HELIX: High-speed Real-Time Experimentation Platform for 6G Wireless Networks

Rafael Ruiz
rafael.ruiz@imdea.org
IMDEA Networks Institute
Madrid, Spain

Jesus O. Lacruz
jesusomar.lacruz@imdea.org
IMDEA Networks Institute
Madrid, Spain

Bastian Bloessl
bbloessl@seemoo.tu-darmstadt.de
TU Darmstadt
Darmstadt, Germany

Matthias Hollick
mhollick@seemoo.tu-darmstadt.de
TU Darmstadt
Darmstadt, Germany

Joerg Widmer
joerg.widmer@imdea.org
IMDEA Networks Institute
Madrid, Spain

ABSTRACT

Mobile networks are evolving rapidly, with 6G promising unprecedented capabilities in terms of data rates and ultra-low latencies. However, the development of testbed platforms for wireless experimentation has not kept pace. Existing platforms typically offer either end-to-end capabilities with low bandwidth or high bandwidth with limited or no real-time functionality. In this paper, we introduce HELIX, an experimentation platform with 6G scalable real-time capabilities. HELIX integrates a comprehensive physical layer subsystem with multi-numerology support alongside an advanced mixed software-hardware control unit responsible for interacting with the fronthaul network and dynamically configuring the functional split in real time. On the server side, we implement the necessary drivers and routines to enable seamless integration with O-RAN systems, thus facilitating open and end-to-end experimentation. We demonstrate the capabilities of HELIX through a variety of experiments at sub-6 GHz, 28 GHz, and 60 GHz frequencies. Notably, HELIX achieves data rates of up to 1200 Mbps using 256-QAM modulation with over 417 MHz of bandwidth, and end-to-end bidirectional latencies of 500 μs. We show advanced features, including the implementation of Integrated Sensing And Communication (ISAC), and discuss how the platform could be extended to support bandwidths of up to 1670 MHz.

CCS CONCEPTS

• **Networks** → **Network experimentation**; **Mobile networks**; • **Computer systems organization** → **Real-time system architecture**; • **Hardware** → **Digital signal processing**.

KEYWORDS

6G, O-RAN, Wireless Experimentation, SDR, FPGA

MobiSys '25, June 23–27, 2025, Anaheim, CA, USA

ACM ISBN 979-8-4007-1453-5/2025/06
https://doi.org/10.1145/3711875.3729152

ACM Reference Format:
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 *The 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25), June 23–27, 2025, Anaheim, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3711875.3729152

1 INTRODUCTION

Mobile networks have continuously evolved to meet the requirements of modern applications such as autonomous systems, augmented reality, and interconnected vehicles [40]. These applications demand high data rates and low latencies, some even requiring end-to-end latencies below 1 ms (see [36] and references therein). As part of this evolution, 5G has been designed to address the limitations of previous mobile network generations. One observation is that sub-6 GHz frequency spectrum is heavily congested, necessitating the exploration of higher frequencies, such as Millimeter-Wave (mmWave). These frequencies offer vast bandwidth resources, enabling support for more users and the allocation of larger bandwidth. To capitalize on this, 5G introduces new numerologies that allow bandwidths exceding 400 MHz per spatial stream, combined with high modulation orders (up to 256-QAM) [1, 47]. Despite the advancements made by 5G, the demands for even higher data rates, lower latencies, and greater capacity are driving the development of 6G. 6G is expected to expand beyond the mmWave band into the THz frequency range, offering order-of-magnitude increases in bandwidth while achieving latencies as low as 0.1 ms [35].

The increased bandwidth of 5G and 6G also provides transformative opportunities, the most notable being the ability to use fine-grained channel measurements inherently required for transceiver equalization for the concept of Integrated Sensing And Communication (ISAC) [21, 37]. ISAC aims to add sensing functionality to communication systems at minimal cost. However, to achieve this vision, significant architectural changes are required. Specifically, channel measurements that are traditionally confined to the physical layer must be made accessible to the upper layers of the network stack, enabling advanced sensing functionalities [20, 42].

Experimentation is a cornerstone for validating the design of future communication standards. Commercial-Off-The-Shelf (COTS) devices offer standard-compliant behavior, but lack flexibility and access to critical parameters, whereas signal generators and analyzers offer the highest flexibility, but lack end-to-end system

support and hardware capabilities are very different from COTS devices. As a consequence, Software-Defined Radio (SDR)-based platforms have gained significant popularity thanks to providing direct access to IQ samples for a wide range of frequencies. Open-source initiatives like srsRAN [11] and OAI [15] have leveraged these platforms to implement full-stack 4G and 5G systems. While such approaches have been crucial in enabling wireless research, they are inherently limited in scalability. As 5G evolved to support bandwidths exceeding 400 MHz at mmWave frequencies, the limitations of software-based approaches become apparent. The fundamental bottleneck lies in the massive fronthaul capacity required to transfer IQ samples between the Radio Unit (RU) and the server running the rest of the stack. This issue becomes even more pronounced in 6G systems, which are expected to operate with significantly larger bandwidths. Furthermore, the increase in data rates amplifies the impact of software-induced latencies in critical components such as channel decoders, Orthogonal Frequency Division Multiplexing (OFDM) (de)modulators, and other physical layer processes, directly affecting system performance. While srsRAN [11] and OAI [15] can use hardware accelerators to offload some of the most computationally intensive tasks (e.g., channel decoding) and NVIDIA's Aerial framework [16] exploits the parallel processing capabilities of GPUs to implement the entire physical layer, these solutions still rely on the transport of full IQ samples between the RU and the server. Despite this, the maximum bandwidth supported by the above systems is 100 MHz.

The challenges faced by current testbeds highlight a critical gap: testbeds are not evolving at the same pace as the technologies themselves. To fill this gap, we introduce HELIX, a high-speed real-time experimentation platform for 6G wireless networks. HELIX is a comprehensive hardware-software co-designed platform that integrates the essential blocks for a 5G-compatible physical layer subsystem into a powerful Radio Frequency System-on-Chip (RFSoC) platform, making it ready to drive future 6G experimentation, thanks to its low latency and scalability to higher bandwidth configurations. It incorporates a low-latency control interface to efficiently handle traffic over the fronthaul link, seamlessly connecting to a server. On the server side, it implements a C++ library to enable smooth integration with high-level applications through simple commands, ensuring ease of use and adaptability. The key HELIX features are:

Dynamic Multi-Split Functionality: HELIX operates as a multi-split platform, allowing real-time relocation of physical layer blocks between hardware and software (and vice versa). This capability aligns with the functional splitting options defined by 3GPP [19, 28], enabling researchers to explore various configurations and adapt to specific experimental needs.

Reconfigurable FR{X} Operation: The platform supports multiple numerologies suitable for multiple frequency bands such as sub-6 GHz (FR1) and mmWave (FR2) frequencies. By dynamically reconfiguring blocks, HELIX can seamlessly switch traffic between frequency bands without requiring a system reboot, supporting flexible experiments across diverse frequency bands.

Scalable Bandwidth and Carrier Aggregation: HELIX provides up to 4 carrier-aggregated channels with a combined bandwidth of up to 1670 MHz. These channels can operate with any combination of numerologies, transforming the platform into a multiband system suitable for advanced 5G and 6G research.

Fronthaul Interface with Side Information: The fronthaul interface is designed to transmit side information—such as Channel Frequency Response (CFR) and Signal-to-Noise Ratio (SNR)—to the server, enabling future 6G ISAC use cases.

Integration with Acceleration Abstraction Layer (AAL): Using a crossbar switch implemented on the RFSoC, HELIX can accelerate latency-critical functions using its AAL. This allows HELIX to also complement other platforms by offloading time-sensitive processing tasks, extending its utility beyond standalone use. The versatility of HELIX to support a variety of configurations and use cases is demonstrated through a series of experiments using different RFSoC platforms, showcasing its portability, integrated with sub-6 GHz and mmWave front ends operating at 28 GHz and 60 GHz. The platform can also be used with other frequency bands such as FR3 and sub-THz, both potential candidates for future 6G networks. The platform supports 1670 MHz of bandwidth with carrier aggregation, as well as bidirectional latencies down to 500 μs. Its innovative design addresses the scalability and integration challenges of current testbeds and bridges the gap between hardware flexibility and standard compliant performance, empowering researchers to explore advanced wireless technologies. We made this implementation available as open-source[1]

2 5G ARCHITECTURE PRIMER

In terms of capabilities, 6G is anticipated to build upon 5G technologies as an evolution [48]. This section explores key aspects of 5G that lay the foundation for developing a 6G experimentation platform. A significant milestone in 5G evolution is the introduction of disaggregated architectures, such as Open Radio Access Network (O-RAN). Unlike traditional Radio Access Network (RAN), O-RAN emphasizes flexibility, modularity, and scalability through standardized interfaces and open-source implementations. A major contribution of the O-RAN framework is its ability to decouple hardware and software components, enabling network operators to create tailored solutions for diverse scenarios.

2.1 O-RAN and Functional splitting

Task disaggregation is a cornerstone of 5G [19]. A defining feature of O-RAN is its support for functional splitting, where traditional base station functions are divided among Centralized Unit (CU), DU, and RU. This enables flexible deployment of RAN components tailored to specific service and network requirements.

Functional splits are classified into high-layer and low-layer options, as shown in Fig. 1. High-layer splits, such as between the CU and DU, separate control functions such as Radio Resource Control (RRC) from real-time tasks such as scheduling. Lower-layer splits, typically connecting the DU and RU, demand greater fronthaul bandwidth due to raw data transport, presenting implementation challenges.

Split 7.2x has emerged as an industry favorite because of its compatibility with O-RAN and its simplified RU design, simply implementing digital beamforming (in case of multi-antenna) and OFDM modulation, offering reduced size, power, and cost. This makes it a practical choice, from the operator point of view, for modern O-RAN deployments.

[1] https://github.com/IMDEANetworksWNG/HELIX

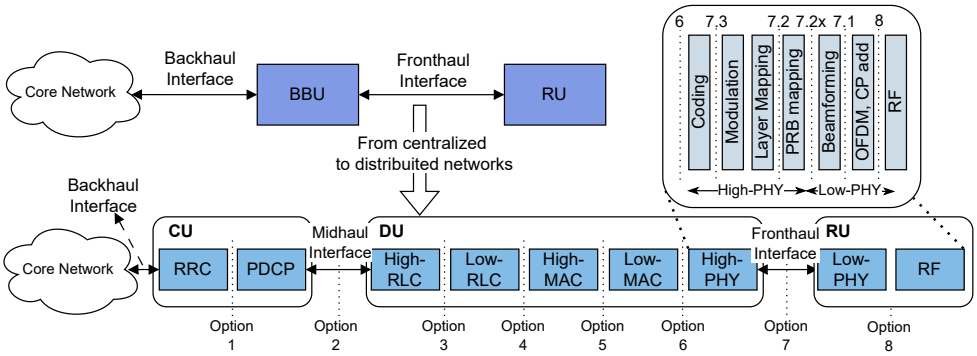


Figure 1: O-RAN architecture: in the example, the Distributed Unit (DU) is executing the High-RLC to High-PHY while the RU only executes the Low-PHY and RF tasks; different common splits are shown in the break-out box

2.2 Fronthaul interface

The DU and RU are connected through the fronthaul interface. O-RAN introduced a standardized fronthaul interface to ensure compatibility across different implementations [2].

The O-RAN fronthaul interface supports low-layer splits such as split 7.2x, where data is transported using the evolved Common Public Radio Interface (eCPRI) protocol on top of Ethernet. This interface requires precise timing to align operations between the DU and RU. There are clear benefits of having standardized fronthaul interfaces. However, there are still challenges to be solved. The specifications mainly focus on split 7.2x, which leaves a big part of the physical layer processing to the server side. As mentioned in Section 1, a massive increase in data rates is expected for 6G, driven by higher bandwidth, modulation and MIMO orders. Such physical layer configurations will impose extremely high fronthaul demands. As a toy example, a system with 4 spatial streams and 1600 MHz of bandwidth would require 200 Gbps of throughput over the fronthaul network using a split 7.2x configuration—this example only considers a fraction of the capabilities envisioned for 6G systems [35, 40].

3 HELIX TOP-LEVEL ARCHITECTURE

HELIX enables 5G and beyond experimentation by combining hardware and software components. From a high level perspective, the platform consists of: i) a powerful RFSoC board where Programmable Logic (PL) and Processing System (PS) cooperate to form a complete embedded system with hardware blocks that implement the physical layer components, tied to a control unit that manages the operation of the system; ii) a C++ library at the server side that allows the integration with applications; and iii) the fronthaul interface, which is carefully designed using hardware cores and software functions to connect the two subsystems levering UDP sockets. Altogether, these components enable a high-performance, flexible, and modular testbed that supports real-time deployments with high-bandwidth operation.

HELIX operates in two different modes: 1) as a standalone system for advanced wireless experimentation or 2) as a plug & play platform that can form a part of O-RAN projects by instantiating it within their framework as physical layer. In Fig. 2, we present the top-level architecture of HELIX, showing the main components and their interconnections. In this section, we cover the design of the fronthaul interface, control manager, and the functional split crossbar, which is key for flexible functional splitting. Building such platform is particularly challenging due to high bandwidth requirements, which demand carefully engineered buffering strategies and low-latency control interfaces to prevent bottlenecks and ensure sustained data throughput across the processing pipeline. Finally, we cover the C++ software library which controls the entire system from the server side. Transmitter and receiver processing blocks are presented in Section 4, since the operation of the system is independent of the operation of the blocks.

3.1 Custom fronthaul interface

For real-time platforms, the fronthaul interface manages the exchange of information between the server and the RU, including data transmission and control of radio parameters. In our design, this is implemented using the RFSoC. To support reliable links and diverse operational modes, the fronthaul must be robust, high-speed, and flexible. Within the O-RAN community, eCPRI is the preferred choice for fronthaul interfaces. However, eCPRI is primarily designed for split 7.x operations at the protocol level. Additionally, while some implementations targeting Field Programmable Gate Arrays (FPGAs) exist, these cores are often excluded from the licensing options accessible to most academic institutions. Other solutions, such as UHD [32], are heavily customized for USRP devices, making integration with other platforms challenging.

To address these limitations, we developed a custom interface that leverages UDP sockets over a 10Gb Ethernet link. This solution ensures robust, low-latency, and high-speed communication. Our design includes dedicated data plane, control plane, and a so-called *side-info* plane, each assigned to specific UDP ports, enabling independent handling of data streams and control information.

The *side-info* plane aims to support emerging 6G applications, such as sensing within an ISAC framework. HELIX facilitates the transmission of additional physical-layer information, such as channel estimates and SNR values, to the server's upper layers. This is achieved through a dedicated UDP port, ensuring efficient delivery of such metadata.

On the RFSoC, HELIX incorporates a hardware-implemented UDP stack based on [10], which we enhanced to support multiple UDP ports and Advanced eXtensible Interface (AXI) for seamless integration with other blocks. This implementation eliminates the

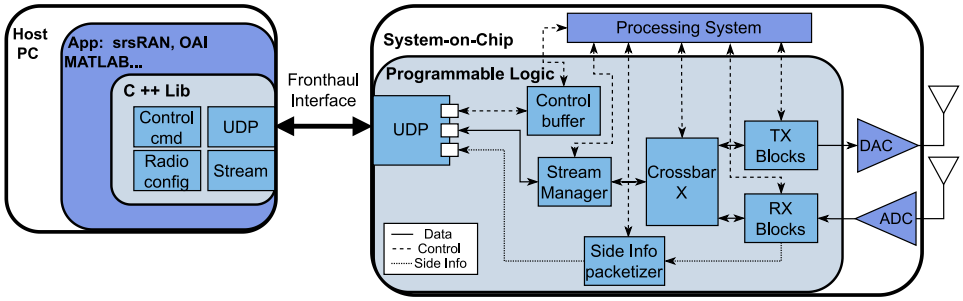


Figure 2: Top-level architecture of HELIX

need for offloading UDP processing to the PS, significantly reducing overall latency. Each UDP port is assigned a dedicated data-path. The *Data Plane* connects directly to the processing blocks via a stream manager and crossbar, enabling efficient data transfer. The *Control Plane* carries messages that are managed by the embedded PS through a control buffer, ensuring efficient handling of configuration and control commands. The *Side-info plane* collects relevant information from the processing blocks and packetizes it to offload it to the server for further real-time processing.

The architecture can provide the flexibility, speed, and low latency required for real-time operations while supporting advanced functionalities such as sensing and multi-port communication. Additionally, the designed fronthaul is transparent to the functional split used. Appropriate control messages configure the system to carry information from the relevant processing blocks based on the chosen split option. The combination of modularity and the protocol-agnostic nature of the fronthaul interface enables dynamic and flexible functional splitting, making the architecture adaptable to various deployment scenarios and evolving system requirements.

3.2 Control and Stream Manager

The RFSoC embedded PS plays a crucial role in managing the system's control plane. Messages received through the control plane port are stored in a buffer, which the PS reads to process and execute commands. These commands are used to: i) configure parameters for the processing blocks in a specific functional split, ii) change the functional split configuration (see Section 3.3), iii) manage data streaming operations and iv) adjust the fronthaul configuration. The embedded PS modifies configuration parameters in the processing blocks using the AXI-lite, enabling fast read/write operations between the PS and the PL. Key configurations include numerology, the number of Physical Resource Block (PRB), modulation order, and coding rate. Additionally, the control plane oversees the triggering of data transmission and reception.

These operations are orchestrated by a block in the programmable logic called *Stream Manager*. The Stream Manager ensures precise control of data streaming, including the start and stop of streams to maintain sample alignment. It also handles critical tasks such as clock domain crossing between the 10Gb Ethernet clock and the clock used by the processing blocks, ensuring seamless data transfer between different domains.

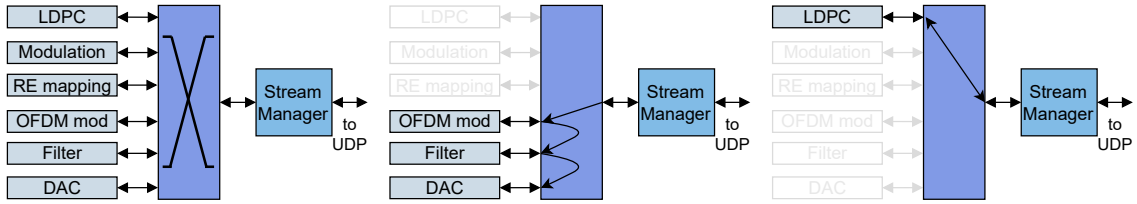
The PS further configures the side-info plane through specific control messages. This configuration determines which side information is shared with the server, optimizing fronthaul bandwidth usage in high-bandwidth scenarios. As detailed in Section 3.1, the side information is transmitted over a dedicated UDP port to avoid collisions with the data plane. The information is packetized and sent from the platform at the end of every received slot.

3.3 Crossbar for Flexible Splitting

A key feature of HELIX is its support for flexible functional splitting, enabling dynamic reconfiguration to meet varying network requirements. This capability is achieved through the integration of the AXI Stream Interconnect IP [44] from Xilinx's IP library. This crossbar interconnects AXI-stream-capable processing blocks, allowing for versatile data routing. While any block can theoretically connect to any other, limiting interconnections during design synthesis can lead to a more optimized implementation. During runtime, the crossbar's configuration can be entirely reprogrammed using simple commands from the control plane. Besides, the split configuration can be independently selected in the receiver and the transmitter, enabling asymmetric functional splitting, adding more flexibility to the platform. Additionally, our crossbar architecture allows switching between functional split configurations at runtime in less than 50 μs without the need to re-flash the FPGA image.

For instance, Fig. 3 illustrates the crossbar used in transmitter processing blocks (see Section 4.2) and various configurations for routing samples from the Stream Manager to the Digital-to-Analog Converter (DAC). In a split 6 configuration (Fig. 3a), data flows sequentially through the LDPC encoder, Modulator, Grid Builder (RE mapping), OFDM modulator, and finally the DAC. If a block needs to be replaced (e.g., with a different implementation), it can simply be disconnected from the crossbar, and the new block added without significant redesign. In a split 7.2x configuration (Fig. 3b), the crossbar adjusts its internal switching, connecting data from the Stream Manager directly to the OFDM modulator. This reconfiguration is achieved by sending a control command from the server. Similar logic applies to other split configurations and receiver blocks, enabling broad experimentation possibilities beyond the physical-layer algorithms implemented in this paper.

As discussed in Section 3, HELIX is designed to work in tandem with other platforms within the context of the AAL defined in O-RAN terminology. The AAL allows a DU to offload critical functions to a pool of accelerators, such as GPUs, FPGAs, or other specialized hardware [5], depending on network requirements. Fig. 3c



(a) Split 6 **(b) Split 7.2x** **(c) Hardware acceleration**

Figure 3: HELIX crossbar for flexible functional splitting

demonstrates how the crossbar facilitates this functionality by routing data from the Stream Manager to an accelerator (e.g., the Low-Density Parity-Check Code (LDPC) encoder), performing the necessary computation, and returning the processed data to the DU for further processing. This setup supports not only single-block offloading but also multi-block configurations based on crossbar interconnections.

Additionally, third-party blocks can be integrated into the crossbar in HELIX, allowing them to function as hardware accelerators, or to replace or cooperate with existing blocks, thereby extending the system's functionality. For example, to support a new or experimental coding scheme - such as non-binary LDPC or polar code variants envisioned for 6G - the corresponding block can simply be added to the crossbar, and samples can be redirected to this block instead of the *legacy* LDPC decoding block. This design enhances the system's modularity and flexibility, aligning well with the principles of O-RAN, promoting interoperability and scalability.

Algorithm 1 Radio Configuration and Streaming

Input: IP address `ip = 192.168.5.10`
Functional split `split_conf_rx = 6`
`split_conf_tx = 7.2x`
Create `radio_example = radio(ip)`
{Configure processing block parameters}
`radio_example.config.set_ofdm_config(...)`
`radio_example.config.set_ldpc_config(...)`
`....`
`radio_example.config.set_split(split_conf_rx)`
`radio_example.config.set_split(split_conf_tx)`
#changes the crossbar configuration at tx and rx
{Data streaming}
`radio_example.stream.transmit(tx_buffer)` *#sends a trigger and transmit the data through the UDP socket*
`radio_example.stream.receive(rx_buffer)`

3.4 Interface Library

To simplify the use of HELIX and facilitate its integration with high-level applications, we developed a C++ interface library that serves as a wrapper between the implementation on the RFSoC and network applications. The library was designed with three main objectives in mind: (i) function abstraction and usability, (ii) seamless integration with applications, and (iii) high performance.

The library provides a user-friendly interface that abstracts the low-level operations needed to configure processing blocks, start transmission, enable reception, and set up splitting options. By hiding the platform's implementation details, the library significantly improves accessibility for users who may lack in-depth knowledge of the system's architecture. Internally, the library manages essential tasks such as radio configuration, UDP transport and port management, and handling of control messages. A notable advantage of the library's design is its portability. If the implementation is migrated to a different platform, the abstracted functions allow for smooth transitions with minimal rewriting, avoiding the need to recreate functionality from scratch.

At the core of the library is a `radio` parent class, which encapsulates all essential functionalities, including data streaming and radio configuration. The library supports the integration of multiple HELIX instances within a single application, making it highly versatile. For instance, the `radio` class can be integrated into O-RAN implementations such as srsRAN or OAI, replacing the legacy radio instantiation. Users can interact with HELIX through simple function calls like `radio.send()` and `radio.transmit()`, enabling seamless data streaming over UDP sockets. Alg. 1 provides a concise example of how to use the library for enabling data transmission and reception. After instantiating the parent class, users can configure various processing blocks (based on specific parameters), select a split configuration, and initiate data transmission and reception—all with minimal code.

Finally, the library is designed with a strong focus on performance, ensuring low-latency and high-throughput transmissions. To achieve this, the library verifies that the platform is prepared for data streaming, whether for transmission or reception. For instance, to initiate a transmission, the C++ library sends a simple trigger signal just before data streaming begins and waits for an acknowledgment reply. This process takes approximately 30 μs, ensuring minimal latency and supporting high data rates, as demonstrated in the experimental results.

4 PROCESSING BLOCKS

In this section, we address the design of the processing blocks on the PL in the RFSoC. All blocks are wrapped with AXI interfaces to connect them with the crossbar (Section 3.3). To support the highest numerology (Subcarrier Spacing (SCS) = 240 kHz), the IQ samples are processed at a sampling rate of 491.52 Msps. To relax the requirements for FPGA place & route phase, we choose a clock frequency of f_{clk} = 245.76 MHz and thus the datapath handles two (complex) samples per clock cycle. Processing multiple samples in parallel in communication transceivers is challenging because many DSP algorithms—such as filters, FFTs, or decoders—rely on dependencies between input samples. When several samples are processed in parallel the design should manage how data is shared,

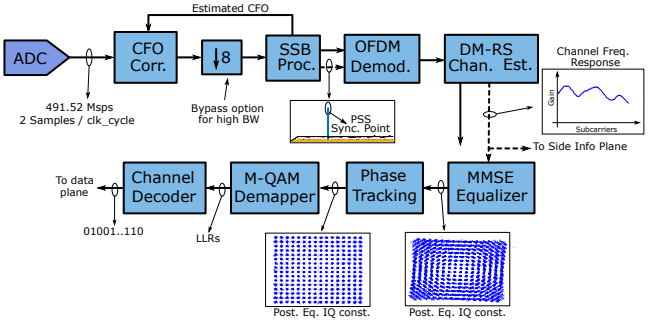


Figure 4: Receiver processing block diagram

reused and ordered, introducing additional complexity in the datapath. For the lower numerology (SCS = 30 kHz), instead of changing the clock frequency, we keep the same datapath and process two samples every eight clock cycles controlling this behavior using the AXI interface. Thus, the same processing blocks can be used for both numerologies, allowing the design to seamlessly transition between them without requiring any reprogramming. All the inputs and outputs of the blocks are quantized to $W = 16$ bits to match the interfaces to the AD/DA converters.

4.1 Receiver blocks

To explain the blocks that compose the receiver sub-system, we go from the samples from the ADC to the decoded bit sequence, as they pass through all the processing blocks, as shown in Fig. 4. Here, we assume a split 6 configuration and omit the crossbar (Section 3.3) for clarity.

Carrier Frequency Offset (CFO) correction implements the CORDIC algorithm [39] to compensate for the frequency mismatch between the transmitter's and receiver's local oscillators. It uses the estimated CFO from the Synchronization Signal Blocks (SSB) block (see below).

Bypassable downsampling filter is designed as a *div8* downsampling filter with two output paths to accommodate different subcarrier spacing (SCS) requirements. One output feeds the SSB processing block to retain the subcarriers where the SSB is located. The second output is selected based on the SCS: for SCS = 30kHz, the output is taken from the downsampling filter, while for 240kHz, the filter is bypassed, feeding the input directly to the output.

SSB Synchronization: In 5G, the SSB enables initial cell search, synchronization and beam management, using Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) signals. Our design, based on [24], includes an extra div8 downsampling filter for SCS=30kHz, bypassed for 240kHz. The PSS estimation sub-block correlates the signal with a time-domain PSS sequence,[2] using a dynamic threshold computed via a moving average filter to detect a valid PSS, as shown in Fig. 4. The CFO is estimated from the phase of the Cyclic Prefix (CP) autocorrelation, averaged across OFDM symbols in the SSB, with a CORDIC implementation in vectoring mode [39]. Synchronization is achieved using a dual-port RAM acting as a shift register, whose length adapts to the SSB's slot position, outputting the slot's start upon valid PSS detection

OFDM demodulator includes a 2048-FFT core designed with the help of HDL-coder (Matlab), which allows to process multiple samples in parallel. We include digital logic to remove CP (considering that first OFDM symbol in the slot has a longer CP than the rest of symbols), apply Fast Fourier Transform (FFT), zero-frequency centering, and removal of the null subcarriers. The block is triggered by the synchronization from the SSB, which allows to retain the OFDM symbols. Note that because of the removal of CP and null subcarriers, the output stream has fewer samples than the input one, reducing overhead.

Demodulation Reference Signal (DM-RS) channel estimation: we implement a pilot-based based channel estimation based on DM-RS in the slot [1] using a two-step procedure. First, we perform an LS estimation $\hat{H_0}[k] = \frac{Y[k]}{X[k]} \; \forall \; k \in \mathcal{K}_{\text{DMRS}}$, where $\mathcal{K}_{\text{DMRS}}$ is the set of subcarriers where DM-RS are located, $Y[k]$ is the received frequency domain signal, $X[k]$ is the transmitted DM-RS and noise was omitted to enhance clarity. Then, the channel is estimated for *all* subcarriers using a kernel regression method [31, 38] that takes the LS estimates over DM-RS subcarriers and applies a Gaussian kernel function $K(k,m) = \exp\left(-\frac{(k-m)^2}{2\sigma^2}\right)$ that determines the influence of the estimated channel at subcarrier m on the estimation at subcarrier k. σ is the *width* (bandwidth) of the kernel, i.e., it controls the influence of the neighbor subcarriers to the one of interest. The interpolated channel at subcarrier k is given by a weighted sum of the known estimates at the DM-RS subcarriers:

$$\hat{H}[k] = \frac{\sum_{m \in \mathcal{K}_{\text{DMRS}}} K(k,m) \cdot \hat{H_0}[m]}{\sum_{m \in \mathcal{K}_{\text{DMRS}}} K(k,m)} \tag{1}$$

In our implementation, LS estimates are computed by multiplying with inverse of the DM-RS symbols $X[k]^{-1}$ to avoid unnecessary divisions. The kernel regression was implemented as a frequency domain filter over $\hat{H_0}$, upsampled to all subcarriers. Finally, since the noise variance is required to implement Minimum Mean Square Error (MMSE) equalization, we compute it from the LS and kernel estimations using

$$\sigma_n^2 = \frac{1}{|\mathcal{K}_{\text{DMRS}}|} \sum_{k \in \mathcal{K}_{\text{DMRS}}} \left|\hat{H_0}[k] - \hat{H}[k]\right|^2 \tag{2}$$

MMSE equalization receives the demodulated OFDM symbols, channel $\hat{H}[k] \; \forall k \in \mathcal{K}$ and noise variance σ_n^2 to compute the equalized symbols $\hat{X}$ as $\hat{X}_{eq} = \frac{\hat{H}^*}{|\hat{H}|^2+\sigma^2} Y$. First, the denominator is computed and then a divider (from the Vivado IP library) is employed to compute its inverse. Then, the output of the divider is multiplied by the conjugate of ($\hat{H}^*$) and then by the input symbols Y. We implement the necessary synchronization and state machines in the block to ensure proper alignment of the different data-paths, considering that these come at different time instants.

Phase tracking: in 5G, Phase-Tracking Reference Signal (PT-RS) are used to mitigate phase noise and improve demodulation accuracy, especially in high-frequency bands like mmWave. These are distributed in the time and frequency domain with a granularity that depends on the configuration. The block first estimates the residual channel $\hat{H_{res}}$ by LS followed by the kernel regression method,

[2] Currently, only one of the three PSS sequences is supported, with plans to include the others.

similar to the channel estimation. Estimation is performed on all symbols where PT-RS are located, and then averaged to reduce the effect of noise. Next, the angle θ of the estimated residual channel is computed using a CORDIC implementation [39] in rotation mode and then compensating for the phase by another CORDIC in vectoring mode.

Symbol demapping is implemented via a soft decision demapper for M-QAM modulation schemes using the method from [23] which allows to compute an approximation of the Log-Likelihood Ratios (LLRs) suitable for soft-decision channel decoders. Following the symbol mapping from 5G [1], we reuse the multipliers required to compute the LLRs regardless of the modulation order which saves FPGA logic elements. While [23] uses the noise variance in the computation, we omit it to avoid unnecessary divisions. Moreover, for channel decoders based on the Min-Sum algorithm, knowledge of the statistics of the channel is not required [8]. Once LLRs are computed, we apply de-scrambling with the same sequence used at the transmitter side. For simplicity, the sequence is fixed and pre-stored in the block.

Channel decoding: we implement an LDPC decoder combined with rate matching as per [26]. We use the hardened LDPC decoder embedded on the RFSoC [13], which implements the Min-Sum LDPC decoding algorithm [8], with configurable scaling factor, decoding iterations, and early termination to improve throughput. The core supports the parameters defined in the standard and uses AXI to ease integration with larger designs. We include digital logic to implement the rate matching and interleaving algorithms following the structure defined for the modulation.

4.2 Transmitter blocks

The transmitter block's key challenge is ensuring high-fidelity signal generation to withstand channel impairments. This depends heavily on a carefully designed quantization strategy to maximize dynamic range. Additionally, maintaining proper data rates across all blocks is crucial to prevent processing interruptions. To address this, we use appropriately sized FIFOs and state machines that ensure continuous processing by triggering blocks only when sufficient data is available. Below, we provide a brief overview of each transmitter block's function.

Modulator takes the sequence of coded bits and maps them to the alphabet defined by the modulation order being used. The mapping criterion is defined in [1]. Prior to the mapping, it scrambles the input bit sequence with the *fixed* scrambling sequence.

Grid builder merges the data symbols from the modulator with the reference symbols (DM-RS, PT-RS) and with the SSB . The block is configurable for different densities (time and frequency) of reference symbols and to locate the SSB in the slot. We design the grid and modulator block using the high-level synthesis tool from the FPGA vendor (Vitis-HLS).

OFDM modulator uses an instance of the FFT block designed for the receiver to implement the inverse FFT by computing the complex conjugate of the input and output of the core. Besides, we design the control logic to insert null subcarriers (prior to IFFT) and append CP (after IFFT).

Bypassable upsampling filter feeds the samples to the DAC at 491.52 Msps in case of the highest SCS (240 kHz). This is the most rate challenging case, and the whole processing chain is designed to ensure this data rate. For the lower SCS (30 kHz), the block consumes two samples every eighth clock cycle and feeds them to an 8× upsampling filter prior to the DAC. Since samples are consumed at a lower pace, we also ensure that FIFOs in the pipeline do not overflow at any time.

5 IMPLEMENTATION

We implement HELIX on the Xilinx RFSoC ZCU111 and ZCU208 boards [45, 46], which integrate Giga-sampling rate AD/DA converters, large number of logic elements, multi-core ARM processor, 10GbE interface and hardened LDPC decoders. However, HELIX is designed to be platform independent thanks to its modularity and standard interfaces. A full implementation of HELIX on those boards requires 22% of the available DSP units, 21% of logic blocks and 45% of Block-RAM. These results show that there is still margin to scale to higher bandwidth configurations as we will show in Section 6.3, since most of the BRAM is used for network buffers that do not augment with the number of channels.

We measured the inherent latencies of the processing blocks. For the transmitter blocks the total latency is 9307 clock cycles (37.87 μs with f_{clk} =245.76 MHz), where the OFDM modulator is responsible for 99% of the latency, due to the CP addition and the FFT computation time itself. For the receiver blocks, latencies sum up to 24701 clock cycles (100.51 μs) with 41% of the latency caused by the SSB synchronization that buffers OFDM symbols until the PSS is detected. The channel decoder constitutes 27% of the latency due to the iterative nature of the algorithm.

HELIX supports a variety of RF front ends, enabling flexibility across different frequency bands. Using the Numerical Controlled Oscillator (NCO) embedded in the AD/DA converters, it is possible to translate the signal in frequency depending on the requirements. For example, for mmWave front ends such as the Sivers 28 GHz and 60 GHz EVK [33] [34] that we use in our experiments, the converters in the RFSoC are configured in IQ/IQ mode, where separate in-phase and quadrature components are processed independently. This configuration is essential for interfacing with the mmWave front ends. Moreover, since in 5G the DC subcarrier conveys information, we employ a low-Intermediate Frequency (IF) architecture using the internal NCO of the converters, and then the mmWave front ends perform the final up/down conversion to RF frequencies. A similar approach could be followed to use terahertz and sub-terahertz front ends. For sub-6 GHz, the AD/DA are able to perform direct RF conversion, by setting the converters in an IQ to real mode (Tx) and vice versa (Rx). The real-time configurable NCO perform the conversion, allowing to quickly select and update the FR1 band *at run-time* from the server with a simple command. Regarding server requirements, only a multi-core server and a 10GbE card are required, with sufficiently large network buffer size (>30 MB) and the interface configured to jumbo packet mode.

6 EVALUATION

In this section, we evaluate the performance of our platform through microbenchmarking and experimental analysis. The microbenchmarks assess key physical layer metrics such as constellation quality,

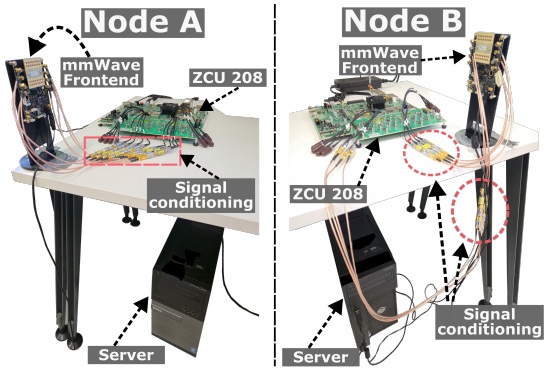


Figure 5: Evaluation setup.

channel estimation accuracy, and Bit Error Rate (BER). We evaluate throughput and fronthaul overhead on system and platform level. Next, we conduct experiments to showcase the potential of the platform in terms of multiband support, scalability, and ISAC support.

To demonstrate the highest numerology (i.e., SCS=240 kHz over mmWave frequencies), we deploy HELIX on two nodes based on the Xilinx RFSoC ZCU208 development board, with mmWave front ends including phased antenna arrays from [33], operating in the 24 GHz to 28 GHz band. We employ one server for each node, attached via 10GbE. While we perform the experiments with mmWave and sub-6GHz front-ends, HELIX can be connected to front-ends in other frequency ranges, such as FR3 band or sub-THz.

6.1 Microbenchmarks

Transmitter quality: To evaluate that our hardware implementation of the transmitter blocks does not introduce significant quantization noise that distorts the transmit signal, we compare (using split 6 configuration) the signal that goes to the DAC, i.e., the output of the Tx upsample block, with its counterpart software implementation (i.e., split 8). As can be seen in Fig. 6a, the total quantization errors are on the order of 10^{-4}, which indicates that the hardware implementation introduces minimal distortion. This low error magnitude demonstrates that the hardware design closely replicates the ideal software model, maintaining signal integrity and meeting the stringent quality requirements of 5G transmission.

CFO compensation: in OFDM, CFO is particularly harmful as it causes inter-carrier interference (ICI), which degrades the received signal. This effect is especially significant at mmWave frequencies, where CFO can be particularly high. Fig. 6b shows the estimated CFO from the SSB over 280 slots, both without compensation and after applying CFO correction using our processing blocks. As shown, after correction, the CFO is reduced to within a fraction of the 240 kHz SCS, effectively mitigating its impact. Notably, this experiment was conducted at 60 GHz mmWave, demonstrating the suitability of the design for high mmWave frequencies.

Channel estimation: One important aspect to consider when implementing functionalities for the FPGA logic is the potential performance degradation caused by fixed-point quantization, as well as design choices made to optimize resource usage and throughput. To illustrate this, we compare the channel estimation in the split 6 configuration—using the method described in Section 4.1—with its software counterpart in the split 8 configuration.[3] As shown in Fig. 6c, the implemented kernel regression method preserves the main characteristics of the channel, although some *fast* oscillations are not captured. Nevertheless, the impact of these oscillations is negligible, as evidenced by the IQ constellation plots in Fig. 7. A similar evaluation was conducted for the remaining processing blocks during the design stage, confirming that the fixed-point implementations preserves performance across the pipeline.

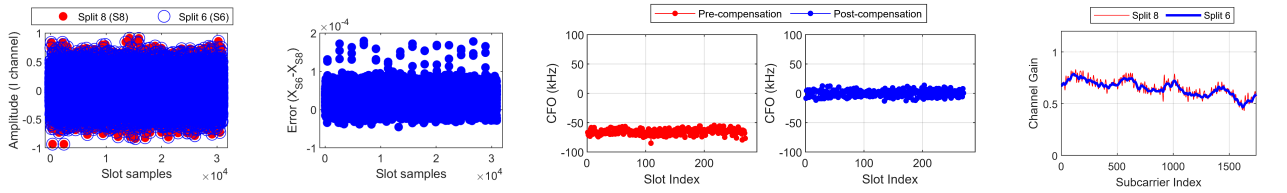
Constellation analysis: The equalizer and phase tracking were evaluated by analyzing the constellations for all modulation orders supported by HELIX. In this experiment, two HELIX nodes equipped with 28 GHz mmWave front ends were deployed in a line-of-sight setup at a distance of 1.5m. Fig. 7 shows the results, where the distinct point clouds indicate that HELIX successfully supports *all* modulation orders defined by 5G. Results for sub-6GHz were omitted due to space constraints, but the constellation shown in Fig. 4, taken with 30 kHz SCS and a 2.3 GHz carrier frequency, demonstrates even better quality than Fig. 7d, as expected.

BER analysis: the most important test for our HELIX physical layer implementation is the BER. For this experiment, we replicate the constellation analysis setup, but vary the transmit power at the mmWave front end for different SNR values. We set three different rates (low, medium and high) using rate matching capability of HELIX. Due of lack of space, we only show two modulation orders, QPSK and 256QAM. For the QPSK case, we compare split 6 (using all the hardware receiver processing blocks) with split 8, where all processing is performed on the server. Results are presented in Fig. 8a, where we see a consistent 2 dB difference comparing the two implementations. In Fig. 8b, we show the results for 256QAM modulation. In this case we choose the example of a 7.3 split to compare against split 8. Here, we also note the 2 dB separation between the curves. The reasoning behind this is that split6 and split7.3 include *all* the effects of fixed-point arithmetic and approximate algorithms, whereas split 8 benefits of floating-point arithmetic and optimized algorithms *only* suitables for DU implementations.

6.2 System level evaluation

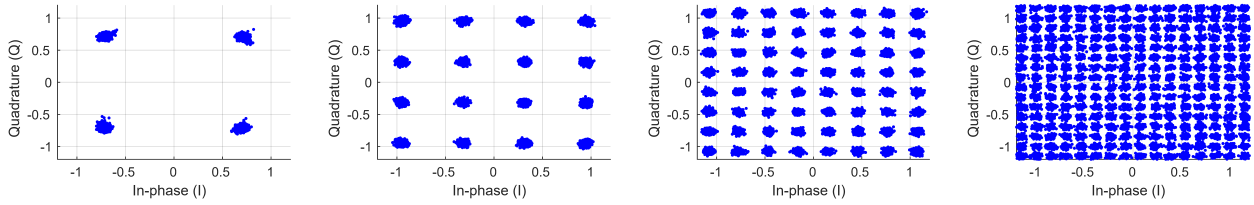
Latency: In this experiment, we analyze the end-to-end latency of the system. We send a message from Node A to Node B and wait for a reply. Then, we measure the time difference in Node A after receiving the reply from Node B. We utilize a split 6 configuration to measure the inherent latency of our implementation at the system level. We repeat the same experiment for more than 40 seconds to extract latency statistics. The results of the experiment from Fig. 9 show that the median end-to-end latency is 544.77 μs and 90% of the points are below 625.77 μs, with a limited number of outliers. Considering the results from Section 5, the delay from AD/DA and the latency of the trigger command from the server to the platform, the measurements from this experiment are coherent. According to these results, HELIX can be utilized for Ultra Reliable Low Latency Communication (URLLC). The existence of outliers is due to the instability of the interrupt-driven kernel socket library. This could be solved by using polling-based packet processing in user space,

[3]For this, we estimate the channel using the function from the MATLAB 5G Toolbox [25].



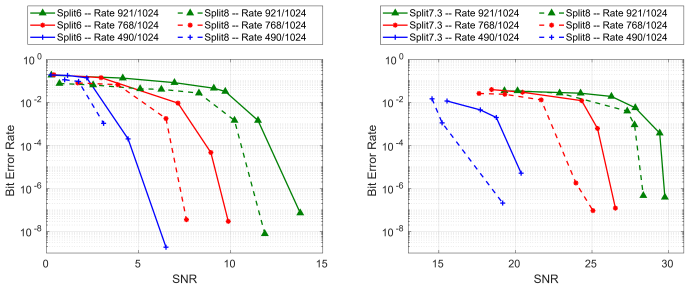
(a) Transmitter comparison (b) CFO pre and post compensation (c) Channel est. comparison

Figure 6: Processing blocks validation



(a) QPSK modulation (b) 16QAM modulation (c) 64QAM modulation (d) 256QAM modulation

Figure 7: Over-the-air transmission



(a) Split 8 and 6 - QPSK (b) Split 8 and 7.3 - 256QAM

Figure 8: BER curves over-the-air

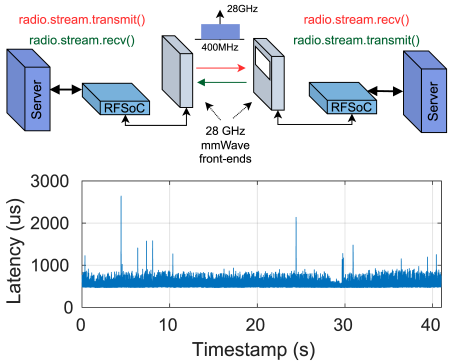


Figure 9: End-to-end latency

e.g., via the Data Plane Development Kit (DPDK) library. This would enable higher packet throughput and more precise scheduling.

Fronthaul overhead: The fronthaul requirements change depending on the split configuration. The lower the split configuration (i.e., split 8), the higher the demands in the fronthaul link. In this experiment, we measure the fronthaul overhead of the different available split configurations in HELIX. We transmit 50000 slots and capture the received packets using Wireshark. We use 145 PRB, which for an FFT size of 2048 corresponds to the 84% of available subcarriers. We compare with 73 PRB, which corresponds to 42% of subcarriers. For this experiment, we also include the overhead caused by transmitting the channel estimation generated from the platform. We considered the maximum possible overhead as the size of a slot in split 8 (i.e., 120KB). The results in Fig. 10 show that the higher the functional splitting, the lower amount of overhead in relation to split 8. The overhead is constant at split 8 among different number of resource elements, since we are working in the time-domain. In addition, in split 7.2x and 8 the channel estimation is implemented in the DU with no additional data to transmit. The results show that for a high number of PRBs, the overhead is still high for splits lower than 6. For the split 6 configuration, the overhead is reduced 92% compared to split 8—even for the most demanding situation where the number of resources is high and the channel estimation is transmitted through the fronthaul interface

Split throughput: As mentioned in last section, lower splits suffer from higher fronthaul bitrate. This issue can impact throughput performance, since the fronthaul interface gets congested because of the high overhead. We conducted a throughput analysis through different functional splits, number of PRB and coding rate, covering different lower and higher throughput scenarios. The results are shown in table 1. Split 6 clearly outperforms the other split configurations in any scenario. The reason is that the split 6 overhead is low enough to be close to theoretical throughput. When the number of PRB is low, the performance of 7.2x is closer to split 6, but when the channel reaches its maximum capacity using 145 PRB, the throughput does not increase at the same pace. The performance of split 8 is approximately 5 times lower than the performance in split 6. As expected, for 256QAM modulation, the throughput performance is

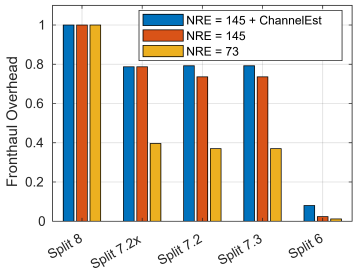


Figure 10: Fronthaul overhead for different functional splits

approximately a factor of 4 higher, with close to 1200 Mbps for 145 PRB and rate 921/1024.

6.3 Applications

In this section, we discuss how HELIX can be used as basis for 6G experimentation beyond the capabilities of 5G.

ISAC-ready platform: sensing is expected to be an integral part of 6G networks where the high bandwidth can be exploited to implement advanced sensing applications such as Human Activity Recognition (HAR), by reusing channel measurements. This is an application exploiting the real-time capability of HELIX, since data collection is not constrained by on-board memory. We showcase ISAC capability by setting up 2 HELIX nodes side by side with 28 GHz mmWave front ends. The system transmits for 20000 slots separated by $T_c = 0.5$ ms each. We set HELIX in split 6 mode with channel measurements through the side info plane enabled. The experiment consists of a person walking back and forth in front of the HELIX nodes while the slots are being transmitted, as shown in Fig. 11.

Collected channel frequency responses are converted to the time domain to obtain the Channel Impulse Response (CIR) and aligned with respect to the Line-of-Sight (LOS) path, thus preventing synchronization offsets from the PSS. Once CIRs values are aligned, we employ CFO cancellation techniques using the LOS as an anchor [29, 43]. This is needed because the CFO estimation from SSB includes the effect of both static and dynamic paths (targets) in the environment. Finally, the μ-Doppler signature is computed by Short-Time Fourier Transform (STFT) using a window size of $M = 100$ samples. With these parameters, the maximum measurable velocity corresponds to $v_{max} = \frac{c}{4 \cdot f_c \cdot T_c} = 5.35$ m/s, which is enough to sense a person at walking speed. A result of this experiment is presented in Fig. 11, where the typical pattern of increasing and decreasing velocity of the person walking is observed. Moreover, we see small oscillations around the main signal, due to the arms and legs. This experiment shows how the platform can be used for advanced sensing applications based on channel measurements.

Scalability: 6G demands significantly more bandwidth than 5G, necessitating massively scaled-up systems. Conventional SDR-based architectures require powerful servers and multiple USRP-like devices with complex synchronization, posing scalability challenges. To demonstrate the scalability of HELIX, we implemented four transmitter and receiver processing blocks on two independent HELIX platforms. Despite a full split-6 implementation, the four transmitters utilized only 28% of logic blocks, 12% of DSPs, and 58% of block RAM. For four receivers, the design required 48% of logic blocks, 72% of DSPs, and 80% of block RAM, with block RAM being the primary bottleneck, as detailed in Section 5. To further enhance scalability, we plan to offload buffers to external DRAM. Importantly, using two RFSoCs for independent transmitter and receiver functionalities does not affect system integration, as multiple devices can be seamlessly instantiated within an application.

Fig. 12 illustrates the operation of four parallel transmitters with 418 MHz bandwidth each. For simplicity, the experiment used four DACs, each configured with a different NCO frequency (300, 750, 1200, and 1650 MHz) to create a 1670 MHz composite channel. However, a single DAC could be used to generate the four channels by implementing the NCO in the FPGA logic and feeding the DAC with the composite signal. By porting the design to a ZCU216 board and leveraging external DRAM for buffering, the design could be scaled to support up to 8 parallel channels, optimizing BRAM usage. Thanks to the high-rate and 256QAM support, HELIX achieves multi-Gbps throughput with split 6. Note that this experiment goes beyond the numerologies defined in 5G-NR, positioning HELIX at the forefront of 6G experimentation, where higher bandwidth per user is expected. Moreover, by porting HELIX to an RFSoC platform with a greater number of AD/DA converters —such as the AMD Xilinx ZCU216 [4]— it becomes possible to aggregate more channels and thereby achieve an even wider bandwidth.

Multiband: a key advantage of HELIX is its ability to seamlessly adapt processing blocks to different numerologies using the same FPGA resources (Section 4), by simply reconfiguring the Tx Upsample and Rx Downsampling filters. As shown in Section 5, sub-6GHz and mmWave front ends require distinct converter configurations. To enable multiband operation, we first consider a single implementation where traffic flows to either one frequency band or the other. On the transmitter side, an AXI demultiplexer redirects samples from the Tx Upsample block to the DACs connected to the mmWave or sub-6GHz front ends. On the receiver side, an AXI multiplexer selects inputs from the ADCs of either front end. Switching between bands is extremely fast, requiring only a simple control plane command.

The second multiband option considers the case where traffic can be sent concurrently via multiple bands. As discussed above, since two (or more) transmitter and receiver processing blocks can be used concurrently, the application can easily split the traffic between the different interfaces. Moreover, in the 4 channel cases, it is possible to use up to 4 different frequency bands altogether, for example, two at different frequencies in the sub-6 GHz band, one at 24-28 GHz and another at 58-70 GHz. More combinations are possible upon availability of front ends for other frequencies.

Acceleration Abstraction Layer (AAL): As mentioned before, HELIX can be used to offload RAN functionality. To evaluate this, we conduct an experiment where instead of processing the full PHY pipeline (or some part depending on the split), the crossbar dynamically changes the data path to route input samples coming from the UDP socket to specific processing blocks. The processed samples are sent back to the RAN using the same interface. This experiments demonstrate how HELIX is able to work in the AAL in conjunction with other platform using the same design.

We evaluate three cases of task acceleration: OFDM demodulation, LDPC decoding and a part of the receiver pipeline including

Table 1: Throughput in Mbps for QPSK modulation

Functional Split	NPRB = 145 and rate 921/1024	NPRB = 73 and rate 921/1024	NPRB = 145 and rate 490/1024	NPRB = 73 and rate 490/1024
Split 6	278,91	119,45	155,16	65,12
Split 7.2x	115,36	75,10	61,49	43,49
Split 8	57,02	27,46	29,66	14,77

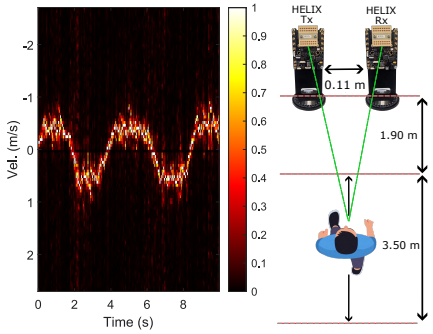


Figure 11: μ-Doppler signature from a walking person using channel measurements from HELIX

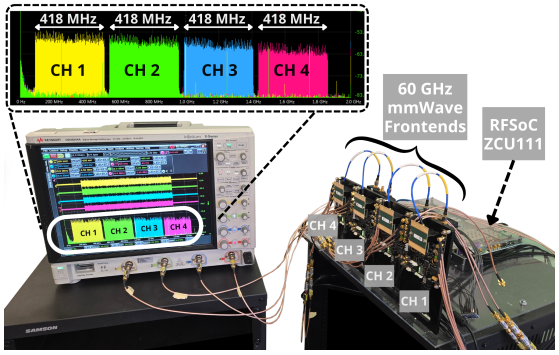


Figure 12: 4 aggregated channels to form a wider 1670 MHz system measured with an oscilloscope

cascaded processing blocks from OFDM demodulator to LDPC decoder. We separate the latency introduced by processing blocks from that of the fronthaul interface and measure throughput at the output of the blocks. For OFDM demodulation, we analyze the effect of different number of PRBs. As shown in Fig.13a, total latency remains constant, with the fronthaul interface contributing more than 85% of it. However, Fig.13b shows that the throughput increases because of the higher spectral efficiency. For decoding acceleration, we examine the impact of the number of LDPC iterations. Fig.14 shows that higher numbers of iterations worsen the performance while increasing the coding rate and a higher number of PRBs improves throughput but increases latency. Finally, we analyze offloading of cascaded processing blocks at the receiver, sending IQ samples to the OFDM demodulator all the way to the LDPC decoder, offloading the decoded bits through our interface, with equal number of the LDPC iterations. Fig.15 shows that this setup has the highest latency and lower throughput due to the high fronthaul overhead required to send the IQ samples to the OFDM demodulator block. All experiments show that the fronthaul interface impacts the overall latency. A more optimized communication scheme (e.g., using a polling based library) could improve the throughput and latency results.

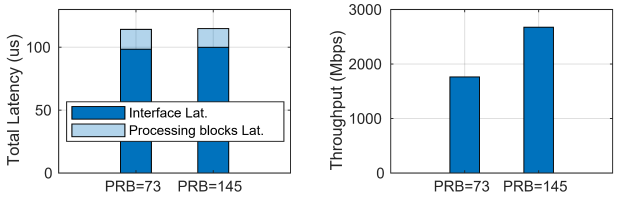
7 RELATED WORK

We describe related work on wireless experimentation platforms in the categories (i) high-bandwidth, non-real-time platforms, (ii) software-based end-to-end testbeds, and (iii) FPGA-based real-time systems.

High-bandwidth, non-real-time platforms: The highest degree of flexibility can be achieved with signal generators together with oscilloscopes or signal analyzers. These tools allow researchers to generate and capture arbitrary signals within the operational limits of the equipment. However, these devices are typically used in smaller lab setups, cannot operate in real-time, and are limited to the physical layer. Similar SDR-based approaches are more portable but also lack real-time 5G/6G capabilities [9, 12, 17, 18, 27].

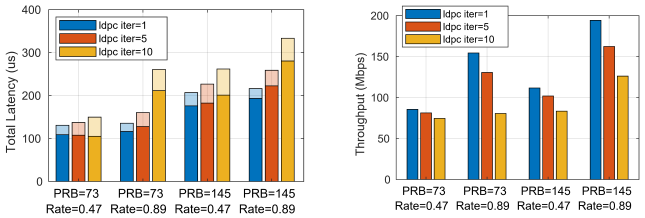
Software-based end-to-end testbeds: There are several software-based implementations for CPUs (e.g., srsRAN [11], OAI [15], and Amarisoft [3]) as well as Nvidia's Aerial [16] implementation for GPUs. While these platforms are flexible and relatively easy to customize, the possibility to integrate them in end-to-end systems comes with limitations in terms of performance (only up to 100 MHz of bandwidth) and the more fundamental issue of fronthaul data rates, stemming from the need to transfer IQ samples to the CPU or GPU. Recognizing these challenges Borromeo et al. [7] present an implementation that offloads low-PHY functions to an FPGA, showing a reduction in processing time, which is partially offset by the memory transfer between FPGA and CPU. Furthermore, open-source initiatives such as srsRAN [11] or OAI [15] provide software-based solutions for 5G. srsRAN supports subcarrier spacing of 15 and 30 KHz for FR1 but does not support FR2, while OAI supports FR2 with 120 kHz subcarrier spacing and 100 MHz of bandwidth. Other projects incorporate hardware accelerators to enhance performance. Savannah [30] integrates an ASIC accelerator for LDPC decoding [22] together with software baseband processing to achieve 100 MHz of bandwidth and 2x2 MIMO in FR2.

FPGA-based real-time systems: Considering the trade-offs of the discussed platforms, there is an increasing interest in FPGA-based real-time implementations [6, 14, 18, 41]. Given the challenge to implement real-time systems, existing projects have limitations, i.e., do not support the full bandwidth [6, 41] or focus on specific physical layer functionality like beam tracking [14, 18]. With HELIX, we present the next generation in this line of research, releasing a flexible platform that supports both sub-6 GHz and



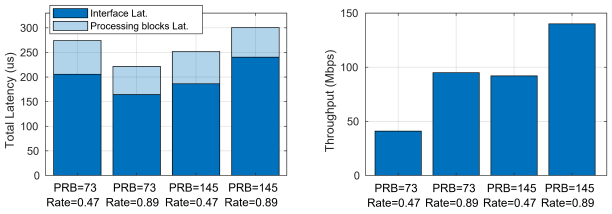
(a) Latency (b) Throughput

Figure 13: FFT hardware acceleration.



(a) Interface latency (dark colors) and Processing latency (light colors) **(b) Throughput**

Figure 14: LDPC hardware acceleration.



(a) Latency (b) Throughput

Figure 15: FFT - Equalization - Phase tracking - Demapping - LDPC acceleration.

mmWave bands with a bandwidth of 417 MHz, data rates up to 1200 Mbps, and end-to-end latencies as low as 500 μs.

8 DISCUSSION

Integration with high-level applications: A key contribution of HELIX is its incorporation of the Low and High PHY layer. In the context of O-RAN, this allows the RU to handle the PHY layer and connect to the DU at split 6. However, this presents a challenge as there are no standardized interfaces between this two units for this split, making integration more complex. HELIX addresses this challenge providing a simple fronthaul interface and library to facilitate integration within high-level applications. As mentioned in section 3.4, the `radio` parent class encapsulates all streaming and configuration functionalities. With some modifications, it could be integrated within an O-RAN implementation while replacing the legacy PHY layer. The DU can directly send and receive MAC layer user data and configure PHY layer parameters through a UDP socket over a 10GbE interface by solely using the library's provided functions. Complementarily, HELIX can be integrated into an O-RAN as part of the AAL, offloading computationally intensive task such as iFFT/FFT or LDPC encoding/decoding, as shown in Section 6.3.

Timing synchronization: Mobile network deployments require tight synchronization to coordinate between cells, enable time-sensitive network operations, and comply with 5G requirements. We plan to incorporate an external reference clock (e.g., GPS or Network Time Protocol (NTP)) to HELIX to enhance the platform's synchronization capabilities.

Control, broadcasting and random access channels: Currently, the testbed implements only the Physical Downlink Shared Channel (PDSCH) and Physical Uplink Shared Channel (PUSCH), while other physical channels such as broadcasting, control, and random access are not yet fully supported. Future work will focus on incorporating these missing components to enhance the platform's capabilities and enable more standard-compliant features.

Integration with the O-RAN Intelligent Controller (RIC): the RIC is a software-defined component that enables real-time network monitoring and optimization of the RAN. It relies on E2 interfaces to communicate with other network components. Although HELIX does not implement any E2 agent, its flexible interface together with real-time side-information from the PHY layer can be the foundation for future integration. HELIX's simple and fast interface provides real-time PHY layer parameters and enables closed-loop operations to control the operation of the network, e.g., for adaptive resource allocation or beam training.

9 CONCLUSIONS AND FUTURE WORK

In this work, we presented HELIX, a wireless 6G-ready experimentation platform that fills the gap between high bandwidth and real-time features. The platform integrates dynamic functional splitting, reconfigurable numerologies to operate in multiple bands, seamless integration with high-level applications and scalability. We showcased the capabilities of the platform through comprehensive evaluation in different bands, demonstrating end-to-end latencies as low as 500 μs and >1 Gbps data rates, alongside advanced wireless experimentation such as ISAC.

For future work, we plan to extend the capabilities of the system by integrating it with O-RAN software such as srsRAN or OAI, addressing the synchronization between the RU and the network. In addition, we intend to explore even higher bandwidth configurations by synchronizing multiple FPGAs, and by expanding to further 6G applications.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon Europe research and innovation programme under the SNS-JU through Grant Agreement No. 101192521 (MultiX) and under the Marie Skłodowska-Curie Actions (UNITE), Grant Agreement No. 101129618. It has also been supported by the Comunidad de Madrid through the DISCO6G-CM project (TEC-2024/COM-360) and TUCAN6-CM (TEC-2024/COM-460), funded under ORDEN 5696/2024. This work has also been funded by project PID2022-136769NB-I00 (ELSA) funded by MCIN/AEI /10.13039/501100011033 / FEDER, UE.

REFERENCES

[1] 3rd Generation Partnership Project (3GPP). 2023. *NR; Physical channels and modulation.* Technical Report TS 38.211. Technical Specification Group Radio Access Network. Available at: https://www.3gpp.org/ftp/.

[2] O-RAN Alliance. 2021. Control, User and Synchronization Plane Specification. O-RAN-WG4.CUS.0-v02. Working Group 4.

[3] Amarisoft. 2024. gNodeB Technical Specification. https://www.amarisoft.com/public-and-private-networks/solutions/ Accessed: 2024-12-09.

[4] AMD. 2021. Zynq UltraScale+ RFSoC ZCU216 Evaluation Kit. https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/zcu216.html Accessed: 2025-03-28.

[5] Fatih Aslan, George Iosifidis, Jose A. Ayala-Romero, Andres Garcia-Saavedra, and Xavier Costa-Perez. 2024. Fair Resource Allocation in Virtualized O-RAN Platforms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 17:1–17:34. https://doi.org/10.1145/3639043

[6] James Bishop, Jean-Marc Chareau, and Fausto Bonavitacola. 2018. Implementing 5G NR Features in FPGA. In *2018 European Conference on Networks and Communications (EuCNC)*. 373–9. https://doi.org/10.1109/EuCNC.2018.8443214

[7] Justine Cris Borromeo, Koteswararao Kondepu, Nicola Andriolli, and Luca Valcarenghi. 2022. FPGA-accelerated SmartNIC for supporting 5G virtualized Radio Access Network. *Computer Networks* 210 (2022), 108931. https://doi.org/10.1016/j.comnet.2022.108931

[8] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu. 2005. Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications* 53, 8 (2005), 1288–1299. https://doi.org/10.1109/TCOMM.2005.852852

[9] 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. https://doi.org/10.1109/ACCESS.2024.3447884

[10] Alex Forencich. [n. d.]. Verilog Ethernet. https://github.com/alexforencich/verilog-ethernet

[11] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton, Pablo Serrano, Cristina Cano, and Doug J. Leith. 2016. SrsLTE: An Open-Source Platform for LTE Evolution and Experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization* (New York City, New York) *(WiNTECH '16)*. Association for Computing Machinery, New York, NY, USA, 25–32. https://doi.org/10.1145/2980159.2980163

[12] Khandaker Foysal Haque, Francesca Meneghello, K M Rumman, and Francesco Restuccia. 2024. m3MIMO: An 8×8 mmWave Multi-User MIMO Testbed for Wireless Research. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking* (Washington D.C., DC, USA) *(ACM MobiCom '24)*. Association for Computing Machinery, New York, NY, USA, 1922–1929. https://doi.org/10.1145/3636534.3697321

[13] Xilinx Inc. 2024. SD-FEC: Soft Decision Forward Error Correction. https://www.xilinx.com/products/intellectual-property/sd-fec.html. Accessed: 2024-12-07.

[14] Ish Kumar Jain, Raghav Subbaraman, Tejas Harekrishna Sadarahalli, Xiangwei Shao, Hou-Wei Lin, and Dinesh Bharadia. 2020. MMobile: Building a MmWave Testbed to Evaluate and Address Mobility Effects. In *Proceedings of the 4th ACM Workshop on Millimeter-Wave Networks and Sensing Systems* (London, United Kingdom) *(mmNets'20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3412060.3418433

[15] Florian Kaltenberger, Aloizio P. Silva, Abhimanyu Gosain, Luhan Wang, and Tien-Thinh Nguyen. 2020. OpenAirInterface: Democratizing innovation in the 5G Era. *Computer Networks* 176 (2020), 107284. https://doi.org/10.1016/j.comnet.2020.107284

[16] Anupa Kelkar and Chris Dick. 2021. NVIDIA Aerial GPU Hosted AI-on-5G. In *2021 IEEE 4th 5G World Forum (5GWF)*. 64–69. https://doi.org/10.1109/5GWF52925.2021.00019

[17] Jesus Omar Lacruz, Dolores Garcia, Pablo Jiménez Mateo, Joan Palacios, and Joerg Widmer. 2020. Mm-FLEX: An Open Platform for Millimeter-Wave Mobile Full-Bandwidth Experimentation. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services* (Toronto, Ontario, Canada) *(MobiSys '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3386901.3389034

[18] Jesus O. Lacruz, Rafael Ruiz Ortiz, and Joerg Widmer. 2021. A Real-Time Experimentation Platform for Sub-6 GHz and Millimeter-Wave MIMO Systems. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (Virtual Event, Wisconsin) *(MobiSys '21)*. Association for Computing Machinery, New York, NY, USA, 427–439. https://doi.org/10.1145/3458864.3466868

[19] Line M. P. Larsen, Aleksandra Checko, and Henrik L. Christiansen. 2019. A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 146–172. https://doi.org/10.1109/COMST.2018.2868805

[20] Fan Liu, Yuanhao Cui, Christos Masouros, Jie Xu, Tony Xiao Han, Yonina C. Eldar, and Stefano Buzzi. 2022. Integrated Sensing and Communications: Toward Dual-Functional Wireless Networks for 6G and Beyond. *IEEE Journal on Selected Areas in Communications* 40, 6 (2022), 1728–1767. https://doi.org/10.1109/JSAC.2022.3156632

[21] Fan Liu, Christos Masouros, Athina P. Petropulu, Hugh Griffiths, and Lajos Hanzo. 2020. Joint Radar and Communication Design: Applications, State-of-the-Art, and the Road Ahead. *IEEE Transactions on Communications* 68, 6 (2020), 3834–3862. https://doi.org/10.1109/TCOMM.2020.2973976

[22] Silicom Ltd. [n. d.]. ACC 100 FEC Accelerator. https://www.silicom-usa.com/wp-content/uploads/2023/09/Lisbon-P2-ACC100-FEC-Accelerator-Extended-temp-1v1.pdf

[23] Juquan Mao, Mahmoud Alfa Abdullahi, Pei Xiao, and Aijun Cao. 2016. A low complexity 256QAM soft demapper for 5G mobile system. In *2016 European Conference on Networks and Communications (EuCNC)*. 16–21. https://doi.org/10.1109/EuCNC.2016.7560996

[24] MathWorks. [n. d.]. NR HDL Cell Search and MIB Recovery Reference Application. https://www.mathworks.com/help/wireless-hdl/ug/nr-hdl-cell-search.html. Accessed: 2024-11-22.

[25] MathWorks. 2024. nrChannelEstimate. https://es.mathworks.com/help/5g/ref/nrchannelestimate.html. Accessed: 2024-12-09.

[26] 3GPP Technical Specification Group Radio Access Network. 2020. *NR; Physical channels and modulation.* Technical Report 38.212. 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/DynaReport/38212.htm

[27] Benjamin Nuss, Patrick Groeschel, Johannes Pfau, Juergen Becker, Martin Vossiek, and Thomas Zwick. 2022. Broadband MIMO Testbed for the Development and Research on 6G. In *27th European Wireless Conference*.

[28] Benetel Olli Andersson. 2021. Functional Splits: The Foundation of an Open 5G RAN. *5G Technology World* (May 2021). https://www.5gtechnologyworld.com/functional-splits-the-foundation-of-an-open-5g-ran/ Accessed: 2024-12-07.

[29] Jacopo Pegoraro, Jesus O. Lacruz, Tommy Azzino, Marco Mezzavilla, Michele Rossi, Joerg Widmer, and Sundeep Rangan. 2024. JUMP: Joint Communication and Sensing With Unsynchronized Transceivers Made Practical. *IEEE Transactions on Wireless Communications* 23, 8 (2024), 9759–9775. https://doi.org/10.1109/TWC.2024.3365853

[30] Zhenzhou Qi, Chung-Hsuan Tung, Anuj Kalia, and Tingjun Chen. 2024. Savannah: Efficient mmWave Baseband Processing with Minimal and Heterogeneous Resources. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking* (Washington D.C., DC, USA) *(ACM MobiCom '24)*. Association for Computing Machinery, New York, NY, USA, 1500–1514. https://doi.org/10.1145/3636534.3690707

[31] S. Rangan. 2024. Wireless Communications Repository. https://github.com/sdrangan/wirelesscomm Accessed: 2024-11-22.

[32] Ettus Research. 2023. *USRP Hardware Driver (UHD).* National Instruments. Accessed: 2023-12-08.

[33] Sivers Semiconductors. 2024. Evaluation Kit EVK02001. https://www.sivers-semiconductors.com/5g-millimeter-wave-mmwave-and-satcom/wireless-products/evaluation-kits/evaluation-kit-evk02001/ Accessed: 2024-12-09.

[34] Sivers Semiconductors. 2024. Evaluation Kit EVK06006. https://www.sivers-semiconductors.com/5g-millimeter-wave-mmwave-and-satcom/wireless-products/evaluation-kits/evaluation-kit-evk06002/ Accessed: 2024-12-09.

[35] Akram Shafie, Nan Yang, Chong Han, Josep Miquel Jornet, Markku Juntti, and Thomas Kürner. 2023. Terahertz Communications for 6G and Beyond Wireless Networks: Challenges, Key Advancements, and Opportunities. *IEEE Network* 37, 3 (2023), 162–169. https://doi.org/10.1109/MNET.118.2200057

[36] Changyang She, Cunhua Pan, Trung Q. Duong, Tony Q. S. Quek, Robert Schober, Meryem Simsek, and Peiying Zhu. 2023. Guest Editorial xURLLC in 6G: Next Generation Ultra-Reliable and Low-Latency Communications. *IEEE Journal on Selected Areas in Communications* 41, 7 (2023), 1963–1968. https://doi.org/10.1109/JSAC.2023.3282543

[37] Christian Sturm and Werner Wiesbeck. 2011. Waveform Design and Signal Processing Aspects for Fusion of Wireless Communications and Radar Sensing. *Proc. IEEE* 99, 7 (2011), 1236–1259. https://doi.org/10.1109/JPROC.2011.2131110

[38] Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. 2007. Kernel Regression for Image Processing and Reconstruction. *IEEE Transactions on Image Processing* 16, 2 (2007), 349–366. https://doi.org/10.1109/TIP.2006.888330

[39] J. Valls, T. Sansaloni, A. Perez-Pascual, V. Torres, and V. Almenar. 2006. The use of CORDIC in software defined radios: a tutorial. *IEEE Communications Magazine* 44, 9 (2006), 46–50. https://doi.org/10.1109/MCOM.2006.1705978

[40] Cheng-Xiang Wang, Xiaohu You, Xiqi Gao, Xiuming Zhu, Zixin Li, Chuan Zhang, Haiming Wang, Yongming Huang, Yunfei Chen, Harald Haas, John S. Thompson, Erik G. Larsson, Marco Di Renzo, Wen Tong, Peiying Zhu, Xuemin Shen, H. Vincent Poor, and Lajos Hanzo. 2023. On the Road to 6G: Visions, Requirements, Key Technologies, and Testbeds. *IEEE Communications Surveys & Tutorials* 25, 2 (2023), 905–974. https://doi.org/10.1109/COMST.2023.3249835

[41] Kang Wang, Xi Yang, Xiao Li, Chao-Kai Went, and Shi Jin. 2019. SDR Implementation of an End-to-End mmWave Testbed Based on Phased Antenna Array. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*. 1–6. https://doi.org/10.1109/WCSP.2019.8928067

[42] Zhongxiang Wei, Fan Liu, Christos Masouros, Nanchi Su, and Athina P. Petropulu. 2022. Toward Multi-Functional 6G Wireless Networks: Integrating Sensing, Communication, and Security. *IEEE Communications Magazine* 60, 4 (2022), 65–71. https://doi.org/10.1109/MCOM.002.2100972

[43] Kai Wu, Jacopo Pegoraro, Francesca Meneghello, J. Andrew Zhang, Jesus O. Lacruz, Joerg Widmer, Francesco Restuccia, Michele Rossi, Xiaojing Huang, Daqing Zhang, Giuseppe Caire, and Y. Jay Guo. 2024. Sensing in Bistatic ISAC Systems With Clock Asynchronism: A signal processing perspective [Special Issue on Signal Processing for the Integrated Sensing and Communications Revolution]. *IEEE Signal Processing Magazine* 41, 5 (2024), 31–43. https://doi.org/10.1109/MSP.2024.3418725

[44] Xilinx. 2024. AXI4-Stream Interconnect Product Page. https://www.xilinx.com/products/intellectual-property/axi4-stream_interconnect.html. Accessed: 2024-12-08.

[45] Xilinx. 2024. RFSoC ZCU111. https://www.xilinx.com/products/boards-and-kits/zcu111.html Accessed: 2024-12-09.

[46] Xilinx. 2024. RFSoC ZCU208. https://www.xilinx.com/products/boards-and-kits/zcu208.html Accessed: 2024-12-09.

[47] Ali Zaidi, Fredrik Athley, Jonas Medbo, Ulf Gustavsson, Giuseppe Durisi, and Xiaoming Chen. 2018. *5G Physical Layer: Principles, Models and Technology Components*. Academic Press. https://doi.org/10.1016/C2016-0-04818-0

[48] Zhengquan Zhang, Yue Xiao, Zheng Ma, Ming Xiao, Zhiguo Ding, Xianfu Lei, George K. Karagiannidis, and Pingzhi Fan. 2019. 6G Wireless Networks: Vision, Requirements, Architecture, and Key Technologies. *IEEE Vehicular Technology Magazine* 14, 3 (2019), 28–41. https://doi.org/10.1109/MVT.2019.2921208