Dual-Mode PSK Transceiver on SDR With FPGA

Wuqiong Zhao, *Student Member, IEEE*

***Abstract*—In this experiment, we implement a dual-mode PSK transceiver on SDR with FPGA, supporting both BPSK and QPSK. Moreover, the transceiver is designed to be able to switch between the two modes by introducing a packet-based communication protocol, where modulation information can be extracted from the packet header. The design is resource-efficient implemented using block diagrams with intellectual property (IP) cores and Verilog modules in Vivado. Both simulation results and experiment observations on an SDR platform verify the effectiveness of the design.**

***Index Terms*—Phase-shift keying (PSK), software-defined radio (SDR), transceiver design, modulation, demodulation, field programmable gate array (FPGA).**

I. Introduction

SOFTWARE-DEFINED radio (SDR) is useful in various applications, including rapid prototyping and research. A millimeter wave (mmWave) SDR platform [1] can enable research in both mmWave physical-layer communications and the high-level networking problems. Phase-shift keying (PSK) is a popular modulation scheme in digital communications. Among PSK, the simplest two are the binary PSK (BPSK) and the quadrature PSK (QPSK). The carrier extraction is required for coherent demodulation, though differential encoding can be used to avoid the carrier extraction. Field programmable gate array (FPGA) is a popular choice for SDR baseband processing, due to its flexibility and high performance. In this project, instead of employing high-level synthesis (HLS) [2], we directly implement the transceiver on FPGA using hardware description language (HDL) Verilog, for better control of the underlying hardware and a more efficient design.

To benefit the research and learning community, the design sources (Vivado project) and this paper (in LaTeX) are open source [3]. The contributions of this paper is summarized as follows:

1) We implement a dual-mode PSK transceiver on SDR with FPGA, enabling both BPSK/QPSK modulation and coherent demodulation, with carrier synchronization and symbol synchronization.
2) A packet-based communication is introduced to enable the switching between BPSK and QPSK based on header fields, another step towards to a flexible and ready-to-use transceiver.
3) The open-source design is implemented on a Zynq-7020 FPGA and verified on an SDR platform, demonstrating its effectiveness.

Date of publication 16 January 2024; date of current version 17 January 2024. The author would like to thank Lecturer Dan Yang for the guidance and help in the course of the experiment. The author would also like to thank AI tools including GitHub Copilot and Claude.AI for their help in preparing this paper.

Wuqiong Zhao is with Southeast University, Nanjing 211189, China. (e-mail: wqzhao@seu.edu.cn, website: https://wqzhao.org).

Online URL: https://go.wqzhao.org/sdr-psk-fpga

II. System Overview

A. Software-Defined Radio

We employ eNodeX 30B [4], an SDR platform equipped with a pair of configurable Global System for Mobile Communications (GSM) transmitter (Tx) and receiver (Rx) antennas. An MS Windows software is provided for configurations, including the sampling frequency, the carrier frequency, attenuation and gain, etc.

B. Transceiver Design

The current transmitter and receiver are implemented on the same FPGA, but the implementation can be readily extended for different FPGAs with small frequency offsets. The system overview is shown in Fig. 1.

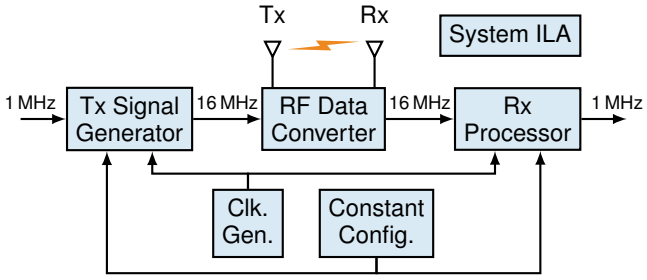


Fig. 1. Transceiver system overview.

Clock Generator. Required clocks are generated from the programmable logic (PL) clock. All reset signals are generated using Processor System Reset Modules [5], which can provide synchronized power-up reset signals.

RF Data Converter. This block contains analog-to-digital converters (ADCs) and digital-to-analog converters (DACs), enabled by a vendored AD9361 module [6].

Tx Signal Generator. The transmitted signal is generated in this block. Currently, it repeats a certain pattern and does not accept external input. But it can be readily extended to a more complicated design, based on the current interface template.

Rx Processor. This block is responsible for processing the received signal, including demodulation and data extraction from a packet. It is the most complex (and the core) block in the system.

System ILA. The system integrated logic analyzer (system ILA) [7] is used to observe the internal signals.

Constant Configurations. Several parameters can be configured in this block. Most importantly, the mode control constants (`MODE_CTRL`) are shown in Table I.

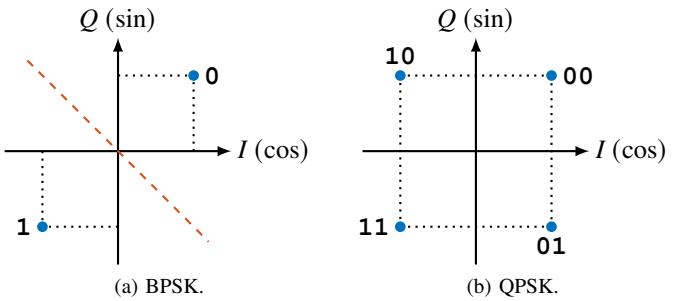
TABLE I
MODE CONTROL CONSTANTS

Mode	Localparam	Value	is_bpsk	Packet
BPSK	MODE_BPSK	4'b0001 (1)	1'b1	No
QPSK	MODE_QPSK	4'b0010 (2)	1'b0	No
Mixed	MODE_MIX	4'b0100 (4)	variable	Yes

The system can be configured to work in BPSK, QPSK, or the mixed mode. The remaining part of the paper will focus on the mixed mode, which has MODE_CTRL equal to MODE_MIX (4'b0100) and has a packet-based communication. Details of the design of packets are in Section V.

C. BPSK/QPSK Modulation

The BPSK and QPSK modulation constellation graphs used in the system are shown in Fig. 2. Different from the traditional setting, out adopted BPSK constellation in Fig. 2(a) is a combination of in-phase (I) and quadrature (Q) components. This is to make sure the phases used in BPSK are among those in QPSK, enabling a smooth transition between the two modes, which is critical because the header field is always modulated in BPSK. The QPSK constellation in Fig. 2(b) satisfies the Gray code: two adjacent values in the constellation differ in only one bit.



(a) BPSK. (b) QPSK.

Fig. 2. BPSK/QPSK modulation constellation used in the system.

The design heavily relies on the **a**dvanced e**x**tensible **i**nterface (AXI) [8], and the AXI stream (AXIS) protocol is used for data transmission, which requires the data width is a multiple of 8 bits (1 byte) in Vivado. Therefore, we define the BPSK uses bit 1, and QPSK uses bits 1 and 0, both counting from the least significant bit (LSB).

III. TRANSMITTER

A. Carrier NCO

The carrier frequency is generated by a numerically controlled oscillator (NCO). In Vivado, we use the Direct Digital Synthesis (DDS) Compiler IP [9] as the NCO. Both the cosine and sine components are used. Due to the need of carrier synchronization (see Section IV-B), the DDS does not have a fixed frequency, but an input stream signal controlling the phase, connected from the module NCO_Phase (see Eq. (3)).

B. PSK Modulation

1) Pseudo-random Number (PN) Generator: In this experiment, the transmitted signal are pseudo-random number (PN) sequences. Typically, we implement the PN generator with $N = 4$ (period is $2^4 - 1 = 15$) and $N = 5$ (period is $2^5 - 1 = 31$) using a shift register [10]. Thus, the 2-bits signal aggregating the I and Q components has the period of 465. The Verilog code for the module PN_Gen is shown below.

```
module PN_Gen # (parameter N = 5) (
  input      clk,
  output reg pn
);
  reg [N-1:0] PN_buf = 1; wire rst;
  generate
  if (N == 5)
    always @ (posedge clk)
      if (rst) begin PN_buf <= 5'd1; pn <= 0; end
      else begin
        PN_buf <= { PN_buf[3:0], PN_buf[4] ^ PN_buf[2] };
        pn <= PN_buf[4];
      end
  else if (N == 4)
    always @ (posedge clk)
      if (rst) begin PN_buf <= 4'd1; pn <= 0; end
      else begin
        PN_buf <= { PN_buf[2:0], PN_buf[3] + PN_buf[2] };
        pn <= PN_buf[3];
      end
  else ; // NOT implemented yet!
  endgenerate
  assign rst = !(|PN_buf);  // reset when PN_buf is all 0
endmodule
```

2) Modulation With I and Q Streams: The modulation is performed by selecting the appropriate carrier phase according to the input bits and the constellation (in Fig. 2). Note the Q component is always 90° ahead of the I component. The modulated I and Q components are then connected to the corresponding DAC ports.

IV. RECEIVER

A. Overview

The receiver performs carrier synchronization, symbol synchronization and PSK detection first, without considering the packet structure. The depacketizer and the packet extraction is based on the synchronized bit stream after PSK detection.

B. Carrier Synchronization Using Costas Loop

A Costas loop [11] is used for carrier synchronization. The basic idea of a Costas loop is to provide an error (carrier phase offset) feedback. The negative feedback is used to lock the carrier phase to the PSK signal. The generation of the error feedback is specific to different modulation schemes, as well as their constellation graphs.

The proposed dual-mode Costas loop is shown in Fig. 3, which can be switched between BPSK and QPSK via a control signal is_bpsk. The two multipliers directly following the PSK stream input are phase detectors, whose low-frequency components represent the data stream signal. Therefore, the data stream I and Q signal can be extracted after passing the mixed signals through a low-pass filter (LPF). Thus, it is ready to map the obtained I/Q signal to the constellation graph, and generate the error signal.

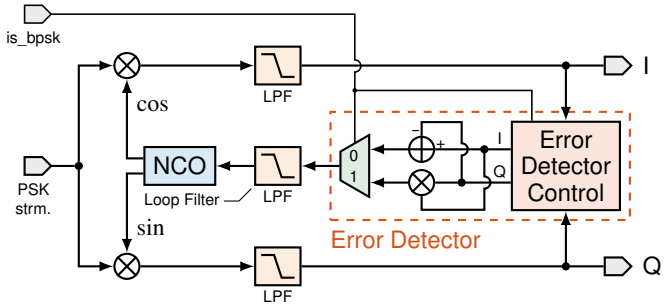


Fig. 3. Costas loop for carrier synchronization with BPSK/QPSK support.

The error feedback for BPSK is defined as

$$e_{\text{BPSK}} = (I+Q)\times(I-Q), \tag{1}$$

and the error feedback for QPSK is defined as

$$e_{\text{QPSK}} = I\cdot\text{sgn}(Q) - Q\cdot\text{sgn}(I). \tag{2}$$

For stability, the loop filter as an LPF is added before the error is fed back to the NCO. The NCO implemented with DDS has the phase defined as

$$\begin{aligned}\phi[n] &= \phi[n-1] + \Delta\phi[n] \\ &= \phi[n-1] + (f_0 + k\cdot f_{\text{feedback}}),\end{aligned} \tag{3}$$

where $\phi[n]$ is the phase increment at time n, f_0 is the free running clock (4.096 MHz in our design), and k is the feedback coefficient. The core Verilog implementation of the Error Detector Control is given below. For QPSK, the feedback value is arithmetically right shifted by 6 bits to coarsely match the scale of that in BPSK.

```
if (is_bpsk) begin // BPSK
  out_I_tdata <= in_I_tvalid ? in_I_tdata + in_Q_tdata : 0;
  out_Q_tdata <= in_Q_tvalid ? in_I_tdata - in_Q_tdata : 0;
end
else begin // QPSK
  out_I_tdata <= in_I_tvalid ? (in_Q_tdata[WIDTH-1] ?
    -in_I_tdata : in_I_tdata) >>> 6 : 0;
  out_Q_tdata <= in_Q_tvalid ? (in_I_tdata[WIDTH-1] ?
    -in_Q_tdata : in_Q_tdata) >>> 6 : 0;
end
```

In the Costas loop, the feedback coefficient k in Eq. (3) is an important coefficient to finetune according to the system. This parameter is reflected as `FEEDBACK_SHIFT`, and $k \triangleq 2^{-\texttt{FEEDBACK\_SHIFT}}$. The convergence of the Costas loop under different carrier frequency offset (CFO) values is discussed in Section VI-B.

C. Symbol Synchronization Using Gardner Loop

A Gardner loop [12] is used to achieve symbol (timing) synchronization. The structure of a Gardner loop is shown in Fig. 4. The loop itself is not directly shown in the figure, which is part of the timing corrector with feedback.

To reduce implementation complexity, we use the sign of strobe values as mentioned in [12]. The total symbol timing error considering I and Q components is then simplified as

$$\begin{aligned}u_t(r) = {} & y_I(r-\tfrac{1}{2})\left[\text{sgn}\left(y_I(r)\right) - \text{sgn}\left(y_I(r-1)\right)\right] \\ & + y_Q(r-\tfrac{1}{2})\left[\text{sgn}\left(y_Q(r)\right) - \text{sgn}\left(y_Q(r-1)\right)\right],\end{aligned} \tag{4}$$

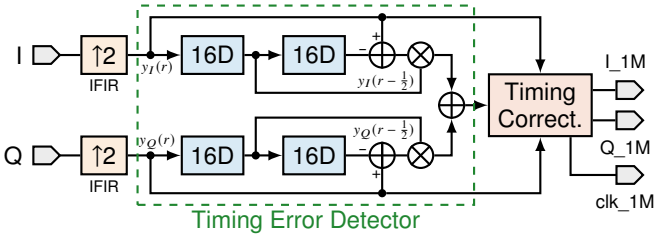


Fig. 4. Structure of a Gardner loop for symbol timing synchronization.

where r has a symbol frequency of 1.024 MHz. For better timing performance, we linearly interpolate the 16.385 MHz input I/Q data to 32.768 MHz. Therefore, $y_I(r-1)$ and $y_Q(r-1)$ are delayed by 32 clocks. In FPGA implementation, for each I/Q stream, two shift registers of depth 16 are used. The output `clk_1M` signal indicates the symbol timing sampling point, which is high for one clock under 32.768 MHz. The I/Q output signals `I_1M` and `Q_1M` are synchronized with `clk_1M`. Notably, since we adopt the BPSK constellation in Fig. 2(a), the symbol timing error depends on both I and Q components, the same as QPSK. The correctness of symbol timing is shown in Fig. 10.

D. PSK Detection

Threshold detection. The detection is based on the constellation graph in Fig. 2, and the I/Q signals after the symbol synchronization:

$$\text{BPSK symbol} = \begin{cases}\texttt{1'b0}, & I+Q \geq 0, \\ \texttt{1'b1}, & I+Q < 0.\end{cases} \tag{5a}$$

$$\text{QPSK symbol} = \begin{cases}\texttt{2'b00}, & I \geq 0, Q \geq 0, \\ \texttt{2'b10}, & I < 0, Q \geq 0, \\ \texttt{2'b11}, & I < 0, Q < 0, \\ \texttt{2'b01}, & I \geq 0, Q < 0.\end{cases} \tag{5b}$$

It can be observed that the case conditions in Eq. (5b) satisfy the Gray code in Section II-C. In Verilog implementation, the threshold comparison is implemented by extracting the MSB of the I/Q signals.

Phase ambiguity. With the carrier synchronization method in Section IV-B, the phase ambiguity for detection is inevitably introduced: the constellation can rotate by 180° for BPSK, and ±90° or 180° for QPSK. For instance, it leads to flipped bits in BPSK. Therefore, the detected data stream should be corrected. The differential encoding [13] can be used, as the data is encoded in the phase difference, which is immune to the phase ambiguity. However, this method can increase the bit error rate (BER) with error propagation.

Phase correction with known sequences. The phase ambiguity can be corrected by using a known sequence. The existence of the known sequence can be detected by its differential results, and the phase of the known sequence is recorded for phase correction during the payload and header transmission. This approach does not add extra overhead in a packet-based communication system as there is a header field. Details of the implementation will be introduced in Section V.

V. Packet-Based Communication

A. Frame Structure

The frame structure is shown in Fig. 5, consisting of the TRN, HDR and the PLD fields.

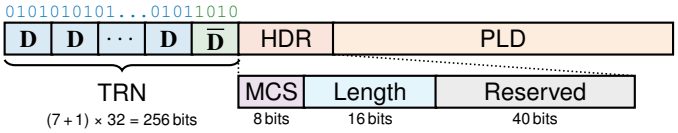


Fig. 5. Frame structure of the packet-based communication.

Training (TRN). The training field is used to provide packet timing information (coarse synchronization), as well as achieving synchronization of the carrier and symbol timing in the meantime. It consists of 7 repetitions of **D** and one $\overline{\mathbf{D}}$, where **D** and $\overline{\mathbf{D}}$ are of length 32. **D** and $\overline{\mathbf{D}}$ are repetitive sequences of '01' and '10', respectively. The design is inspired by the short training field (STF) in IEEE 802.11ad [14], but easier to implement at the cost of a lower signal-to-noise ratio (SNR). The training field from bit 0 to $(7+1)\times 32-1=255$ is defined as

$$\mathrm{TRN}[i]=\begin{cases}\mathrm{mod}(i,2), & i=0,1,\ldots,223,\\ \mathrm{mod}(i+1,2), & i=224,225,\ldots,255,\end{cases} \tag{6}$$

where $\mathrm{mod}(a,n)$ is the modulo operation returning the remainder of a division. Notably, the phase transition from bit 223 to 224 is used to indicate the boundary of the packet.

Header (HDR). The header field is used to provide packet information, including the modulation and coding scheme (MCS) and the packet length (Length) in bits. The remaining bits are reserved for future use. The MCS field currently only determines the use of BPSK or QPSK (no channel coding is used in this design). The MCSs for BPSK and QPSK are defined as '01010101' and '10101010' respectively.

Payload (PLD). The payload field is used to carry the actual data. Its length in bits (1 bit for each BPSK symbol, 2 bits for each QPSK symbol) should match the Length field in the header.

B. Packetizer Design

The packetizer finite state machine (FSM) has 5 states: IDLE, HDR, PLD, LAST and WAIT. The IDLE state is used to wait for the start of a packet. It transitions to the HDR state when both the `tvalid` and `tready` signals are high, i.e., the packet starts transmitting. The HDR state is used to transmit the header (including TRN and HDR)[1], as discussed in Section V-A. It then transmits to the PLD state[2] to transmit the payload stored in the first in, first out (FIFO). Notably, the FIFO depth should be no smaller than the header length (i.e., 320). The LAST state is used to transmit the last symbol of the payload, when the AXIS `tlast` signal will be high. After transmitting all payload data, the WAIT state is employed to consume all remaining data in the FIFO. This is only used in our design to better demonstrate the relationship between the transmitted bits and the received bits, and this state can be removed for other applications. Finally, after the FIFO is cleared, the state will transition to the IDLE state waiting for new data to be transmitted.

[1]For the packetizer, we do not distinguish TRN and HDR for simplicity, and they are called the header in contrast to the payload.

[2]If the payload symbol length is 1, it will directly transition to the LAST state.

C. SPB Detection

SPD detection consists of strength detection (SD), packet detection (PD) and boundary detection (BD). They jointly provide information for the coarse packet timing. The PD and BD are designed similar to [15] for IEEE 802.11ad/ay.

1) Strength Detection (SD): The strength detection checks the data stream I/Q amplitude from the Costas loop. It is useful because the PSK detector will always output a value, and it can confuse the packet parsing when the noise coincides with the packet sequence. For stability, the `SD_flag` is asserted when either the I or Q signal amplitude is larger than `RX_SD_THRESHOLD` at one clock within the window of `RX_SD_WINDOW` clocks.

2) Packet Detection (PD): The presence of a packet can be identified by detecting repetitive '010101…' sequence (modulated using BPSK). This can be performed by checking the continuous 1s for the differential results. The `PD_flag` is asserted when the differential results remain 1 for `RX_PD_WINDOW` clocks. Interestingly, the packet detection itself does not require a synchronized carrier, as a carrier phase shift does not significantly affect the '01' sequence detection.

3) Boundary Detection (BD): Exploiting the phase transition from bit 223 to 224 in the TRN field (see Section V-A and Fig. 5), the boundary detection can be performed by checking the differential value being 0 after the `PD_flag` being asserted. To ensure stability, the `BD_flag` is asserted after `RX_BD_WINDOW` clocks of continuous 1s in differential values.[3] At boundary detection, the sign of the bit 223 and 224 are also recorded in `BD_sgn`, which can be used to counter the phase ambiguity in BPSK demodulation.

D. Depacketizer Design

The depacketizer works the opposite way of the packetizer. It has 6 states: IDLE, TRN, HDR, PLD, LAST and WAIT. The TRN and the HDR state must be distinguished because of the need to extract useful information in the TRN field. The IDLE state transitions to the TRN state when `BD_flag` is high. Since the `BD_flag` contains the packet timing information, the depacketizer can count the number of clocks to determine the start of the HDR field. In the HDR field, both the MCS and packet length are extracted. Then, the depacketizer transitions to the PLD state to extract the payload. The `tuser` signal is used to indicate the modulation scheme (1 for BPSK and 0 for QPSK). The `tlast` signal is used to indicate the end of the payload, at the LAST state. To correct disassert the PD and BD signal, the WAIT state is employed to wait for one more clock before it goes to the IDLE state.

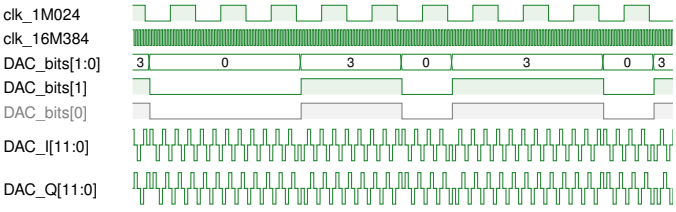
[3]Please refer to the comments in https://github.com/Teddy-van-Jerry/sdr-psk-fpga/blob/master/verilog/Rx_BD.v for an illustration.

VI. Simulation Results

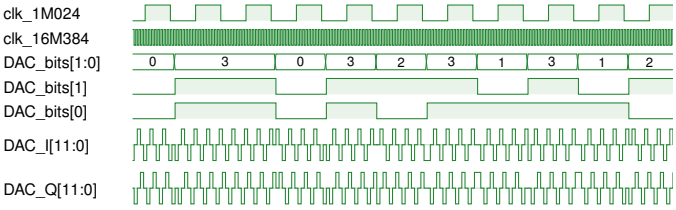
In all Verilog simulation, random noise is added to the Rx ADC, to better simulate the real-world scenario.

A. Transmitter

The transmitter DAC simulaiton results are shown in Fig. 6.



(a) BPSK modulation.



(b) QPSK modulation.

Fig. 6. DAC simulation results of the transmitter. The gray line denotes the wave is irrelevant. The DAC waveforms are displayed in the *hold* mode, in contrast to the *linear* mode.

Due to the limited number of samples (4 per period), the DAC output is not smooth, and shapes like a triangle wave. Nevertheless, the phase shift for both I and Q at symbol transitions is clear.

B. Carrier Synchronization Convergence

The simulation results of the Costas loop are shown in Fig. 7. The convergence is relatively fast. In the BPSK mode, the Costas loop can successfully handle the CFO smaller than 7.81 kHz. It can be verified by the fact that the feedback saturates around a constant, which corresponds to the CFO value. In the QPSK mode, the threshold value is 1.96 kHz.

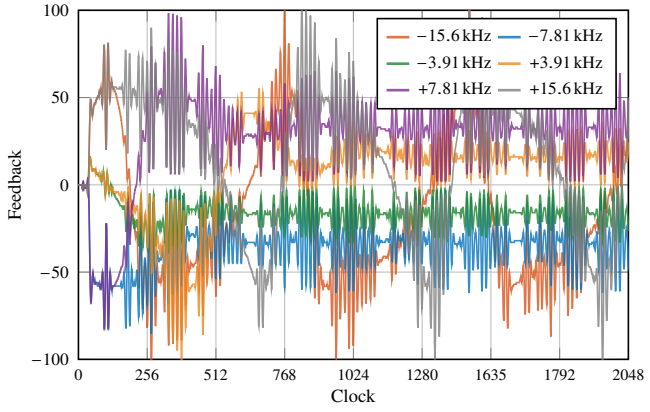


Fig. 7. Carrier synchronization for BPSK. The feedback value after the loop filter v.s. time, with different CFO.

Notably, the convergence performance of QPSK is poorer than BPSK in the current parameter set, which is by design. For a dual-mode system which transmits payload in both BPSK and QPSK with a BPSK header, the carrier synchronization is achieved at the BPSK header. Therefore, QPSK does not need a strong feedback. Furthermore, a smaller feedback can make the BPSK and QPSK transition smoother, avoiding a sudden phase jump of 90°.

C. Mixed-Mode Receiver

In this simulation, we show the successful transmission of packets in both BPSK and QPSK modulations. The simulation results are shown in Fig. 8.

The DAC transmits data when `DAC_vld` is high, and the payload is transmitted when `Tx_vld` is high. There is a delay of The two packet streams are clearly shown, with the first being QPSK and the second being BPSK, both of 128 bits length. The `Rx_tuser` signal shows the detected modulation scheme at the Rx, with 1 for BPSK and 0 for QPSK. Since QPSk has a 2× bit rate, the first packet payload is transmitted 2× faster than the first one.

VII. FPGA Implementation

The design is implemented in Vivado 2022.2 using block diagrams. Some block diagram designs are shown in Section A.

The hardware resources consumption on the Zynq-7020 (xc7z020clg484-1) board with the default synthesis and implementation strategy is shown in Table II. Therefore, the design itself is resource-efficient, and can be readily incorporated into a larger system.

TABLE II
Hardware Resources Consumption on Zynq-7020

Resource	Utilization	Available	Util. Rate
LUT	7,248	53,200	8.04%
LUTRAM	1,121	17,400	6.44%
FF	8,144	106,400	7.65%
BRAM	27	140	19.29%
DSP	51	220	23.18%
IO	33	200	16.50%
BUFG	10	32	31.25%
MMCM	2	4	50.00%

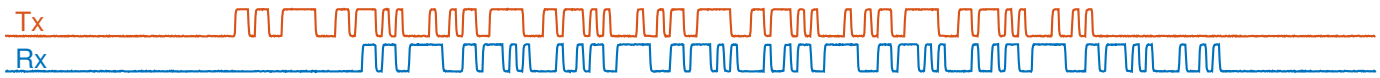
The FPGA design source is open source [3], and parts of the design can be easily reused in other projects.

VIII. Experiment Results on SDR

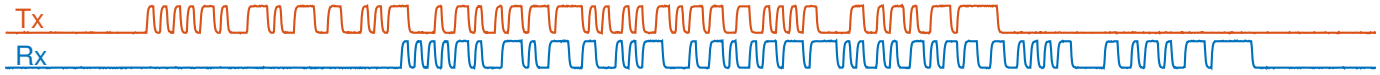
The eNodeX SDR platform [4] is configured to provide 1 Tx and 1 Rx at a sampling frequency of 16.384 MHz. The GSM antenna carrier frequency is 800 MHz, and the two antennas have a line-of-sight (LoS) channel.

The parameters used in this design are specially adjusted for received signal with strength ranging between 1/2 and 3/4 of the full ADC range. Therefore, you should adapt the antenna attenuation and gains to get the appropriate signal strength. With AD9361 [6], each ADC stream has 12 bits as a signed number, i.e., ranging between −2048 and 2047.

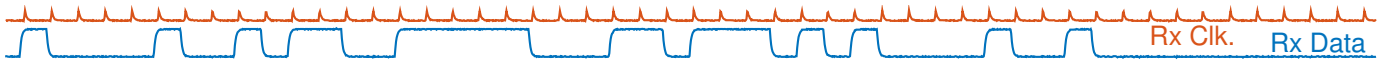
Fig. 8. Mixed-mode simulation results. The first packet is transmitted using QPSK, and the second packet is transmitted using BPSK.



(a) BPSK packet data (duration: 200 µs).



(b) QPSK packet data (duration: 100 µs).



(c) Symbol timing clock and BPSK data at Rx (duration: 50 µs).

Fig. 9. Two-channel oscilloscope results of GPIO outputs.

The experiment results are observed via a system ILA in Vivado, and 4 general-purpose input/output (GPIO) pins are used to output some 1-bit signals, including the 1-bit Tx and Rx data stream and their corresponding clock.

In our design, 4 GPIO pins are connected, as listed in Table III. Fig. 9 shows the oscilloscope results of the GPIO outputs.

TABLE III
GPIO PIN CONNECTIONS

Pin	Signal
GPIO_TH1	Tx 1-bit sequence (clock: 1.024 MHz or 2.048 MHz)
GPIO_TH2	Rx 1-bit sequence (sync. w/ Rx timing clock when BPSK)
GPIO_TH3	2.048 MHz global clock from the clock divider
GPIO_TH4	Rx timing clock (~1.024 MHz)

In Fig. 9(a) and Fig. 9(b), the GPIO_TH1 and GPIO_TH2 pins are connected, showing the transmitted 1-bit sequence (Tx) and the respective received 1-bit sequence (Rx). Clearly, the sequence is successfully recovered in both cases, with a certain time delay. The structure of pn_5 sequence in BPSK is clearly seen in Fig. 9(a), which has a bit frequency of 1.024 MHz. By contrast, the bit frequency of QPSK packets in Fig. 9(b) is 2.048 MHz. The symbol timing clock is shown in Fig. 9(c), which is not synchronized with the Rx data in BPSK, as the Rx data is connected after a FIFO with the master clock of 1.024 MHz. The symbol timing clock has a measured frequency of near 1.024 MHz on the oscilloscope.

Fig. 10 gives the ILA results for two cases involving BPSK and QPSK. The ILA results are consistent with the oscilloscope results, and provide additional insight into the system, including PSK detection and packet extraction. The I_16M and the Q_16M signals are the I and Q components of the data stream signal from the Costas loop, while the I_1M and the Q_1M signals are provided by the Gardner loop after symbol timing. The out_clk_1M clock is the symbol timing clock, which is near optimal, as designed by the Gardner loop.

IX. DISCUSSIONS

A. Possible Enhancement

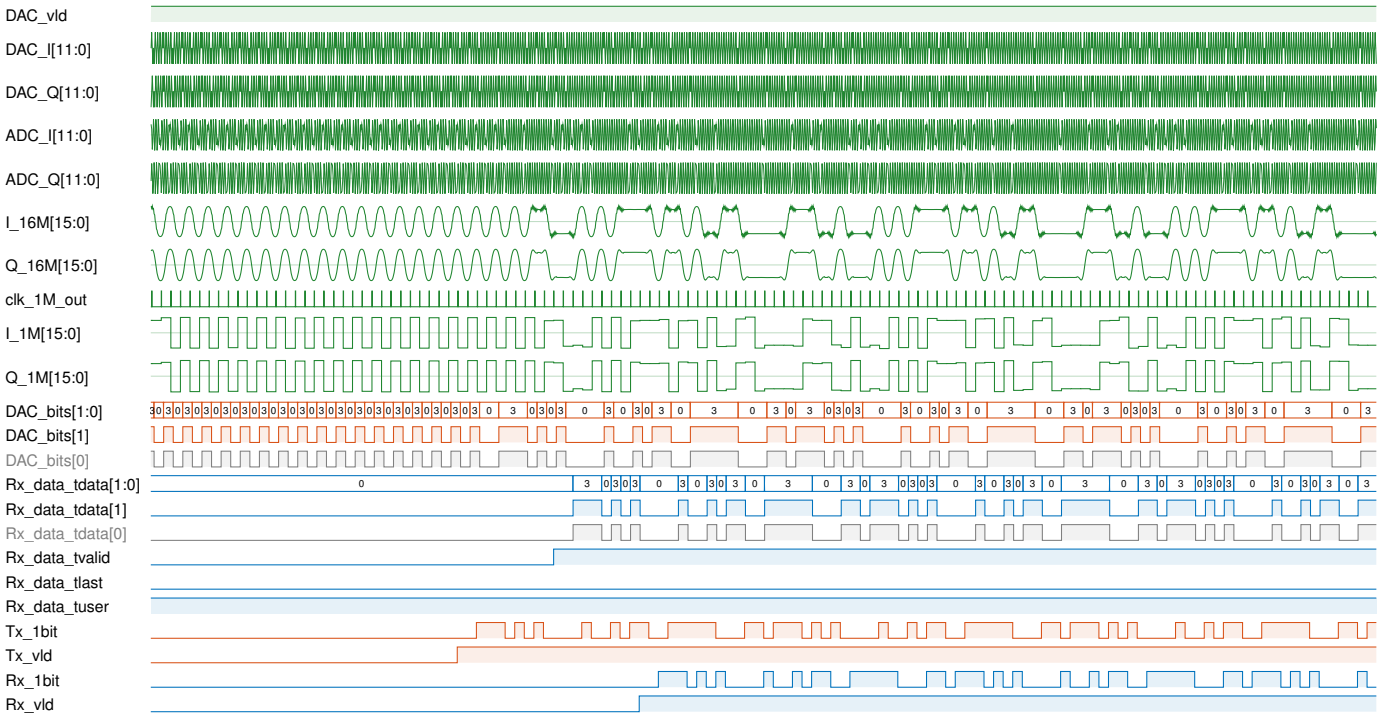
Frame structure design. CRC and/or checksums can be added to the frame structure to enhance the packet transmission stability in Section V-A. Additionally, the HDR and TRN field in Section V-A may have some improvements. The Golay sequence used in [14]–[16] is one possible better solution: it has a better performance at low SNR, and the packet detection based on autocorrelation [15] can reduce power consumption by disabling the demodulation when there is no packet.

Changing parameters on the fly. AXI peripheral [17] can be used to change the parameters in the Const_Config on the fly, if the board allows. Therefore, the BPSK/QPSK/mixed mode can be switched directly, and the parameters in the design can be changed without recompiling the design, allowing easier use in various scenarios.

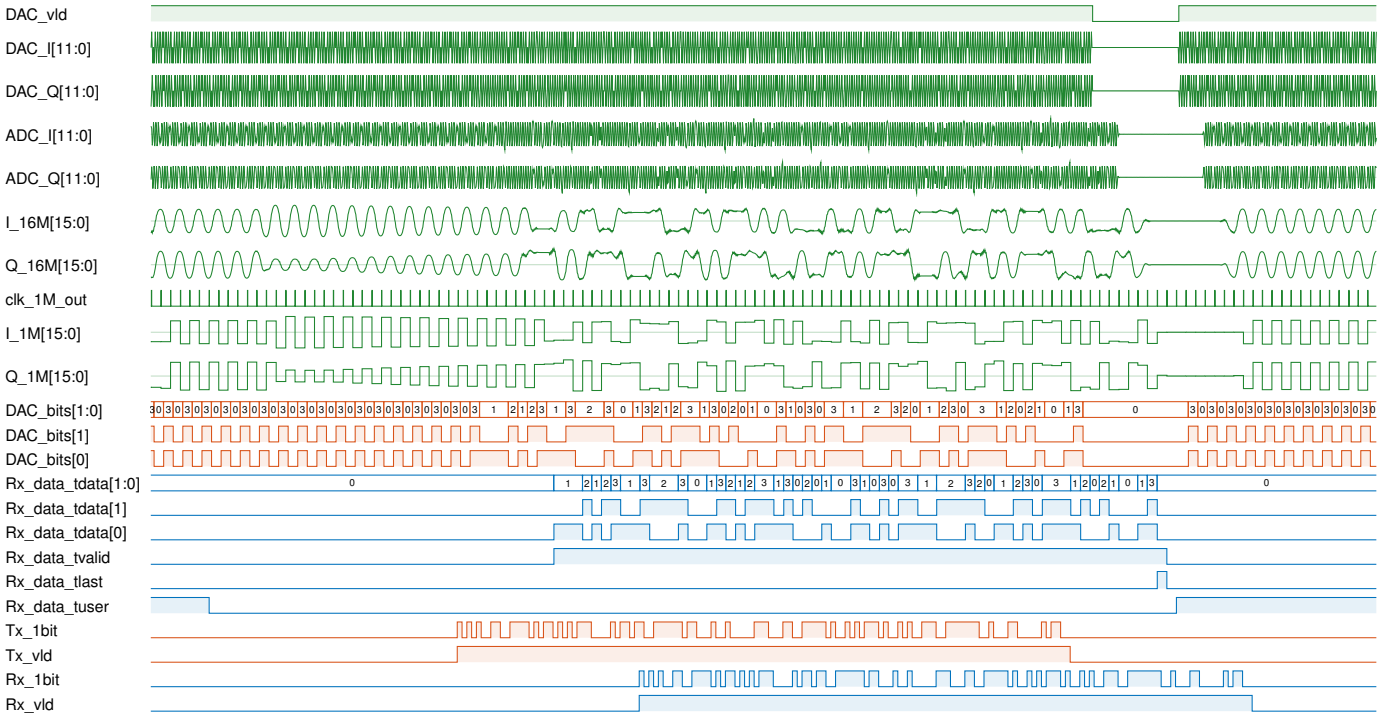
Pulse shaping. The SNR performance will be improved if appropriate pulse shaping is employed. This enhancement is left as an exercise for the reader.

B. Possible Extensions Beyond the Experiment

The training (TRN) field can be better utilized for additional experiments. For example, SNR can be estimated at the TRN field, so a plot with BER against SNR can be obtained. Channel estimation algorithms [18], [19] can also be investigated on the platform, allowing the test of the performance of the algorithms in real-world scenarios. Besides, it is also possible to investigate the auto generation [16] of digital circuits, so that the design can be automatically generated according to different requirements.



(a) A BPSK packet (with Rx_data_tuser high).



(b) A QPSK packet (with Rx_data_tuser low).

Fig. 10. System ILA results (some signals not shown, and this is plotted with the saved CSV data). The system ILA has a window of 4,096, with the ILA clock frequency of 32.768 MHz (twice the ADC/DAC sampling rate). The red and blue signals are at the Tx and Rx side, respectively.

X. Conclusion

In this paper, we implement the PSK transceiver with modulation and coherent demodulation on SDR with FPGA. The proposed high-level system design and communication techniques feature carrier synchronization, symbol synchronization and packet-based communication. The design is tested and verified on a SDR platform, successfully performing wireless transmission in 2.048 Mb/s data rate using QPSK or 1.024 Mb/s data rate using BPSK with a 16.384 MHz bandwidth (considering I/Q sampling) channel. As an open source project, this paper opens up new possibilities of easy extension of the PSK transceiver.

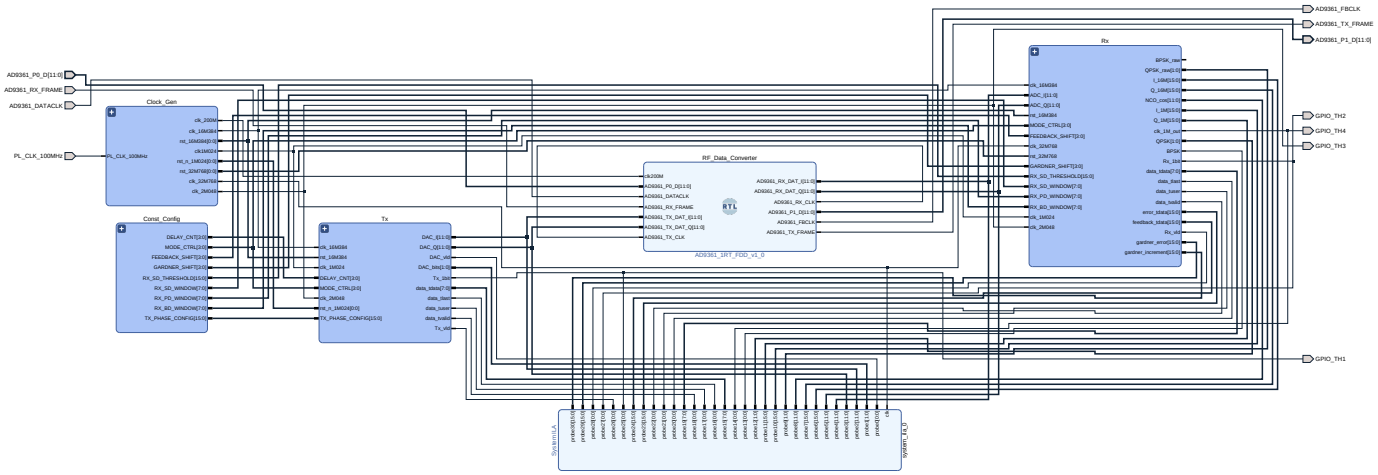


Fig. 11. Top block diagram.

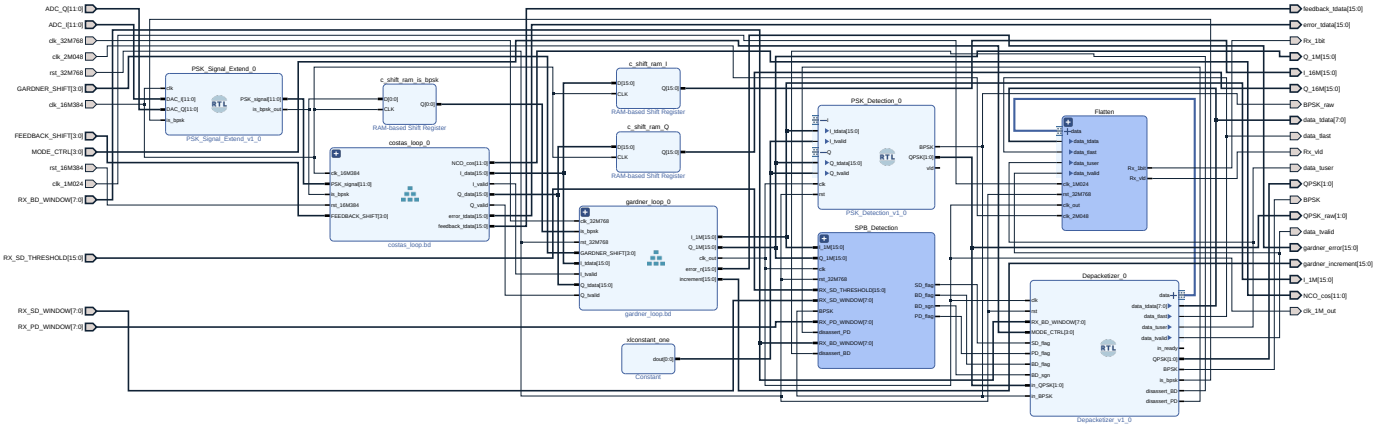


Fig. 12. Rx processor block diagram.

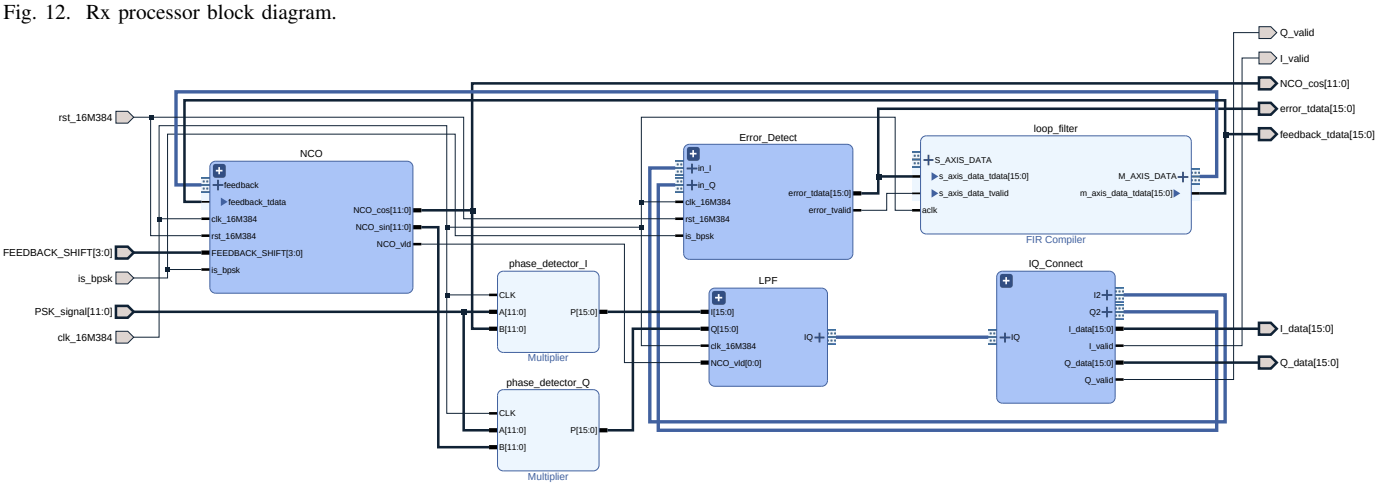


Fig. 13. Costas loop block diagram for carrier synchronization.

APPENDIX A
BLOCK DIAGRAMS

A. Block Diagrams Design

The top level block diagram corresponding to Fig. 1 is shown in Fig. 11, and the Rx processor block diagram is shown in Fig. 12. Among the Rx processor, the Costas loop and the Gardner are both inserted block diagrams.

The Costas loop design is shown in Fig. 13, where the phase detectors are implemented using the multiplier IP [20], and the NCO hierarchy is mainly composed of the DDS Compiler IP [9]. The LPF hierarchy contains a low-pass filter with 2 channels (I and Q).

More block diagrams are provided at https://github.com/Teddy-van-Jerry/sdr-psk-fpga/tree/master/schematic [3].

B. Debugging With Block Diagrams

AXI connections. AXI and AXI stream (AXIS) interfaces should be carefully dealt with in the block diagram. When connecting one of the signal in the bus elsewhere (e.g., to ILA), a manual connection of the signal to the corresponding AXIS interface signal is required. Besides, when the AXI interface is not associated with a clock (for example a combinational logic with no register output) the `FREQ_HZ` property needs to be correctly set (either using Tcl or GUI) before validating the design.

Testbenches for block diagrams. It is relatively difficult to simulate block diagrams than Verilog modules. The possible way is to generate the output products and find the correct module (name containing `impl`). Be careful when adding sources for simulation, and Tcl scripts are provided in [3] to ensure the correctness. Vivado can be buggy (mostly reluctant!) when updating sources from the block diagram (for both simulation and implementation), and therefore it is advised to double-check the netlist file timestamp. A workaround by forcing Vivado to update the design is adding a dummy port, updating the module, and then removing it.

Suggestions. The project could have been more smooth if I did without block diagrams (or for AMD to fix all these bugs). You can directly use the Verilog modules in [3] for your own project if you are not accustomed to block diagrams. Nevertheless, block diagrams are at least better looking and easier to use for a system-level project, especially when dealing with the system on a chip (SoC).

APPENDIX B
FIGURES IN THIS PAPER

All figures except for block diagrams in this paper are created using T*i*kZ, part of LaTeX. The way I create them is quite interesting, involving extensive Python processing and optimization, and you can find the source code in the GitHub repository [3].

REFERENCES

[1] R. Zhao, T. Woodford, T. Wei, K. Qian, and X. Zhang, "M-cube: A millimeter-wave massive MIMO software radio," in *Proc. ACM 26th Annu. Int. Conf. Mobile Comput. Network. (MobiCom)*, Sep. 2020, pp. 1–14.

[2] W. Zhao, C. Li, Z. Ji, Z. Guo, X. Chen, Y. You, Y. Huang, X. You, and C. Zhang, "Flexible high-level synthesis library for linear transformations," *IEEE Trans. Circuits Syst. II*, 2023, to be published.

[3] W. Zhao, "sdr-psk-fpga," GitHub repository, 2024. [Online]. Available: https://github.com/Teddy-van-Jerry/sdr-psk-fpga

[4] Wuhan Labtech, "eNodeX 30B SDR innovation & implementation platform," 2024. [Online]. Available: http://www.labtech.cn/Product-software-eNodeX-30B.html

[5] AMD, "Processor system reset module v5.0 product guide (PG164)," Nov. 2015. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg164-proc-sys-reset

[6] A. Devices, "Ad9361 reference manual (UG-570)," Jun. 2015. [Online]. Available: https://www.analog.com/media/en/technical-documentation/user-guides/ad9361.pdf

[7] AMD, "System integrated logic analyzer v1.1 product guide (PG261)," Feb. 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg261-system-ila

[8] ——, "Vivado design suite: AXI reference guide (UG1037)," Jul. 2017. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide

[9] ——, "DDS compiler v6.0 product guide (PG141)," Jan. 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg141-dds-compiler

[10] F. James, "A review of pseudorandom number generators," *Comput. Phys. Commun.*, vol. 60, no. 3, pp. 329–344, Oct. 1990.

[11] M. Simon and W. Lindsey, "Optimum performance of suppressed carrier receivers with Costas loop tracking," *IEEE Trans. Commun.*, vol. 25, no. 2, pp. 215–227, Feb. 1977.

[12] F. Gardner, "A BPSK/QPSK timing-error detector for sampled receivers," *IEEE Trans. Commun.*, vol. 34, no. 5, pp. 423–429, May 1986.

[13] W. Weber III, "Differential encoding for multiple amplitude and phase shift keying systems," *IEEE Trans. Commun.*, vol. 26, no. 3, pp. 385–391, Mar. 1978.

[14] *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications – Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band*, IEEE Std. 802.11ad, 2012.

[15] J. O. Lacruz, R. R. Ortiz, and J. Widmer, "A real-time experimentation platform for sub-6 GHz and millimeter-wave MIMO systems," in *Proc. ACM 19th Annu. Int. Conf. Mobile Syst. Appl. Serv. (MobiSys)*, Jun. 2021, pp. 427–439.

[16] W. Zhao, C. Li, Z. Ji, Y. You, X. You, and C. Zhang, "Automatic timing-driven top-level hardware design for digital signal processing," in *Proc. IEEE 15th Int. Conf. ASIC (ASICON)*, Oct. 2023.

[17] AMD, "AXI external peripheral controller (EPC) v2.0 product guide (PG127)," Oct. 2016. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg127-axi-epc

[18] W. Zhao, Y. You, L. Zhang, X. You, and C. Zhang, "OMPL-SBL algorithm for intelligent reflecting surface-aided mmWave channel estimation," *IEEE Trans. Veh. Technol.*, vol. 72, no. 11, pp. 15 121–15 126, Nov. 2023.

[19] Y. You, W. Zhao, L. Zhang, X. You, and C. Zhang, "Beam pattern and reflection pattern design for channel estimation in RIS-assisted mmwave MIMO systems," *IEEE Trans. Veh. Technol.*, 2023, to be published, doi: 10.1109/TVT.2023.3309950.

[20] AMD, "Multiplier v12.0 product guide (PG108)," Nov. 2015. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg108-mult-gen

Wuqiong Zhao (S'22) is a senior undergraduate student pursuing the Bachelor's Degree in communications engineering, at the National Mobile Communications Research Laboratory of Southeast University, Nanjing, China. His research interests include wireless systems and networking, signal processing, and hardware implementation. He has published several journal papers at IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (TVT) and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II (TCAS-II). He was also reviewers of IEEE ISCAS 2023 and IEEE TCAS-II. Visit the website https://wqzhao.org for more information.