rapid dynamic control requirement without interrupting the streaming pipeline.

4 Implementation

So far, we have explained the system architecture of RadioStream. We have also implemented RadioStream on a real hardware platform. For the host side, we used two independent x86-based servers with two different NICs (Mellanox ConnectX-5 100 Gbps and Intel Ethernet E810-CQDA2 Network Adapter) to ensure repeatable platform performance. Similarly, on the radio side, we demonstrated the design’s

scalability by implementing the FPGA design on two different radio units.

We first used a standard commercially available RFSoc 4x2 [2] based on the Xilinx XCZU48DR SoC developed by RealDigital. The board provides four ADC channels at up to 5 Gsps and two DAC channels at up to 9.85 Gsps with direct RF sampling capability, a 100 Gbps QSFP28 interface, and 4 GB of PL-accessible DDR4 memory. To improve scalability and evaluate end-to-end performance, we implemented the system on a customized 8T8R (8 Transmitters and 8 Receivers) RFSoc with all required peripherals for SDR-based applications, as shown in Fig. 3. This system consists of two important components: 1) SOM (System on Module) containing an RFSoc, and 2) RF board containing all the peripherals and Ethernet interfaces for external connectivity.

The platform is built around a custom RFSoc SOM based on the Xilinx XCZU48DR, integrating eight ADC and eight DAC channels, each conditioned through a VGA stage with a maximum input level of 1 dBm. PL-side DDR4 at 1333 MHz, accessed via a MIG controller, which serves as the slot-indexed IQ buffer for downlink prefetch and uplink capture. Two 100 Gbps Ethernet interfaces are exposed via the ARF connector, and we use one. A Zero Delay Multiplier (ZDM) clock circuit on the SOM manages timing, accepts a GPS or PTP reference, and drives all ADC and DAC tiles, eliminating inter-tile skew.

The SOM mounts onto a custom RF board via an FMC+ connector. The board breaks out all interfaces required for the target application: dual QSFP28 cages for the two 100 Gbps ports, SMA connectors for all eight TX and eight RX RF chains, a GPS input for the ZDM reference, a PCIe x8 edge connector from the FMC+ GTY lanes, a 1 Gbps management Ethernet port, and a USB bridge for JTAG and UART access. In this work, the active interfaces are the primary QSFP28 port and all RF SMA connectors. The remaining peripherals are present on the board but unused in this implementation.

5 Evaluation

We evaluate RadioStream at two levels. We first use microbenchmarks to quantify how well the platform satisfies the three challenges from Section 1: high-speed, low-latency data transfer and precise slot-level timing between the host and the radio, and rapid slot-scoped control. We then show the ease-of-use of the platform by studying three representative application scenarios and how these properties translate into practical wireless experiments across both mmWave and sub-6 GHz front-ends (Fig. 4.)

5.1 Microbenchmarks

Our microbenchmarks focus on four questions. First, how accurately does the host observe the FPGA-generated notion of time? Second, how quickly can the platform support

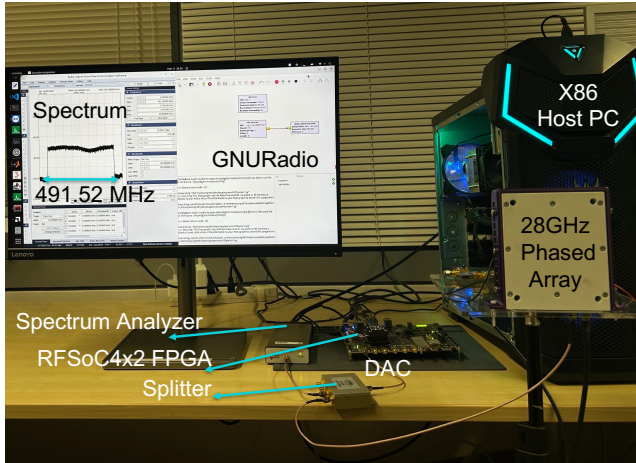


Figure 4: Experimental setup for the mmWave transmission test. GNU Radio running on the x86 host enables a single DAC channel at 491.52 MHz bandwidth on the RFSoc 4x2 platform. The DAC output is split using an RF splitter to feed the 28 GHz phased-array front-end and a Signal Hound spectrum analyzer, where the transmitted spectrum is monitored.

low-latency SDR/system control, as captured by a representative NCO frequency update experiment on the same control path used for slot-scoped reconfiguration? Third, can the host software path keep up with the fronthaul link without becoming the bottleneck? Fourth, where does the current implementation saturate as the aggregate RF bandwidth increases?

Throughput versus aggregated RF bandwidth: To showcase the high-throughput streaming capabilities of RadioStream, we sweep the aggregate RF bandwidth and the number of channels and report both sustained throughput and data transfer latency between the FPGA and the host controller, as well as the Host Controller to the Application environment/GNURadio. This sweep identifies the point at which the current RFSoc4x2 implementation stops being limited by the 100 Gbps Ethernet link and instead becomes constrained by the memory path and buffering architecture. Due to the symmetry in the system, the results will remain the same irrespective of the UL/DL slot configuration. The evaluation setup is shown in Fig. 4. Across all measured configurations described in Table 2, a maximum channel bandwidth of 2 GHz is achieved (4 channels with 491.52Mps sampling rate), and the optical link sustains between 84 and 91 Gbps averaged over a 20-minute test, with no packet drops observed during an extended 5-hour experiment, demonstrating that the DPDK-based transport path delivers reliable high-throughput data transfer across the full tested operating range between the host and the FPGA.

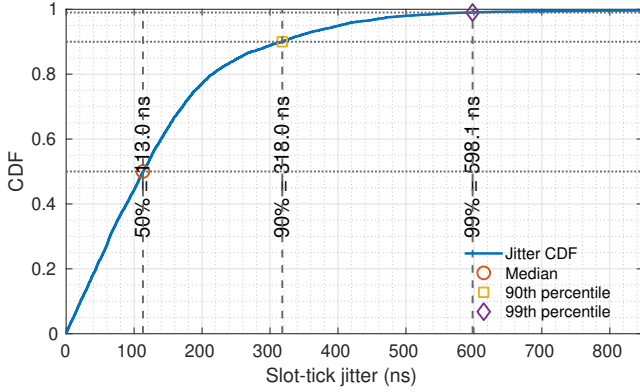
Uplink reliability versus a commercial baseline: We compare the uplink streaming reliability of RadioStream against a widely used commercial RFSoc SDR, the USRP X410

driven by UHD 4.8, by running the *same* GNU Radio application on both platforms and counting dropped packets from the application logs over one-hour runs. To isolate the transport path from any downstream signal-processing load, we run a *dry* GNU Radio flowgraph on each platform in which the received samples are discarded immediately after the source block, so the measurement reflects only the ability of the host stack to ingest the uplink stream at line rate. Both platforms carry sc16 samples, the 16-bit complex format the radios produce natively at four bytes each, over a 100 Gbps optical Ethernet link into a host running a DPDK-based transport with hugepage-backed buffers. The two stacks are further matched at the NIC, with a 9000-byte Jumbo MTU and a single receive queue on each. To keep each host path deterministic, we isolate the streaming cores from both the scheduler and device interrupts while pinning them to a fixed 5.0 GHz. RadioStream runs on only *three* such cores, whereas we provision the X410 baseline on a more capable server with additional cores and a dedicated core for its DPDK poll thread alone, deliberately favoring the baseline. We report the aggregate uplink rate each platform sustains, reaching each target rate by configuring active-layer count and per-layer bandwidth.

We begin each measurement after a 10-second warm-up to allow the transport to reach a steady state. Both platforms are lossless through 491.52 Mps. At 983.04 Mps, i.e., two layers of 491.52 Mps, the X410 nearly keeps pace, losing about 166 million of the 3.54 trillion expected samples over the hour, a 0.0047% loss, to two brief host-side interruptions, while RadioStream loses none. At 1966.08 Mps, i.e., four layers of 491.52 Mps, the X410 no longer keeps up. Over the hour, it delivers none of the roughly 7.08 trillion expected samples to the application, a 99% loss, while RadioStream sustains the rate. The NIC reports no missed, errored, or buffer-starved frames on either configuration for the full hour, so nothing is lost in transit. The shortfall lies in the host-side receive path, where a stall backs up the radio's FIFO and forces a restart, and the samples not produced during the restart are counted as lost. At four layers, these stalls become near-continuous, cycling between overrun and restart, and the radio is stopped for most of the hour. The X410 cannot sustain this rate with a single QSFP28 link even on the more capable server, while RadioStream does so on only three cores. This suggests that the constraint is the host-side, software-timed transport, not raw host compute. The same behavior appears under UHD's own native benchmark_rate utility, with GNU Radio removed from the path. Because uplink samples that arrive late or are dropped cannot be recovered in a real-time pipeline, the missing samples are unrecoverable gaps in the received stream, which RadioStream avoids by anchoring streaming to the FPGA (Section 2).

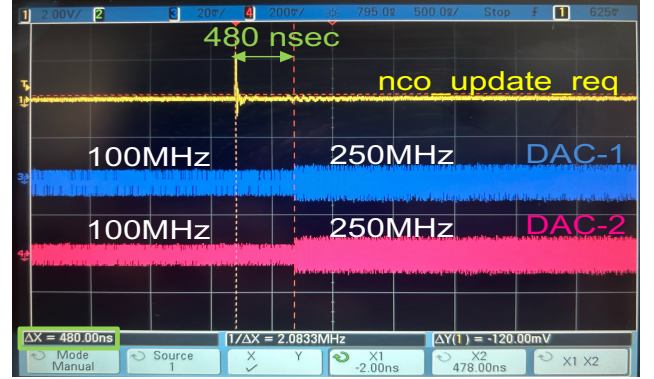
Table 2: Uplink and Downlink data transport duration (Transfer Time (μ s)) between the FPGA and the Host Controller, and the instantaneous throughput (Inst. Thr (Gbps)) for the data transfer averaged across a 1-hour test

BW	Path	1 Channel		2 Channels		4 Channels		8 Channels	
		Time	Inst. Thr	Time	Inst. Thr	Time	Inst. Thr	Time	Inst. Thr
122.88 MHz	FPGA $\leftrightarrow$ Host_Ctrl	19.98	98.39	39.82	98.76	79.60	98.80	159.30	98.74
	Host_Ctrl $\leftrightarrow$ App	15.11	130.12	30.09	130.70	59.88	131.32	120.17	130.89
245.76 MHz	FPGA $\leftrightarrow$ Host_Ctrl	39.81	98.77	79.59	98.81	159.28	98.75	346.60	90.76
	Host_Ctrl $\leftrightarrow$ App	29.97	131.20	59.81	131.48	120.42	130.61	294.06	106.98
491.52 MHz	FPGA $\leftrightarrow$ Host_Ctrl	79.59	98.81	159.29	98.74	345.67	91.00	–	–
	Host_Ctrl $\leftrightarrow$ App	59.88	131.35	120.19	130.86	295.75	106.36	–	–

**Figure 5: CDF of slot-tick timing jitter received at the host controller via the custom DPDK framework. The median, 90th percentile, and 99th percentile jitter are approximately 113 ns, 318 ns, and 598 ns, respectively, demonstrating sub-microsecond timing variability in slot scheduling.**

Slot-tick jitter and timing stability: The FPGA emits a 64-byte slot-tick packet once per slot, and the host uses this packet as its reference for deciding whether a slot should be used for uplink reception, downlink transmission, or idle processing. We timestamp these packets at the host using the CPU clock and measure the inter-arrival jitter relative to the expected 0.5 ms slot period. Fig. 5 shows the CDF of the absolute slot-tick jitter measured in nanoseconds. The results demonstrate that the timing signal is highly stable: the median jitter remains below 115 ns, while the 90th and 99th percentile jitter are bounded by 320 ns and 600 ns, respectively. These results confirm that the packetized slot-tick mechanism provides a sufficiently precise timing reference for coordinating host-side processing with the FPGA timeline. This experiment directly validates the central architectural claim that the FPGA remains the timing master while the host follows that timeline through explicit packetized timing cues rather than through cycle-synchronous execution.

Low-latency SDR/system control: To quantify the responsiveness of the control plane more generally, we include a microbenchmark for low-latency SDR/system control and instantiate it using an NCO frequency update measurement. This choice is representative because the same slot-scoped

**Figure 6: Measured NCO frequency switching latency. The oscilloscope capture shows approximately 480 ns across both DAC channels, confirming that the platform can support fine-grained, slot-level control of radio parameters.**

control path also carries other runtime updates, such as duplex mode selection, layer configuration, antenna mapping, NCO phase, and other per-slot radio parameters. In this experiment, the host triggers an RFDC reconfiguration command. The trigger is routed to a GPIO for visibility, while the RFDC busy signal and DAC output are observed using the on-board ILA and an oscilloscope, as shown in Fig. 6. The oscilloscope captures the time between the update request and the observable change in the DAC output spectrum, which corresponds to approximately 480 ns. To better understand the internal latency components, we also inspect the RFDC control path using the on-chip ILA, which shows that the RFDC update operation itself completes in 36 clock cycles at the 100 MHz control clock, corresponding to 360 ns. The remaining latency is therefore attributable to the DAC pipeline and analog output delay (Δ_{DAC}). It is worth noting that for sweep-driven applications such as spectrum sensing, the look-ahead latency corresponds to a one-time pipeline fill rather than a recurring per-step cost. Overall, these results demonstrate that control actions propagate through the system and manifest at the RF output in well under 1 μ s, confirming that the platform can support fine-grained, slot-level control of radio parameters. This experiment is particularly important because dynamic per-slot control is one of the central architectural claims of the platform.

Hardware Cost and Modularity: Every module in the Vivado pipeline is written in Verilog and exposes an AXI-Stream interface, allowing new DSP blocks such as filters to be incorporated into the datapath with minimal effort. The latency of each added module must be compensated in the gating logic to preserve over-the-air (OTA) synchronization between the transmit and receive paths. On the target Zynq UltraScale+ RFSoc, the design remains lightweight, leaving **over 84% of LUTs and 99% of DSPs free** for additional blocks. Programmable logic consumes accounts for only ≈ 3.5 W of the **14.23 W** total on-chip power, with the remaining ≈ 10.7 W coming from fixed RFSoc platform overhead such as the RF data converters, GTY transceivers, and processing system.

5.2 System-level Applications

We use three application scenarios to demonstrate the capabilities of RadioStream beyond the microbenchmarks above. These experiments are not intended to claim new communication or localization algorithms. Instead, they show that the proposed transport and control architecture supports continuous wideband communication, directional beam management, and multi-antenna sub-6 GHz processing with minimal application-specific glue logic.

Point-to-point wideband communication: We first evaluate the platform in a two-node point-to-point link using phased-array front-ends operating at 28 GHz. One node continuously transmits a 5G NR OFDM waveform (similar to Fig. 4) with an EIRP of about 5 dBm, where the on-chip NCO synthesizes the intermediate frequency. The transmitted waveform employs 256-QAM modulation and follows the standard 5G slot structure consisting of 14 OFDM symbols per slot. Among these symbols, three carry DMRS pilots occupying every other subcarrier to enable channel estimation at the receiver. The system uses 273 physical resource blocks, corresponding to $273 \times 12 = 3276$ subcarriers. The receiving node captures the downconverted IQ samples and streams them to the host for baseband processing. For this experiment, we record approximately 5–6 ms of continuous data, corresponding to roughly 10 slot periods with a slot duration of 0.5 ms and a 30 kHz subcarrier spacing. As shown in Fig. 7, the packet-detection output reveals slot boundaries occurring at the expected 0.5 ms periodicity, indicating that the platform reliably delivers complete slot payloads without burst losses or packet drops. At the receiver, an OFDM equalizer is applied using the DMRS pilots, and the resulting EVM measurements as shown in Fig. 13 remain stable across both slot index and resource block, with values in the range of 4–6%. These results confirm that the received sample stream is continuous and that the platform’s timing architecture does not introduce slot-to-slot discontinuities. We note that these capture durations correspond to the record lengths

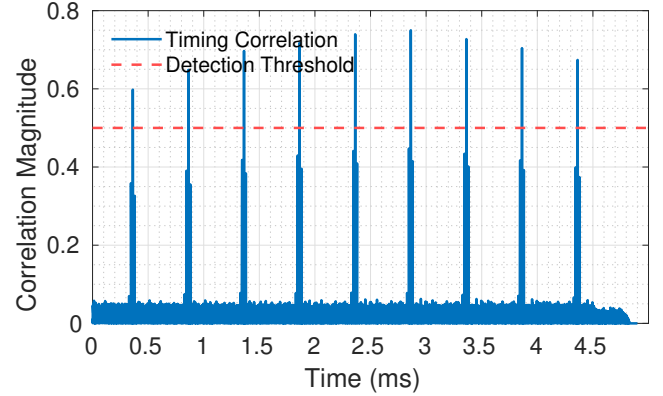


Figure 7: Packet-detection output for a continuous 5G OFDM capture over the 28 GHz phased-array link. The detections at every 0.5 ms interval confirm slot-aligned continuous streaming. chosen for offline analysis rather than any limitation of the platform’s continuous streaming capability.

To complement the 5G result with a more communication-centric view, we also run a WiFi waveform whose packet structure includes standard STF/LTF-based packet detection, channel estimation, and CFO/SFO compensation. For this test, the waveform uses 1024 subcarriers, with 480 KHz subcarrier spacing (i.e., 491.52 MHz instantaneous bandwidth) at a 4 GHz center frequency using the on-chip NCO. Fig. 8 shows the received constellations for 64-QAM and 256-QAM. The distinct clusters across both modulation orders demonstrate that the platform preserves sample integrity well enough to support conventional wideband communication workloads, and that the communication story in this paper is not limited to a single sensing-style workload.

28GHz mmWave beam sweeping: We next evaluate a directional communication procedure that exercises both continuous streaming and phased-array control. The transmitter is placed at three azimuth locations, 0° , $+30^\circ$, and -30° , while the receiver sweeps its beam across the azimuth plane from -60° to 60° in 5° increments (Fig. 9.a). The receiver uses the same 28 GHz phased-array front end, with all receiver elements enabled at 491.52 MHz bandwidth, and Fig. 9.b shows the resulting SNR profile. The peaks in the SNR curve align with the known transmitter directions, while the side lobes and nulls are also captured as expected. The beam selection results show about a 2.5° peak-angle error, with negligible 0.5 dB SNR degradation. This experiment shows that RadioStream supports both high-speed IQ transport and the complete data path required for practical directional communication procedures such as beam sweeping and beam selection.

Sub-6 GHz AoA and range estimation: Our third scenario uses the platform as a coherent wideband capture engine for sub-6 GHz localization. We operate at 4 GHz and localize the two sources from wideband channel measurements collected

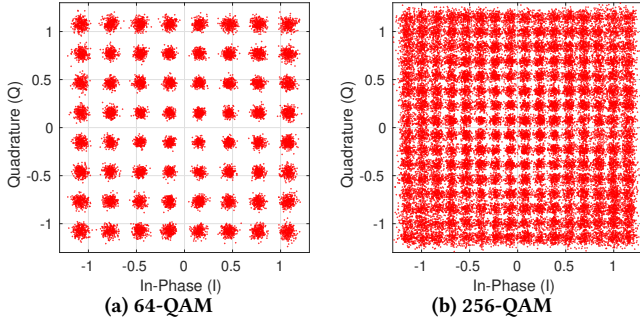


Figure 8: Received WiFi constellations for 64-QAM and 256-QAM demonstrating clean wideband communication bring-up.

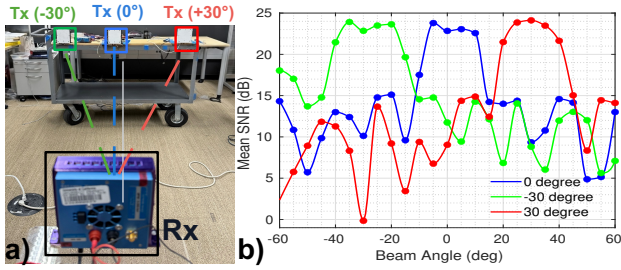


Figure 9: mmWave beam sweeping: a) the setup shows the location of three transmitters with respect to the receiver phased-array antenna, b) demonstrates the platform capability for accurate beam selection for three different angles.

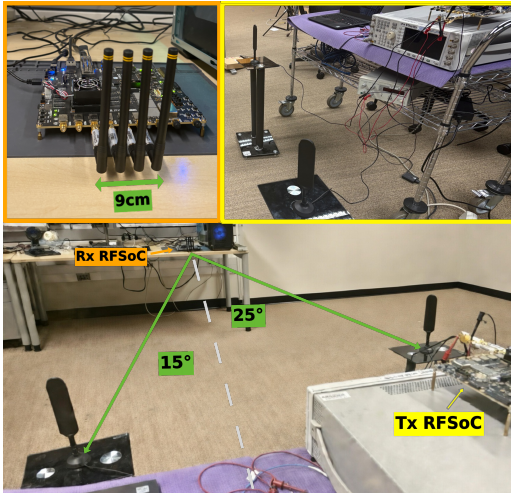


Figure 10: Experimental setup for the Sub-6 GHz AoA and range estimation test. Each receive antenna is spaced 3cm apart, and two transmit antennas are at 15° and -25° from the orthogonal line of the receive array.

across an antenna array whose element spacing is chosen to avoid the grating-lobe issue ($3\text{ cm} < \frac{\lambda}{2}$) as shown in Fig. 10. In this experiment, we vary the number of antennas and the signal bandwidth and observe how this improves angular and range resolution. This was not feasible with traditional SDR

architectures due to limited bandwidth and antenna count, as well as phase incoherence between ADCs. Before diving into the results, we tested the synchronization enabled by the RFSoc multi-tile synchronization (MTS) feature through a wired test. As shown in Fig. 14, The results demonstrate that the ADC channels remain tightly phase-aligned across the frequency sweep, confirming the effectiveness of the multi-tile synchronization mechanism.

Fig. 11 shows the resulting profiles. This experiment does not introduce a new AoA algorithm, and is instead meant to demonstrate the platform preserving phase-consistent, wide-band, multi-antenna measurements suitable for localization and other inference workloads using the same streaming architecture. As shown, the two-antenna configuration with 30.72 or 122.88 MHz bandwidth, representative of traditional SDR platforms, cannot clearly distinguish the two sources. Increasing the number of antennas to four improves the angular resolution, allowing Fig. 11(c) to separate the sources in angle. This improvement is reflected by the narrower angular spread for the larger array ($+20^\circ$ for a 4-element array compared with $+40^\circ$ for a 2-element array). Finally, Fig. 11(d) shows that increasing the bandwidth to 491.52 MHz, combined with multiple antennas, yields a much sharper profile in which the two sources are clearly distinguishable and validates the distance-resolution improvement from 2.4 m for a 122.88 MHz signal to 0.6 m for 491.52 MHz.

Taken together, these three examples show that RadioStream supports continuous wideband communication, dynamic fine-grained control over parameters like NCO frequency and phase, directional phased-array operation, and coherent multi-antenna sensing workloads using the same underlying timing, transport, and control substrate. The microbenchmarks above quantify the platform properties that make these experiments possible, while the application results here demonstrate what those properties enable in practice.

6 Related Work

Experimental wireless platforms have long traded flexibility against the data rates and timing fidelity the hardware could sustain. Early software radio showed commodity processors could run real-time PHY: Sora ran full 802.11a/b/g on multi-core CPUs via a custom PCIe radio bridge [51], and GNU Radio with USRP front-ends made rapid prototyping widely accessible [12]. Argos distributes beamforming across large arrays [46], LuMaMi performs real-time 100-antenna processing in a dedicated chassis [30, 37], and shared testbeds [8, 10, 13, 15, 25, 40] lowered the barrier to at-scale experimentation, but each fixes a particular radio front-end and transport fabric. While the RFSoc removed the converter bottleneck by integrating high-speed ADCs and DACs directly with the FPGA, it shifted the challenge to

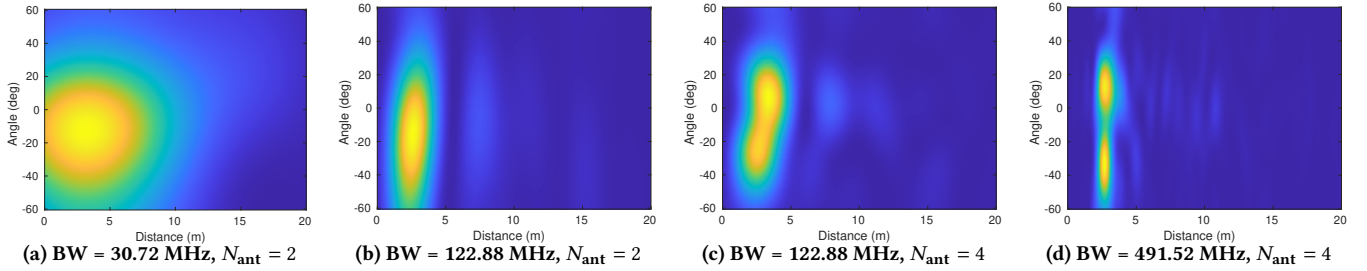


Figure 11: Joint AoA/range profile from a sub-6 GHz wideband capture. This experiment highlights that the platform can deliver phase-consistent multi-antenna data to higher-layer sensing and inference pipelines.

wideband streaming between the radio and a flexible host. As detailed in Section 2, at these rates the transport interface between the deterministic radio and the non-deterministic host becomes a first-order design problem, which is the focus of our work.

On the baseband side, a complementary line of work shows how far software-driven experimentation can be pushed once samples reach the host. Agora shows that optimized software can support real-time massive-MIMO processing on commodity servers [24], and RENEW argues more broadly that wireless research needs infrastructure that combines software flexibility with deployable hardware [25]. But these systems largely assume that high-rate IQ already arrives reliably. Recent mmWave and beamforming prototypes similarly demonstrate the benefits of tighter hardware–software co-design, enabling advanced beamforming techniques difficult to realize with generic SDR stacks [22, 26, 33], but each targets a specific signal-processing pipeline or array architecture.

In contrast, we focus on the architecture: a streaming interface that must deliver samples across heterogeneous timing domains without sacrificing utilization or fine-grained control. RadioStream complements these efforts by providing the missing middle layer: a slot-aware, FPGA-anchored streaming substrate that connects high-rate RF hardware to flexible host computation while preserving deterministic timing, high transport efficiency, and fine-grained control.

7 Limitations and Future Work

Scalability to higher channels: RadioStream currently operates over a single point-to-point 100 Gbps fronthaul link. Scaling to 8-channel full-duplex operation at 491.52 MHz per channel pushes the aggregate IQ data rate to approximately 126 Gbps, exceeding a single link. For 5G-like applications requiring the full 0.5 ms slot duration, you can address this by activating the second QSFP port already routed on the SOM and EVK hardware. For applications that permit a shorter slot duration, the slot buffers can instead be realized entirely in on-chip RAM, eliminating the external DDR path.

Full-stack framework integration: RadioStream’s implementation targets a single RFSoc platform with a fixed 500 μ s slot cadence and a DPDK-based host data path. A

natural next step is integration with software RAN frameworks such as srsRAN or OpenAirInterface, which is feasible through a thin adapter layer with no changes to the FPGA design or the DPDK fast path. Exposing the DPDK host program through each framework’s existing radio abstraction, the radio session and baseband gateway interface in srsRAN, or the equivalent device abstraction in OpenAirInterface, makes the fronthaul appear as an ordinary radio device, so the modem can drive it as it would any other radio. Staging buffers in the adapter decouple the fixed FPGA slot cadence from the modem’s variable buffer sizes and de-interleave the layer-multiplexed slot payload into the per-antenna buffers both frameworks expect.

Multi-node synchronization: All current experiments operate over a single RFSoc–host link. The slot-synchronous contract extends naturally to multi-node deployments through a shared GPS or PTP reference, but their synchronization remains an open direction for future work.

8 Conclusion

RadioStream addresses three fundamental challenges in wideband multi-antenna SDR platforms: i) for host–radio time synchronization, it anchors slot timing in the FPGA and exports slot boundaries to the host via explicit slot-tick Ethernet packets, avoiding the shared-clock issues of host-centric stacks, with slot-tick jitter below 600 ns at the 99th percentile; ii) for wideband streaming, a streamlined three-packet Ethernet fronthaul sustains 90 Gbps across all tested channel counts and bandwidths with no packet drops; and iii) for fine-grained radio control, a two-slot look-ahead delivers per-slot updates to NCO frequency, phase, data direction, and antenna configuration, with control reaching the RF output in under 500 ns. Evaluated across communication and sensing workloads with GNU Radio integration, RadioStream shows that all three properties hold in practice, enabling a platform for next-generation wireless applications.

Acknowledgment

This work was supported in part by the U.S. National Science Foundation (NSF) under Award Nos. 2213689 and 2232481. The authors also thank the anonymous reviewers.

References

- [1] [n.d.]. Avnet. <https://www.avnet.com/americas/products/avnet-boards/applications/rfsoc-and-mmwave>.
- [2] [n.d.]. RFSoc 4x2. <https://www.realdigital.org/hardware/rfsoc-4x2>.
- [3] [n.d.]. X410 SDR. <https://www.ettus.com/all-products/usrp-x410/>.
- [4] [n.d.]. X440 SDR. <https://www.ettus.com/all-products/usrp-x440/>.
- [5] n.d.. RFSoc-MTS: Multi-Tile Synchronization Reference Design. <https://github.com/Xilinx/RFSoc-MTS>. Accessed: 2026-09-30.
- [6] Advanced Micro Devices, Inc. 2023. *Zynq UltraScale+ RFSoc Data Sheet: Overview (DS889)*. AMD (formerly Xilinx, Inc.). <https://docs.amd.com/v1.14>.
- [7] Ian F Akyildiz, Ahan Kak, and Shuai Nie. 2020. 6G and beyond: The future of wireless communications systems. *IEEE access* 8 (2020), 133995–134030.
- [8] Kiarash Amiri, Yang Sun, Patrick Murphy, Chris Hunter, Joseph R. Cavallaro, and Ashutosh Sabharwal. 2007. WARP, a Unified Wireless Network Testbed for Education and Research. In *Proc. IEEE Int. Conf. on Microelectronic Systems Education (MSE)*.
- [9] Abdulaziz Bazammul, Kasra Karkheiran, Belal Jahannia, Benjamin Hayden, Abdollah Amirany, Chandraman Patil, Elham Heidari, and Hamed Dalir. 2025. High-speed DAC/ADC implementation using RFSoc FPGA for quantum computing. In *Photonic computing: from materials and devices to systems and applications II*, Vol. 13581. SPIE, 42–50.
- [10] Lorenzo Bertizzolo, Leonardo Bonati, Emrehan Demirors, Amani Al-Shawabka, Salvatore D'Oro, Francesco Restuccia, and Tommaso Melodia. 2020. Arena: A 64-antenna SDR-based Ceiling Grid Testbed for Sub-6 GHz Radio Spectrum Research. *Computer Networks* 181 (2020).
- [11] Eric Blossom. 2004. GNU radio: tools for exploring the radio frequency spectrum. *Linux journal* 2004, 122 (2004), 4.
- [12] Eric Blossom. 2004. GNU Radio: Tools for Exploring the Radio Frequency Spectrum. *Linux Journal* 2004, 122.
- [13] Leonardo Bonati, Salvatore D'Oro, Lorenzo Bertizzolo, Emrehan Demirors, Zhangyu Guan, Stefano Basagni, and Tommaso Melodia. 2021. Colosseum: Large-scale Wireless Experimentation Through Hardware-in-the-loop Network Emulation. (2021).
- [14] Martin Braun, Jonathon Pendulum, and Matt Ettus. 2016. RFNoC: RF Network-on-Chip. In *Proceedings of the GNU Radio Conference*.
- [15] Joe Breen, Andrew Buffmire, Jonathon Duerig, Kevin Dutt, Eric Eide, Mike Hibler, David Johnson, Sneha Kumar Kasera, Earl Lewis, Dustin Maas, et al. 2020. POWDER: Platform for Open Wireless Data-driven Experimental Research. In *Proc. ACM Int. Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH)*.
- [16] Lewis J Brown, Louise H Crockett, and Robert W Stewart. 2025. A single-chip Split-6 PHY implementation for 5G NR, using RFSoc and PYNQ. *Franklin Open* (2025), 100408.
- [17] Rodolfo Carobene, Alessandro Candido, Javier Serrano, Alvaro Orgaz-Fuertes, Andrea Giachero, and Stefano Carrazza. 2025. Qibosoq: an open-source framework for quantum circuit RFSoc programming. *Quantum Science and Technology* 10, 3 (2025), 035010.
- [18] Tingjun Chen, Prasanthi Maddala, Panagiotis Skrimponis, Jakub Kolodziejski, Xiaoxiong Gu, Arun Paidimarri, Sundeep Rangan, Gil Zussman, and Ivan Seskar. 2022. Programmable and open-access millimeter-wave radios in the PAWR COSMOS testbed. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*. 1–8.
- [19] Wei Cheng, Zhihui Gao, and Tingjun Chen. 2024. Real-time Wideband Software-defined Radio with Python Programmability based on RFSoc. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1772–1774.
- [20] Wei Cheng, Zhihui Gao, and Tingjun Chen. 2024. SPEAR: Software-defined Python-enhanced RFSoc for wideband radio applications. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1834–1841.
- [21] Wei Cheng, Zhihui Gao, Jose Guajardo, Hesham Beshary, Ali M Niknejad, and Tingjun Chen. 2025. SPEAR+: Streaming-Based Multi-Channel SDR Implementation Using the RFSoc Platform. In *MILCOM 2025-2025 IEEE Military Communications Conference (MILCOM)*. IEEE, 317–322.
- [22] MinKeun Chung, Liang Liu, Andreas Johansson, Martin Nilsson, Olof Zander, Zhinong Ying, Fredrik Tufvesson, and Ove Edfors. 2020. Millimeter-wave massive MIMO testbed with hybrid beamforming. In *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 309–313.
- [23] James Craig, Louise H Crockett, Robert W Stewart, Ian Bowyer, and Garrey Rice. 2025. CP-OFDM PUSCH model-based design for 5G new radio transmitter on ZCU216 RFSoc. In *2025 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–6.
- [24] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. 2020. Agora: Real-time massive MIMO baseband processing in software. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 232–244.
- [25] Rahman Doost-Mohammady, Oscar Bejarano, and Ashutosh Sabharwal. 2021. Good times for wireless research. *Computer Networks* 188 (2021), 107870. <https://doi.org/10.1016/j.comnet.2021.107870>
- [26] Sourjya Dutta, C Nicolas Barati, David Ramirez, Aditya Dhananjay, James F Buckwalter, and Sundeep Rangan. 2019. A case for digital beamforming at mmWave. *IEEE Transactions on Wireless Communications* 19, 2 (2019), 756–770.
- [27] Maximilian Engelhardt, Sebastian Giehl, Michael Schubert, Alexander Ihlow, Christian Schneider, Alexander Ebert, Markus Landmann, Giovanni Del Galdo, and Carsten Andrich. 2024. Accelerating innovation in 6G research: real-time capable SDR system architecture for rapid prototyping. *IEEE Access* 12 (2024), 118718–118732.
- [28] Richard Gebauer, Nick Karcher, Mehmed Gueler, and Oliver Sander. 2023. Qicells: A modular rfsoc-based approach to interface superconducting quantum bits. *ACM transactions on reconfigurable technology and systems* 16, 2 (2023), 1–23.
- [29] Agrim Gupta, Sajjad Nassirpour, Manideep Dunna, Eamon Patamasing, Alireza Vahid, and Dinesh Bharadia. 2023. Greenmo: Enabling virtualized, sustainable massive mimo with a single rf chain. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–17.
- [30] Adel Heidari, Agrim Gupta, Ish Kumar Jain, and Dinesh Bharadia. 2025. PhaseMO: A Universal Massive MIMO Architecture for Sustainable NextG. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*. IEEE, 1–10.
- [31] Nafis Irtija, Jim Plusquellic, Eirini Eleni Tsiropoulou, Joshua Goldberg, Daniel Lobser, and Daniel Stick. 2023. Design and analysis of digital communication within an SoC-based control system for trapped-ion quantum computing. *IEEE Transactions on Quantum Engineering* 4 (2023), 1–24.
- [32] Sessa Sai Rakesh Jonnavithula, Ish Kumar Jain, and Dinesh Bharadia. 2024. Beamarmor5g: Demonstrating mimo anti-jamming and localization with srsran 5g stack. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1817–1819.
- [33] Thomas Kühne, Piotr Gawłowicz, Anatolij Zubov, Falko Dressler, and Giuseppe Caire. 2020. Bringing hybrid analog-digital beamforming to commercial MU-MIMO wifi networks. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–3.
- [34] Jesus O Lacruz and Rafael Ruiz. 2022. MIMORPH: A general-purpose experimentation platform for sub-6 GHz and mmWave frequency bands. *GetMobile: Mobile Computing and Communications* 26, 1 (2022), 5–8.

- [35] Jaewon Lee, Seoyoung Jang, Yujin Choi, Donggeon Kim, Matthias Braendli, Thomas Morf, Marcel Kossel, Pier-Andrea Francese, and Gain Kim. 2025. A 2-lane DAC-/ADC-based 2×2 MIMO PAM-4 MMSE-DFE wireline transceiver with FEXT cancellation on RFSoc platform. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2025).
- [36] Wei Lei, Yue Zhang, Hanfeng Xie, Zebin Chen, Zengping Chen, and Weixing Li. 2024. A fully digital transmitting-receiving platform for MIMO radar waveform diversity experiment. *IEICE Transactions on Information and Systems* (2024), 2023EDP7276.
- [37] Steffen Malkowsky, Joao Vieira, Liang Liu, Paul Harris, Karl Nieman, Nikhil Kundargi, Ian C. Wong, Fredrik Tufvesson, Viktor Öwall, and Ove Edfors. 2017. The World's First Real-Time Testbed for Massive MIMO: Design, Implementation, and Validation. *IEEE Access* 5 (2017).
- [38] Filip Michalak, Wojciech Zabolotny, Łukasz Podkalicki, Mateusz Malanowski, Marcin Piasecki, and Krzysztof Kulpa. 2022. Universal RFSoc-based signal recorder for radar applications. In *2022 23rd International Radar Symposium (IRS)*. IEEE, 136–140.
- [39] O-RAN Alliance. 2020. O-RAN Architecture Description. White Paper.
- [40] Dipankar Raychaudhuri, Ivan Seskar, Max Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. 2005. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-generation Wireless Network Protocols. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*.
- [41] Mikael Ade Krisna Respati and Byung Moo Lee. 2024. A survey on machine learning enhanced integrated sensing and communication systems: Architectures, algorithms, and applications. *IEEE Access* (2024).
- [42] Rafael Ruiz, Jesus O Lacruz, Bastian Bloessl, Matthias Hollick, and Joerg Widmer. 2025. HELIX: High-speed real-time experimentation platform for 6G wireless networks. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services*. 305–318.
- [43] Riya Sachdeva, Aakanksha Tewari, Sumit J Darak, Shobha Sundar Ram, and Sanat K Biswas. 2026. RFSoc-Based Integrated Navigation and Sensing Using NavIC. *arXiv preprint arXiv:2602.08596* (2026).
- [44] Alphan Şahin, Mihail L Sichiitiu, and İsmail Güvenç. 2023. A millimeter-wave software-defined radio for wireless experimentation. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–6.
- [45] Fatemeh Salehi, Nagarjun Bhat, Manideep Dunna, and Dinesh Bhara-dia. 2026. Rethinking Passive RIS at Sub-6 GHz: A Reality Check on Practical Gains. In *Proceedings of the Twenty-Seventh International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing* (Tokyo, Japan) (*MobiHoc '26*). Association for Computing Machinery. <https://doi.org/10.1145/3842721.3849692>
- [46] Clayton Shepard, Hang Yu, Narendra Anand, Erran Li, Thomas Marzetta, Richard Yang, and Lin Zhong. 2012. Argos: Practical Many-antenna Base Stations. In *Proc. ACM MobiCom*.
- [47] Marius Šiaučiusis, David Northcote, Josh Goldsmith, Louise H Crockett, and Šarūnas Kaladė. 2023. 100Gbit/s RF sample offload for RFSoc using GNU Radio and PYNQ. In *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 1–5.
- [48] Vaibhav Pratap Singh, Anil Prabhakar, et al. 2025. High Accuracy Timing Clock Synchronization on RFSoc Board. In *2025 National Conference on Communications (NCC)*. IEEE, 1–6.
- [49] George Sklivanitis, Adam Gannon, Stella N Batalama, and Dimitris A Pados. 2016. Addressing next-generation wireless challenges with commercial software-defined radio platforms. *IEEE Communications Magazine* 54, 1 (2016), 59–67.
- [50] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, et al. 2022. The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors. *Review of Scientific Instruments* 93, 4 (2022).
- [51] Kun Tan, Jiansong Zhang, Ji Fang, He Liu, Yusheng Ye, Shen Wang, Yongguang Zhang, Haitao Wu, Wei Wang, and Geoffrey M. Voelker. 2009. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In *Proc. USENIX NSDI*.
- [52] Harsh Tataria, Mansoor Shafi, Andreas F Molisch, Mischa Dohler, Henrik Sjöland, and Fredrik Tufvesson. 2021. 6G wireless systems: Vision, requirements, challenges, insights, and opportunities. *Proc. IEEE* 109, 7 (2021), 1166–1199.
- [53] Magnus Oksbøl Therkelsen, Marek Peca, and Dirk Thurnes. 2023. Characterisation of RFSoc Gen 3 data converters for satellite RF payloads. In *2023 European Data Handling & Data Processing Conference (EDHPC)*. IEEE, 1–7.
- [54] Yucong Wang, Ming Ouyang, Tengyu Zhang, and Feifei Gao. 2025. An RFSoc-Based Scalable OTFS Prototyping Platform for Integrated Sensing and Communications. *IEEE Transactions on Wireless Communications* (2025).
- [55] Dingzhu Wen, Yong Zhou, Xiaoyang Li, Yuanming Shi, Kaibin Huang, and Khaled B Letaief. 2024. A survey on integrated sensing, communication, and computation. *IEEE Communications Surveys & Tutorials* (2024).
- [56] Kai Zheng, Wuqiong Zhao, Timothy Woodford, Renjie Zhao, Xinyu Zhang, and Yingbo Hua. 2024. Enhancing mmwave radar sensing using a phased-mimo architecture. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 56–69.

A Appendix

A.1 Custom SDR Hardware Description

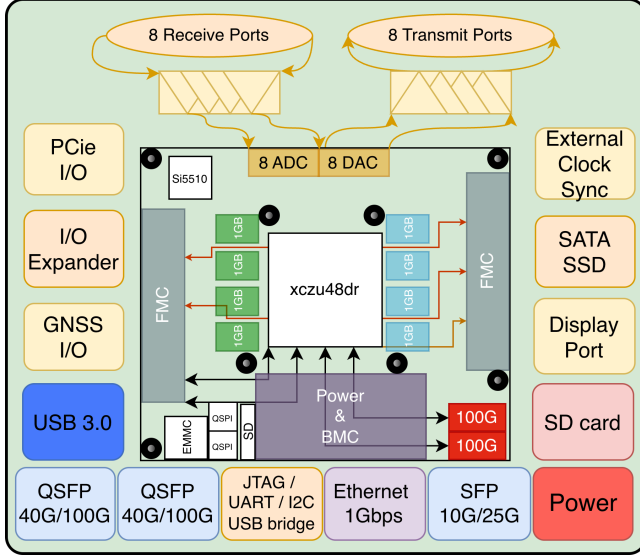


Figure 12: Block diagram schematic of SOM module in the center and the RF board containing all the peripherals; the SOM Module and the RF Board are connected via FMC+ connectors.

Fig. 12 shows the block diagram of the custom System on Module (SOM) used as the radio unit in RadioStream. The SOM is built around the Xilinx xczu48dr, a Zynq UltraScale+ RFSoC Gen 3 device that integrates eight RF-ADC channels operating at up to 5 Gbps and eight RF-DAC channels at up to 9.85 Gbps directly on the same die as the programmable logic fabric and ARM processing subsystem. This monolithic integration eliminates the external JESD204 interfaces required by conventional multi-chip radio front-ends, reducing board complexity and inter-component timing skew.

The SOM measures approximately 4×5 inches and exposes the full peripheral set of the xczu48dr through VITA 57.4-compliant FMC+ connectors, including high speed GTY transceivers for multi-gigabit serial links, PS-side MIO peripherals, and direct ADC/DAC signal access. On-module support circuitry includes 4 GB of PL-side DDR4 at 1333 MHz accessed via a MIG controller, 4 GB of PS-side DDR4 at 1200 MHz for the ARM subsystem, dual 256 MB QSPI flash, 8 GB eMMC storage, and a Zero Delay Multiplier (ZDM) clock distribution circuit that accepts a GPS or PTP reference and distributes phase-aligned clocks to all ADC and DAC tiles. In RadioStream, the ARM subsystem and PS-side memory are unused, with all baseband and transport logic executing entirely within the PL fabric.

The SOM is mounted on a custom RF board via the FMC+ connector, which routes all high-speed interfaces to application-specific peripherals: dual QSFP28 cages for two independent 100 Gbps Ethernet ports, SMA connectors for all eight TX and eight RX RF chains, a GPS synchronization input for the ZDM reference, a PCIe x8 edge connector from the FMC+ GTY lanes, a 1 Gbps management Ethernet port, and a USB bridge providing simultaneous JTAG and UART access. In this work, the active interfaces are the primary QSFP28 port, all RF SMA connectors, and the GPS input. The modular SOM-plus-carrier architecture separates RF-agnostic compute and memory resources from application-specific connectivity, allowing the same SOM to be redeployed on alternative carrier boards without FPGA redesign.

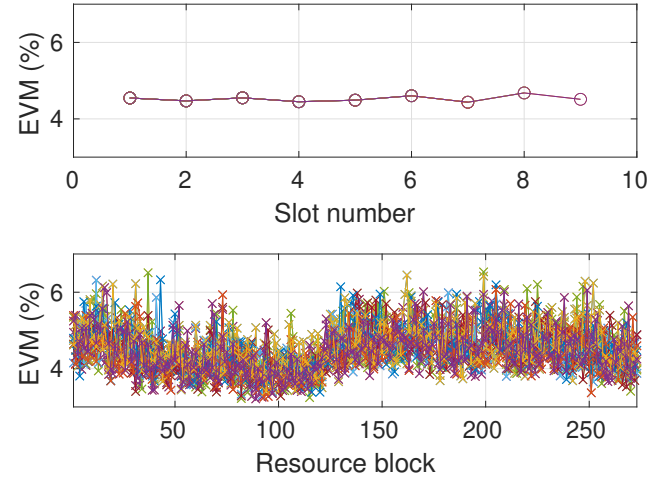


Figure 13: EVM across slot index and resource blocks for the 28 GHz point-to-point link. The stability of the EVM profile indicates that the platform does not introduce slot-scale discontinuities or missing samples.

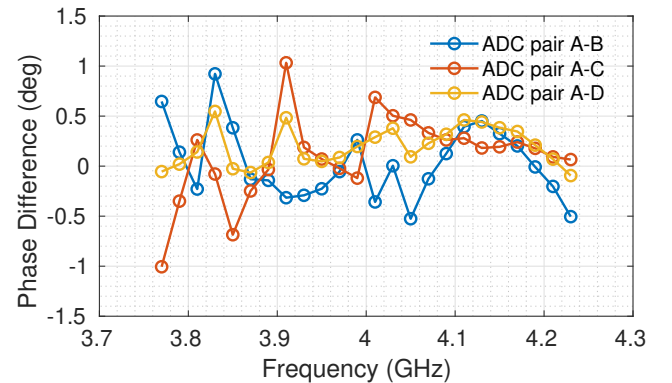


Figure 14: Phase coherence across ADC channels enabled by RFSoC multi-tile synchronization (MTS)