SPEAR: Software-defined Python-Enhanced RFSoC for Wideband Radio Applications

Wei Cheng, Zhihui Gao, Tingjun Chen
Department of Electrical and Computer Engineering, Duke University

ABSTRACT

Next-generation wireless systems utilize large signal bandwidths to meet the growing data rate demands of emerging applications and to provide enhanced resolution for wireless sensing and imaging. This poses significant challenges in the design of the underlying datapaths that carry and transfer signals across different domains, such as between memory and data converters in various software-defined radio (SDR) platforms. In this paper, we present the design and implementation of SPEAR, which is an SDR platform based on the Xilinx RFSoC ZCU216 evaluation board capable of supporting real-time streaming of signals with a bandwidth of up to 1.25 GHz employing the direct RF radio architecture. SPEAR features hardware-assisted direct memory access (DMA) control for real-time data streaming, and a Python-based hardware configuration tool and signal processing framework. Our experiments show that SPEAR can support a real-time bandwidth of up to 1.25 GHz for 256QAM modulation that satisfies the 3GPP error vector magnitude (EVM) requirement of 3.5%.

CCS CONCEPTS

• **Networks** → **Network experimentation**; • **Computer systems organization** → **Real-time system architecture**.

KEYWORDS

Software-defined radio; FPGA; RFSoC; hardware-software co-design; testbed

ACM Reference Format:
Wei Cheng, Zhihui Gao, Tingjun Chen. 2024. **SPEAR**: Software-defined Python-Enhanced RFSoC for Wideband Radio Applications. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3636534.3697310

ACM MobiCom '24, November 18–22, 2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/11…$15.00
https://doi.org/10.1145/3636534.3697310

1 INTRODUCTION

Next-generation wireless systems utilize large signal bandwidths to meet the growing data rate demands of emerging applications and to provide enhanced resolution for wireless sensing and imaging [17, 21, 22]. For example, 5G new radio (NR) supports channel bandwidth of up to 400 MHz in frequency range 2 (FR2) [3], and IEEE 802.11ad supports a bandwidth of up to 2.16 GHz in the 60 GHz unlicensed band [1]. However, supporting such high data rates poses significant challenges in the design of the underlying datapaths that carry and transfer signals across different domains, such as between memory and digital-to-analog converters (DACs) as well as analog-to-digital converters (ADCs), on a variety of software-defined radio (SDR) platforms.

In recent years, Radio Frequency System-on-Chip (RFSoC) has emerged as a promising solution to address these challenges by providing built-in multi-GHz sampling rates RF data converters (RF-DC) and powerful FPGA fabrics for datapath routing. RFSoC platforms have been used across various domains to build high-performance SDRs and beamforming systems [15, 26–28], MIMO radios and radars [13, 16, 18, 23, 24], as well as quantum control [14, 25] and analog computing systems. These works customized the hardware and/or software design for specific applications, yet there remains potential for improving system performance and further investigating the trade-offs between real-time data streaming support and ease of programmability.

In this paper, we present the design and implementation of **SPEAR** (**S**oftware-defined **P**ython-**E**nh**A**nced **R**FSoC), an SDR platform based on the Xilinx RFSoC ZCU216 evaluation board [12]. SPEAR supports real-time streaming of signals with a bandwidth of up to 1.25 GHz employing the direct RF architecture. The design of SPEAR includes two key components: (*i*) a real-time, streaming-based hardware design for the transmitter (TX) and receiver (RX) datapaths, and (*ii*) a Python interface that enables users to easily configure the RFSoC hardware for wideband radio applications while being able to experiment with customized waveforms and digital signal processing (DSP) algorithms using the Python programming language. This design effectively decouples the

Table 1: Comparison of previous RFSoC-based SDR system designs and SPEAR.

	mmWaveSDR [23]	MIMORPH [18]	qsfp28_offload [24]	SPEAR (This Work)
Hardware Platform	RFSoC 2×2	ZCU111	RFSoC 4×2	ZCU216
Radio Architecture	Zero-IF	Zero-IF/Direct RF	Direct RF	Direct RF
TX Dataflow	PS DRAM→DAC	BRAM→DAC	PS DRAM→DAC	PS DRAM→DAC
TX DMA Type	Xilinx DMA	Xilinx DMA	Xilinx DMA	Custom IP
TX DMA Control	CPU busy waiting	CPU busy waiting	HW state machine	HW state machine
RX Dataflow	ADC→PS DRAM	ADC→PL DRAM	ADC→CMAC	ADC→PL DRAM
RX DMA Type	Xilinx DMA	Custom IP	N/A	Custom IP
RX DMA Control	CPU busy waiting	HW state machine	N/A	HW state machine
Real-Time Streaming	No	Yes	Yes	Yes
Bandwidth	1.536 GHz	1.76 GHz	2.457 GHz	1.25 GHz
Preamble Detection	Yes (SW/HW)	Yes (HW)	No	Yes (SW)
Python Interface	Yes (control only)	No	Yes (control only)	Yes (control and DSP)

software-based control of direct memory access (DMA) for streaming and offloads it into the hardware, therefore ensuring robust, real-time streaming capabilities while alleviating the CPU's workload and freeing it up for other tasks.

We evaluate the performance of SPEAR using two setups in terms of the supported real-time bandwidth and link quality of transmission (QoT). **RFSoC-USRP**, which utilizes a ZCU216 boards as the TX and a USRP X310 SDR as the RX, and **RFSoC-RFSoC**, which utilizes two ZCU216 boards as the TX and RX. We consider link QoT metrics including the signal-to-noise ratio (SNR), error vector magnitude (EVM), and bit error rate (BER). We explore different system configurations with varying modulation schemes from 16QAM to 1024QAM, signal bandwidth from 100 MHz to 1.25 GHz, and tunable carrier frequency within the 1st Nyquist zone. The experimental results show that SPEAR can support a real-time bandwidth of up to 1.25 GHz for 256QAM modulation that satisfies the 3GPP EVM requirement of 3.5% [3]. Overall, SPEAR provides a versatile and flexible platform for users interested in operating real-time RF communication systems with GHz+ bandwidth. Both the software and hardware design of SPEAR, and its supported example experiments, are open-sourced in [9].

2 RELATED WORK

Existing RFSoC-based SDR platforms, including Xilinx's reference design [6], mmWaveSDR [23], MIMORPH [18], and qsfp28_offload [24], have shown that creating a wideband SDR platform necessitates specific hardware-software (HW-SW) co-design strategies and high-speed datapath designs to support the real-time streaming of signals with large bandwidth. In this section, we provide an overview of existing RFSoC-based SDR systems including the hardware design and software architecture, as summarized in Table 1.

mmWaveSDR [23]. A mmWave SDR platform is proposed in this work by connecting an RFSoC 2×2 board [8] with the Sivers EVK06002 60 GHz phased array transceiver modules. mmWaveSDR incorporates a waveform-triggered reception hardware IP for hardware-based preamble detection and transfers the detected signal frames to a host computer for offline baseband processing. Software-triggered reception is also available by capturing a fixed number of I/Q samples for post-processing. Both the software- and hardware-triggered methods in this work are intended to bridge the processing speed difference between ADCs and the host computer. However, the TX/RX datapath design of mmWaveSDR offers slower data rates compared to what the DAC/ADC datapath is capable of. This implies that mmWaveSDR cannot operate in real-time. Further analysis of the design requirements based on the datapath data rate and DAC/ADC sampling rates is provided in §3.1.

MIMORPH [18]. MIMORPH is a real-time reconfigurable SDR platform supporting 60 GHz 4×4 MIMO with a bandwidth of 1.536 GHz and sub-6 GHz 4×4 MIMO with a bandwidth of 160 MHz. It also supports closed-loop configuration, where the devices switch between TX and RX modes and can react to the received packets. MIMORPH's TX design leverages loopback memory (LBM) blocks, which is block RAM (BRAM) working as circular buffers to continuously transmit samples. However, the number of I/Q samples that can be streamed continuously is limited due to the size of the FIFO and the fact that only a limited number of FIFOs can be synthesized on an FPGA. On the RX side, MIMORPH implemented highly optimized IPs and drivers to support continuously writing the received samples into the PL DRAM.

qsfp28_offload [24]. qsfp28_offload demonstrated that it is possible to offload a full 2.457 GHz bandwidth RF signal captured by a direct RF sampling RX using an RFSoC 4×2 board to a host machine, which can then perform the FFT operation using a GPU to obtain the spectrum density and occupancy information. Furthermore, hardware UDP IP is added so that the signal offloading process can be done across

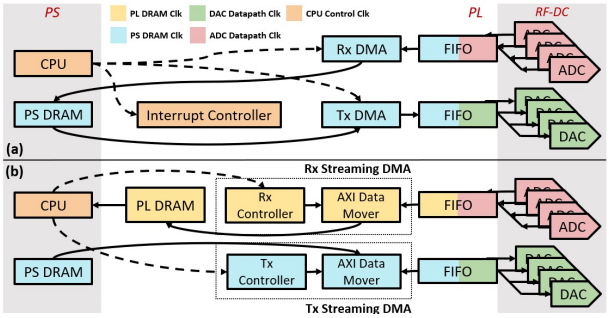


Figure 1: Hardware architecture of (a) reference design from Xilinx, and (b) SPEAR. Note that both CPU and PS DRAM belong to the Processing System (PS) while the other components belong to the Processing Logic (PL).

the Internet by continuously streaming UDP packets through the 100 GbE transceiver on the board. On the TX side, the signal generator module allows generating waveforms that are pre-defined in the PS DRAM to be transferred repeatedly to DAC using DMA configured in the cyclic mode. This DMA configuration allows the hardware to go through a list of user-defined memory segments and move data within those scattered memory regions to the DAC. qsfp28_offload generates waveform using the ARM CPU cores and streams the received I/Q samples to the host machine running the GNU Radio gr-fosphor module, which visualizes the real-time spectrum using GPU acceleration.

Our design. The comparison between existing RFSoC-based SDR designs and our proposed design, SPEAR, is detailed in Table 1. In this work, we study various design choices including: (*i*) TX/RX dataflow; (*ii*) TX/RX DMA type and control; (*iii*) maximum supported bandwidth with real-time data streaming; and (*iv*) other system supports such as packet detection, DSP, and the control interface. To the best of our knowledge, SPEAR is the first design that supports real-time data streaming of signal with a bandwidth up to 1.25 GHz leveraging *hardware-assisted data streaming*. It allows user-friendly hardware configuration and highly customizable DSP workflows, both via a Python interface.

3 IMPLEMENTATION OF SPEAR

In this section, we first provide an overview of Xilinx's RFSoC reference design and its limitations, particularly on the software-based DMA control. Then, we present the design of SPEAR including its hardware and software components.

3.1 RFSoC Reference Design

Hardware design. Xilinx's reference hardware design is shown in Fig. 1(a), which comprises a quad-core ARM CPU (Processing System, PS), PS DRAM, FPGA fabric (Processing Logic, PL), FIFOs on the TX and RX datapaths, Direct Memory Access (DMA) IP that transfers data between the DAC/ADC and DRAM, an interrupt controller that notifies the CPU once a DMA transaction is completed, and RF Data Converters (RF-DC) that contains DACs/ADCs. This reference design supports data transfer (e.g., streaming of I/Q samples) between the PS DRAM and DACs/ADCs, allowing users to generate baseband waveforms and process captured signals for different SDR applications.

In RFSoC-based SDRs, clock domain management is a critical aspect of the hardware design, since the DACs/ADCs usually operate at multi-GHz sampling rates while their datapaths operate at only 100s of MHz. To ensure consistent data rates across clock domains, the Super Sample Rate (SSR) circuits is used. In particular, the required datapath clock frequency for a specific sampling rate can be determined by

$$f_{\text{samp}} = L \cdot K \cdot f_{\text{clk, datapath}} \cdot \text{mode} \tag{1}$$

where f_{samp} denotes the sampling rate of the DAC/ADC with an interpolation/decimation factor of L, $f_{\text{clk, datapath}}$ denotes the clock frequency for the DAC/ADC datapath, K denotes the number of samples (I and Q) handled per clock cycle, and mode = 1 and 1/2 for the real-to-real and I/Q-to-real mode, respectively. For example, a real-time 2.5 GSaps DAC with 2× interpolation requires a datapath with 4 I/Q samples per cycle to operate at 312.5 MHz in the I/Q-to-real mode.

When operating an RFSoC in the direct RF radio architecture, the effective I/Q sampling rate, or the supported signal bandwidth, of an ADC operating in the real-to-I/Q mode is equal to the raw sampling rate of the ADC divided by the decimation factor. For instance, a 2.5 GSaps ADC with 2× decimation factor generates a 1.25 GSaps data stream for each of the I and Q channels, where each channel captures 0.625 GHz signals bandwidth. This also applies to the calculation of the I/Q sampling rate on the TX side using a DAC with an interpolation factor.

To ensure real-time data streaming, the hardware design needs to make sure samples can be pulled from memory fast enough so that the datapath is never "starved". For example, assuming a TX DMA is pulling I/Q samples from the PS DRAM at clock speed of 333 MHz with 4 I/Q samples per clock cycle and 2× interpolation factor, the supported DAC sampling rate can go up to a maximum of 2.66 GSaps ($2 \times 8 \times 333\,\text{MHz} \times \frac{1}{2}$) according to (1). This is equivalent to a data transfer speed of 42.6 Gbps from the DRAM with 16-bit I/Q samples, and this configuration shall be sufficient for real-time streaming of I/Q samples from the PS DRAM to the DAC operating at a sampling rate of 2.5 GSaps.

Software architecture. Xilinx provides both Python- and C-based APIs to facilitate user interaction with FPGA hardware [10]. While C-based APIs are widely used in embedded devices, Python-based APIs provide users easier access to a broad range of versatile libraries for arithmetic operations

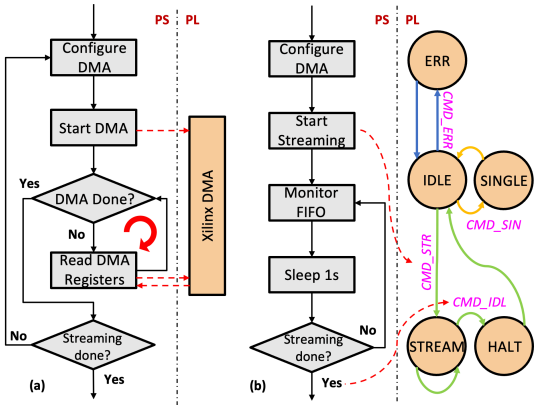


Figure 2: The software execution flow of DMA control with (a) CPU busy waiting (reference design), and (b) hardware-assisted streaming (SPEAR). The black dotted line indicates a separation between PS and PL while the red dotted line represents the interaction between software running on PS and hardware state machines implemented on PL.

and signal processing, and therefore, can significantly reduce development time and smoothen the learning curve for users. However, compared to C-based APIs, Python-based APIs usually suffer from a higher control latency with larger variation. For example, our measurements using the ZCU216 board [12] show that writing to a memory-mapped address to trigger a DMA transfer using the Python-based PYNQ [11] DMA driver incurs msec-level delay.

To optimize the trade-offs between adopting a Python API and ensuring system performance, we first identify the software execution flow that is used to control Xilinx's DMA. The combination of Xilinx DMA and CPU busy waiting software control follows the execution flow as shown in Fig. 2(a), where a nested control loop is involved. In particular, the CPU constantly reads DMA registers to check whether a DMA transfer has been completed as indicated by the red circle. This busy waiting process consumes significant CPU time to achieve real-time performance.

3.2 Design of SPEAR

Fig. 1(b) depicts the dataflow of our hardware design, where the generated I/Q samples on the TX side are streamed from the PS-DRAM to the DAC(s), and the captured I/Q samples on the RX side are streamed from the ADC(s) to the PL DRAM. This asymmetric dataflow design prevents resource contention that may occur when reading and writing I/Q samples to the same piece of memory.[1] In particular, the TX datapath employs a streaming DMA consisting of a customized TX controller and Xilinx's AXI Data Mover IP [20].

[1]Either PS or PL DRAM works for streaming data to/from the TX/RX datapath. The dataflow of our design achieves the best performance when streaming data from the PS DRAM to DAC(s) and ADC(s) to PL DRAM.

As mentioned in §3.1, the PS DRAM clock domain must operate at a higher clock speed than the DAC datapath clock domain to ensure real-time data transfer. Similarly, the RX datapath employs a RX streaming DMA consisting of a customized RX controller and Xilinx's AXI Data Mover IP. The requirement for real-time data transfer meionted above also applies to the RX datapath. In addition to the TX and RX datapaths, the CPU control clock domain operates at a frequency of 100 MHz, connecting the CPU with other components via an AXI interface [19].

Streaming DMA based on hardware FSM. The Streaming DMA is the most important hardware component that distinguishes our design from previous works. It includes a custom controller and the AXI Data Mover IP [20] provided by Xilinx. The AXI Data Mover processes DMA requests via an AXI-Stream port, facilitating data transfer between an AXI port connecting to memory and another AXI-Stream port connecting to a DAC/ADC.

Our customized Streaming DMA is used in both the TX and RX datapaths, as illustrated in Fig. 1(b). The customized controller IP within the streaming DMA functions as a finite state machine (FSM), whose transition diagram is shown in Fig. 2(b). Initially, the system remains in the IDLE state and transitions to the corresponding state when a command is issued. Transitioning from the IDLE state to SINGLE state via a CMD_SIN command indicates that a single DMA transfer will occur by initiating a single request to the AXI Data Mover IP [20]. This request specifies the number of samples to be moved to/from their corresponding memory address.

The transition from the IDLE state to STREAMING state is triggered by a CMD_STR command, indicating streaming DMA requests to the AXI Data Mover and initiates continuous DMA transfers. Based on the analysis in §3.1 and (1), our streaming DMA design is capable of moving samples from/to the PS/PL DRAM at 42.6 Gbps. The controller will exit the STREAMING state only if it receives a CMD_IDL command to transition back to the IDLE state. When a command is invalid, the FSM will transition to the ERROR state. As highlighted in [20], it is important to gracefully terminate the streaming process. We observed that the AXI Data Mover continues to fetch samples from memory even after that memory segment has been deallocated by the operating system, which can result in arbitrary data being streamed to/from the DAC/ADC and may lead to unstable outputs. To prevent this, the user must send an CMD_IDLE command to transition the controller to a HALT state, effectively terminating ongoing data transfers. Once the AXI Data Mover is completely halted, the controller will transition back to IDLE state. Fig. 3 illustrates the detailed signaling corresponding to the transition between the IDLE, STREAM, HALT, and SINGLE states.

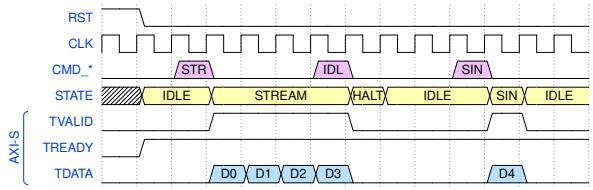


Figure 3: The waveform-level behavior of the TX/RX controller as part of the designed Streaming DMA.

Software architecture and DMA control method. For both TX and RX datapaths, it is important to have a DMA that can efficiently and repeatedly transfer samples between memory (PS and/or PL DRAM) and the DACs/ADCs in real-time. The different DMA types and control methods compared to previous RFSoC-based SDR platforms are summarized in Table 1. In contrast to the DMA control employed by previous works as shown in Fig. 2(a), SPEAR offloads the inner control loop to the hardware FSM, as shown in Fig. 2(b). In our design, CPU is only responsible for the FIFO monitoring task executed at 1-second intervals, which can be further relaxed if FIFO monitoring is required at a coarser granularity. Here, the FIFO monitoring task is created to ensure real-time performance by periodically checking the number of remaining samples in the TX and RX FIFOs. If the TX FIFO is never empty and RX FIFO is never full then real-time streaming is guaranteed. Leveraging this Streaming DMA, SPEAR relieves CPU from busy waiting so that CPU time can be repurposed to other tasks such as DSP as well as controlling and interfacing with other peripheral devices.

3.3 DSP pipeline with an OFDM PHY

We also develop a Python-based DSP pipeline that employs an OFDM-based physical layer, which can be used to generate customized I/Q waveforms as well as process and analyze received I/Q waveforms. The OFDM-based PHY layer employs an FFT size of 64 (which can be customized), and leverages fast Fourier transform (FFT) and its inverse (IFFT) to convert frequency-domain data symbols to the time-domain I/Q waveforms. We consider four modulation schemes including 16QAM, 64QAM, 256QAM, and 1024QAM. The channel state information (CSI) between the TX and RX can be estimated using predefined pilot symbols, which is then used for the equalization and demodulation of the received signal. Note that since the RFSoC boards are configured to operate in the direct RF architecture, there is no carrier frequency offset (CFO) between the TX and RX due to the absence of an RF mixer driven by a local oscillator (LO). The Python-based DSP pipeline allows the calculation of the following metrics of the received signal to evaluate the link performance: SNR, EVM, and BER.

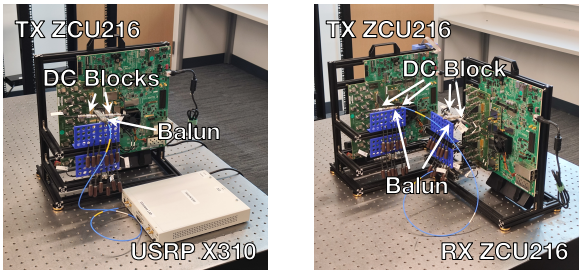


Figure 4: RFSoC-USRP setup used in Config 1 (*Left*), and RFSoC-RFSoC setup used in Config 2 and Config 3 (*Right*).

Table 2: Three configurations in our evaluation.

	Config 1	Config 2	Config 3
Transmitter (TX)	ZCU216	ZCU216	ZCU216
DAC Samp. Rate	4.0 GSaps	2.5 GSaps	2.5 GSaps
Interpolation	40×	8×	2×
Receiver (RX)	USRP X310	ZCU216	ZCU216
ADC Samp. Rate	0.2 GSaps	2.5 GSaps	2.5 GSaps
Decimation	2×	8×	2×
Link Bandwidth	100 MHz	312.5 MHz	1.25 GHz

4 EXPERIMENTS AND EVALUATIONS

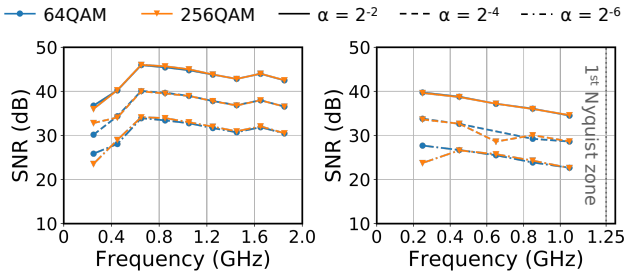
4.1 Experimental Setup

We evaluate the performance of SPEAR using two setups shown in Fig. 4 under three configurations (see Table 2):

- **Config 1: RFSoC-USRP (100 MHz)**, where the TX is an RFSoC ZCU216 board whose DAC runs at a sampling rate of 4.0 GSaps with 40× interpolation, and the RX is a USRP X310 SDR. This configuration enables 100 MHz link bandwidth where the TX carrier frequency can be swept between 0–2 GHz (the 1st Nyquist zone) by tuning the NCO;
- **Config 2: RFSoC-RFSoC (312.5 MHz)**, where two RFSoC ZCU216 boards are used as the TX and RX, whose DAC and ADC run at a sampling rate of 2.5 GSaps with 8× interpolation and decimation, respectively. This configuration enables a link with 312.5 MHz bandwidth, whose carrier frequency can be swept between 0–1.25 GHz (the 1st Nyquist zone) by tuning the NCO;
- **Config 3: RFSoC-RFSoC (1.25 GHz)**, where two RFSoC ZCU216 boards are used as the TX and RX, whose DAC and ADC run at a sampling rate of 2.5 GSaps with 2× interpolation and decimation, respectively. This configuration enables a link with 1.25 GHz bandwidth. Since this bandwidth occupies the entire the 1st Nyquist zone, the carrier frequency is fixed at 625 MHz.

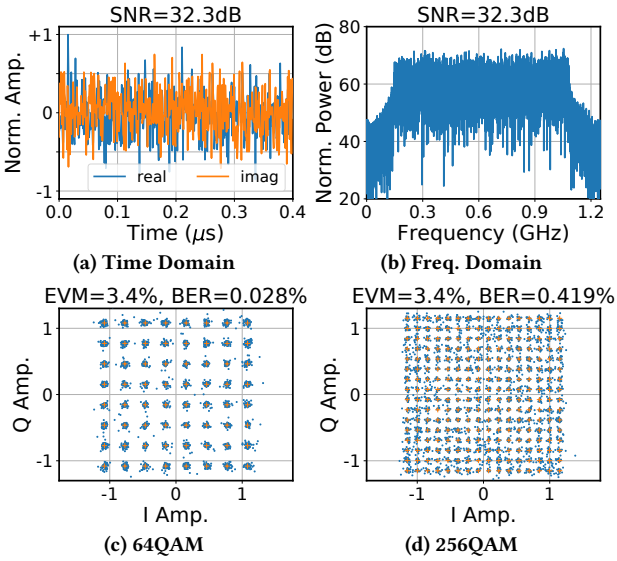
The inverse sinc filter [7] for the RFSoC DAC is activated to mitigate the roll-off at the edge of the 1st Nyquist zone.

To benchmark SPEAR's performance under each configuration, we establish a wired channel between the TX and RX using an SMA cable. A 0.2 MHz–6.0 GHz wideband Marki



(a) Config 1 (b) Config 2

Figure 5: Measured link SNR under varying scaling factors, α, and NCO settings within the 1st Nyquist zone for Config 1 (0–2.0 GHz) and Config 2 (0–1.25 GHz).



(a) Time Domain (b) Freq. Domain

(c) 64QAM (d) 256QAM

Figure 6: (a)–(b) Received I/Q waveform for 256QAM visualized in the time and frequency domains, and (c)–(d) Example constellation diagrams of 64QAM and 256QAM for the 1.25 GHz bandwidth link in Config 3.

BALH-0006 balun [4] is employed to convert the differential signal output from the RFSoC DAC to single-ended output. In addition, two BLK-89-S+ DC blocks [5] are applied on the differential ports of the balun. For the RFSoC to RFSoC connections, two Marki BALH-0006 baluns [4] with four BLK-89-S+ DC blocks are connected in a symmetric manner to handle the differential outputs of the DACs and ADCs.

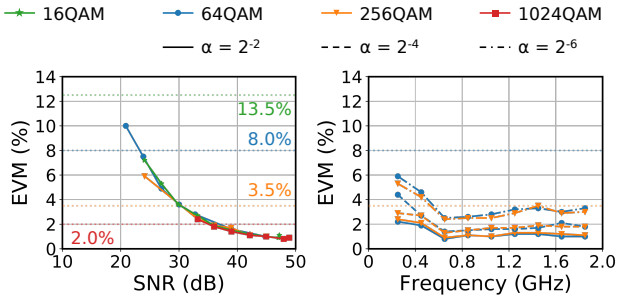
The link SNR is measured by taking the ratio of the received signal power to the RX noise level, and can be adjusted by scaling the amplitude of the TX baseband I/Q waveform by a factor of $\sqrt{\alpha} \in [0, 1]$. For example, under Config 3, a maximum TX power (with $\alpha = 1$) at the output of the DAC is measured as -10.5 dBm when the TX waveform is scaled to the DAC's maximum input level of 2^{13} (one extra bit is used to indicate the ± voltage of the 14-bit DAC), and the TX power can be reduced by 3 dB or 6 dB from the maximum power by setting $\alpha = 2^{-1}$ or 2^{-2}. Fig. 5 shows the measured link SNR values with different modulation schemes and scaling factors across the 1st Nyquist zone for **Config 1** and **Config 2**. The lower link SNR at the low part of the 1st Nyquist zone in **Config 1** is due to the frequency response of USRP UBX-160 daughterboard [2]. Such an effect is not observed for **Config 2**, however, the link SNR slightly degrades at high carrier frequencies closer to the boundary of the 1st Nyquist zone. For each experiment, we transmit and-collect 100 OFDM packets, from which the SNR, EVM, and BER results are obtained.

4.2 Verifying the Design of SPEAR

The FIFO monitoring task mentioned in §3 periodically checks the remaining elements in a FIFO to ensure real-time streaming at the circuit level. To further confirm this, we validate real-time TX streaming by streaming a pre-defined continuous-wave (CW) signal and verifying the continuity and integrity of the output signal from the DAC in both the time and frequency domains over a period of 30 minutes, using a high-speed oscilloscope and spectrum analyzer, respectively. To validate real-time RX streaming, we ensure that the hardware is capable of capturing a pre-defined waveform generated by the real-time TX without missing any samples.

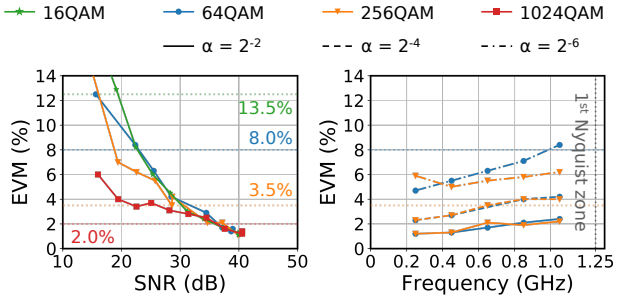
We also validate our Python-based DSP pipeline, adapted from [16], which can be used to generate OFDM-based PHY waveforms with customized parameters such as the modulation order. We configure the Streaming DMA on the TX of RFSoC to the "Streaming Mode", enabling real-time transmission of the waveform. The streaming DMA on the RX of RFSoC is initially set to the "Streaming Mode", and a FIFO monitoring task is executed to ensure real-time performance. It is then switched to the "Single Transfer Mode" to capture consecutive I/Q samples. These captured samples are processed using the Python-based DSP pipeline, including packet detection, equalization, demodulation, and decoding. Figs. 6(a) and 6(b) show the captured waveform with 256QAM modulation under **Config 3** in the time and frequency domains, respectively. Example demodulated constellation diagrams for 64QAM and 256QAM under the same configuration are shown in Figs. 6(c) and 6(d), respectively. In particular, with a link SNR of 32.3 dB, 64QAM achieves an EVM of 3.4% and a BER of 0.03%, and 256QAM achieves an EVM of 3.4% and a BER of 0.42%.

While the RX of RFSoC can capture and stream samples in real-time, the current Python-based DSP pipeline running on PS cannot process all the received packets at line rates due to limited CPU processing power. Future work includes migrating packet detection and lower PHY functions into the RFSoC PL as well as offloading the remaining DSP tasks to an edge server (for details see §5).



(a) With fixed NCO at 1.05 GHz (b) EVM for 64QAM and 256QAM

Figure 7: (a) Measured SNR versus EVM in at 1.05 GHz carrier frequency, and (b) measured EVM for 64QAM and 256QAM EVM with varying scaling factors under Config 1.



(a) With fixed NCO at 650 MHz (b) EVM for 64QAM and 256QAM

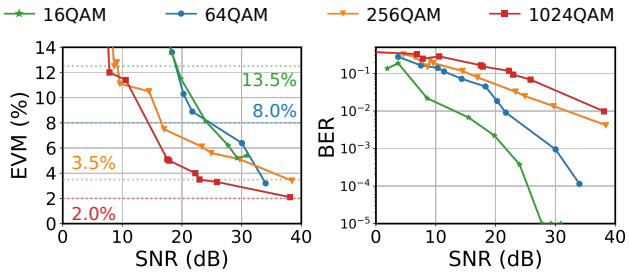
Figure 8: (a) Measured SNR versus EVM in at 650 MHz carrier frequency, and (b) measured EVM for 64QAM and 256QAM EVM with varying scaling factors under Config 2.

4.3 Results

We evaluate the link performance for each configuration in terms of the achieved EVM under varying link SNR values and NCO settings in the direct RF radio architecture.

RFSoC-USRP with 100 MHz bandwidth. For **Config 1**, we sweep the NCO frequency across the 1st Nyquist zone up to 2.0 GHz at 200 MHz step size, and Fig. 7(a) shows the EVM of the signals with different modulations received by the USRP, when the carrier frequency is set to be 1.05 GHz. We follow the EVM requirements of 12.5%, 8%, and 3.5% for 16QAM, 64QAM, 256QAM, respectively, as defined by 3GPP [3], and set the EVM requirement to be 2% for 1024QAM. These EVM thresholds are indicated by the horizontal dashed lines. The results show that the minimum required link SNR for 64QAM and 256QAM to satisfy the corresponding 3GPP EVM requirements is 23.9 dB and 33.0 dB, respectively. Fig. 7(b) shows the EVM across varying NCO settings for 64QAM and 256QAM with different scaling factors, α, corresponding to a link SNR range of 32.8-45.0 dB across all considered settings. Note that the worse EVM performance at smaller NCO values is due to the lower link SNR (see Fig. 5(a)).

RFSoC-RFSoC with 312.5 MHz bandwidth. Using a second ZCU216 board as the RX, this **Config 2** supports a larger signal bandwidth. We sweep the NCO frequency across the



(a) EVM vs. SNR (b) BER vs. SNR

Figure 9: Measured EVM-SNR and BER-SNR relationship with different modulation schemes under Config 3.

1st Nyquist zone up to 1.25 GHz at 200 MHz step size. Fig. 8(a) shows the EVM of the signals with different modulations received by the RX ZCU216, when the carrier frequency is set to be 650 MHz. The results show that the minimum required link SNR for 64QAM and 256QAM to satisfy the corresponding 3GPP EVM requirements is 25.5 dB and 28.6 dB, respectively. The EVM performance is slightly worse compared to **Config 1** at larger signal bandwidth. Fig. 8(b) shows the EVM across varying NCO settings for 64QAM and 256QAM with different scaling factors, α, corresponding to a link SNR range of 25.5-28.6 dB across all considered settings. Under the same value of α, the EVM performance slightly degrades larger NCO values due to the decreased link SNR closer to the boundary of the 1st Nyquist zone (see Fig. 5(b)).

RFSoC-RFSoC with 1.25 GHz bandwidth. We evaluate the performance of the RFSoC-RFSoC link with 1.25 GHz real-time bandwidth under **Config 3**. Fig. 9 plots the measured EVM and BER across varying link SNR values with different modulation schemes. While this link supports 16QAM, 64QAM, and 256QAM with the desired EVM performance, it is on the verge of meeting the EVM requirement for 1024QAM at the maximum supported SNR of 38.1 dB, where the corresponding EVM is 2.1%. This is due to the system nonlinearity as we saturate the DAC to increase the SNR without using an RF amplifier, and the frequency selectivity of the DAC/ADC across a large signal bandwidth that spans the entire 1st Nyquist zone.

5 LIMITATIONS AND FUTURE WORK

Currently, our platform utilizes a Python-based software for preamble detection and demodulation, which is constrained by the processing capabilities of the PS on the ZCU216 board. We plan to migrate the packet preamble detection algorithm to hardware and offload the demodulation process to a remote server with a DSP pipeline, in a virtualized radio access network (RAN) setting. In this proposed setup, only the detected symbols would be captured and transmitted over a high-speed link connecting the ZCU216 board and a host computer. In addition, FFT and CSI estimation operations consume a significant portion of time in the DSP pipeline;

therefore, we plan to migrate the FFT and other low-PHY operations to the FPGA fabrics. Finally, the ZCU216 [12] supports 16 DACs and 16 ADCs, and we plan to expand our system to a multi-channel configuration and explore the tradeoffs between efficient hardware-software co-design and the maximum supported bandwidth per channel. We plan to develop a subsystem that is capable of handling duty cycling to further relieve the pressure on the TX/RX datapath.

6 CONCLUSIONS

In this paper, we present the design and implementation of SPEAR and demonstrate its ability to support wideband radio applications. SPEAR consists of a hardware-assisted DMA control module and a Python-based interface for both hardware configuration and highly customizable DSP. We also presented comprehensive evaluations of SPEAR using RFSoC-USRP and RFSoC-RFSoC setups under different configurations, and showed that it can support a real-time channel bandwidth of up to 1.25 GHz for 256QAM modulation that satisfies the 3GPP EVM requirement of 3.5%.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-2128638, CNS-2211944, and AST-2232458, and the Center for Ubiquitous Connectivity (CUbiC), sponsored by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program.

REFERENCES

[1] 2012. IEEE Standard for Information Technology - IEEE 802.11ad.
[2] 2015. RF Characterization Data of UBX USRP Daughterboard. https://files.ettus.com/performance_data/ubx/.
[3] 2020. 5G NR Base Station (BS) Radio Transmission and Reception. Technical Specification (TS) 38.211. 3rd Generation Partnership Project (3GPP).
[4] 2024. Baluns - BALH-0006. https://markimicrowave.com/products/connectorized/baluns/balh-0006/.
[5] 2024. DC Blocks - BLK-89-S+. https://www.mouser.com/datasheet/2/1030/BLK_89_S_2b-1700195.pdf.
[6] 2024. Designing with the Zynq UltraScale+ RFSoC. https://learningcatalog-amd.netexam.com/Certification/46089/designing-with-the-zynq-ultrascale-rfsoc.
[7] 2024. Introduction to Zynq UltraScale+ RFSoC RF Data Converter v2.6 Gen 1/2/3/DFE LogiCORE IP Product Guide (PG269). https://docs.amd.com/r/en-US/pg269-rf-data-converter.
[8] 2024. RFSoC 2x2 kit. https://www.amd.com/en/corporate/university-program/aup-boards/rfsoc2x2.html.
[9] 2024. SPEAR GitHub. https://github.com/functions-lab/SPEAR.
[10] 2024. Xilinx RF-DC C / Python API. https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/rfdc.
[11] 2024. Xilinx/PYNQ. https://github.com/Xilinx/PYNQ.
[12] 2024. Zynq UltraScale+ RFSoC ZCU216 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/zcu216.html.
[13] Hoda Barkhordar-Pour, Jin Gyu Lim, Mohammed Almoneer, Patrick Mitran, and Slim Boumaiza. 2023. Real-time FPGA-based implementation of digital predistorters for fully digital MIMO transmitters. In *Proc. IEEE IMS'23*.
[14] Rodolfo Carobene, Alessandro Candido, Javier Serrano, Alvaro Orgaz-Fuertes, Andrea Giachero, and Stefano Carrazza. 2023. Qibosoq: an open-source framework for quantum circuit RFSoC programming. *arXiv preprint arXiv:2310.05851* (2023).
[15] Tingjun Chen, Prasanthi Maddala, Panagiotis Skrimponis, Jakub Kolodziejski, Abhishek Adhikari, Hang Hu, Zhihui Gao, Arun Paidimarri, Alberto Valdes-Garcia, Myung Lee, Sundeep Rangan, Gil Zussman, and Ivan Seskar. 2023. Programmable and open-access millimeter-wave radios in the COSMOS Testbed: Design, deployment, and experimentation. *Computer Networks* 234 (2023), 109922.
[16] Zhihui Gao, Zhenzhou Qi, and Tingjun Chen. 2024. Mambas: Maneuvering analog multi-user beamforming using an array of subarrays in mmWave networks. In *Proc. ACM MobiCom'24*.
[17] Junfeng Guan, Arun Paidimarri, Alberto Valdes-Garcia, and Bodhisatwa Sadhu. 2021. 3-D imaging using millimeter-wave 5G signal reflections. *IEEE Trans. Microw. Theory Tech.* 69, 6 (2021), 2936–2948.
[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 *Proc. ACM MobiSys'21*.
[19] Arm Ltd. 2021. AMBA AXI-Stream Protocol Specification.
[20] Xilinx Ltd. 2022. AXI DataMover v5.1 LogiCORE IP Product Guide. https://docs.amd.com/r/en-US/pg022_axi_datamover.
[21] Arun Paidimarri, Asaf Tzadok, Sara Garcia Sanchez, Atsutse Kludze, Alexandra Gallyas-Sanhueza, and Alberto Valdes-Garcia. 2024. EyeBeam: A mmWave 5G-compliant platform for integrated communications and sensing enabling AI-based object recognition. *IEEE J. Sel. Areas Commun.* 42 (2024), 2354–2368.
[22] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, Harish Krishnaswamy, Sumit Maheshwari, Panagiotis Skrimponis, and Craig Gutterman. 2020. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. ACM MobiCom'20*.
[23] Alphan Şahin, Mihail L Sichitiu, and İsmail Güvenç. 2023. A millimeter-wave software-defined radio for wireless experimentation. In *Proc. IEEE CNERT'23*.
[24] Marius Šiaučiulis, David Northcote, Josh Goldsmith, Louise H Crockett, and Šarūnas Kaladė. 2023. 100GBit/s RF sample offload for RFSoC using GNU Radio and PYNQ. In *Proc. IEEE NEWCAS'23*.
[25] 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. *Rev. Sci. Instrum.* 93, 4 (2022).
[26] Kyle A Steiner and Mark B Yeary. 2023. A 1.6-GHz sub-Nyquist-sampled wideband beamformer on an RFSoC. *IEEE Trans. on Radar Syst.* 1 (2023), 308–317.
[27] David Volz, Andreas Koch, and Bastian Bloessl. 2023. Software-defined wireless communication systems for heterogeneous architectures. In *Proc. ACM MobiCom'23*.
[28] Renjie Zhao, Timothy Woodford, Teng Wei, Kun Qian, and Xinyu Zhang. 2020. M-cube: A millimeter-wave massive MIMO software radio. In *Proc. ACM MobiCom'20*.