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 packet-based communication, where modulation information can be extracted from the packet header.**

***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 interesting, like application in millimeter wave [1]. FPGA is also interesting!

Instead of employing high-level synthesis (HLS) [2], we directly implement the transceiver on FPGA using hardware description language (HDL) Verilog, for a better control of the underlying hardware.

The design source (Vivado project) and this paper (in LaTeX) are open source [3].

II. System Overview

A. Software-Defined Radio

B. Transceiver Design

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

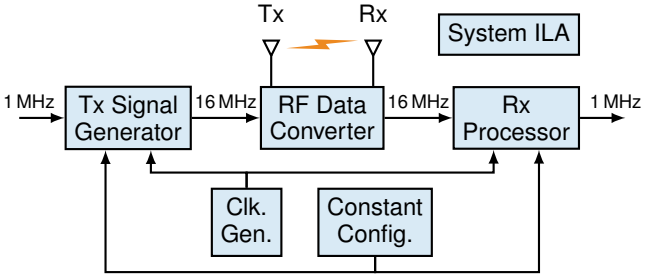


Fig. 1. Transceiver system overview.

Clock Generator. All reset signals are generated using Processor System Reset Modules [4], which can provide synchronized power-up reset signals.

Date of publication 4 January 2024; date of current version 4 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

RF Data Converter. This block is contains analog-to-digital converter (ADC) and digital-to-analog converter (DAC), enabled by a vendored AD9361 module.

Tx Signal Generator.

Rx Processor.

System ILA. The system integrated logic analyzer (system ILA) [5] is used to observe the internal signals.

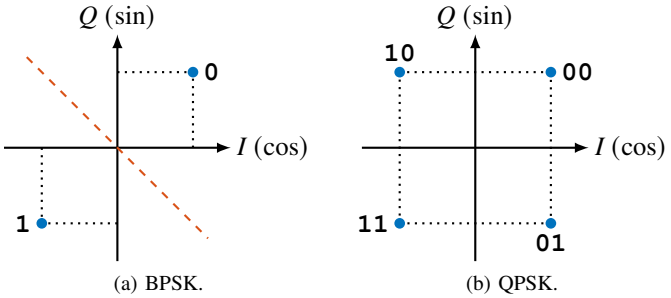
Constant Configurations. Several parameters can be configured in this block. Most importantly, the mode control constants (`MODE_CTRL`) are shown in Table I.

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

C. BPSK/QPSK Modulation

The BPSK and QPSK modulation constellation graphs used in our system are shown in Fig. 2. Different from the traditional setting, out adopted BPSK constellation in Fig. 2(a) is a combination of I and Q components.



(a) BPSK. (b) QPSK.

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

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 core to generate the NCO.

B. PSK Modulation

C. Pseudo-random Noise (PN) Generator

In this experiment, the transmitted signal are pseudo-random noise (PN) sequences. Typically, we implement the PN generator with $N = 4$ and $N = 5$.

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
```

IV. Receiver

A. Overview

B. Carrier Synchronization Using Costas Loop

Costas loop [6].

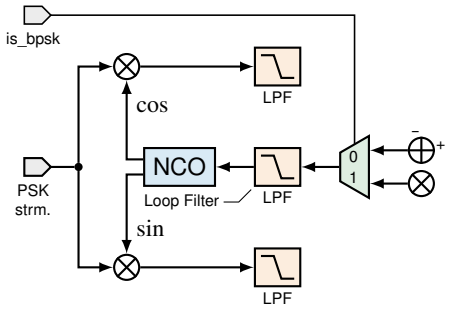


Fig. 3. Costas loop for carrier synchronization with BPSK/QPSK support.

```
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
```

C. Symbol Synchronization Using Gardner Loop

A Gardner loop [7] is used to achieve symbol (timing) synchronization. The structure of a Gardner loop is shown in Fig. 4.

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

To reduce implementation complexity, we use the sign of strobe values as mentioned in [7]. The total symbol timing error considering I and Q components is

$$u_t(r) = y_I(r-\tfrac{1}{2})\left[\operatorname{sgn}\left(y_I(r)\right) - \operatorname{sgn}\left(y_I(r-1)\right)\right] + y_Q(r-\tfrac{1}{2})\left[\operatorname{sgn}\left(y_Q(r)\right) - \operatorname{sgn}\left(y_Q(r-1)\right)\right], \quad (1)$$

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. 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.

V. Packet-Based Communication

A. Frame Structure

The frame structure is shown in Fig. 5.

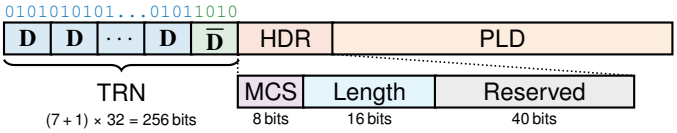


Fig. 5. Frame structure of the packet-based communication.

Training (TRN). The training field is used to provide packet timing information (coarse synchronization), as well as synchronize the carrier and symbol timing. 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 training field from bit 0 to $(7+1)\times 32-1 = 255$ is defined as

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

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. 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

C. Depacketizer Design

D. SPB Detection

1) Strength Detection (SD):

2) Packet Detection (SD):

3) Boundary Detection (BD):

VI. Simulation Results

A. Transmitter

B. BPSK-Mode Receiver

C. QPSK-Mode Receiver

D. Mixed-Mode Receiver

VII. Experiment Results on SDR

The design is implemented in Vivado 2022.2. 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.

VIII. Discussions

A. Possible Enhancement

Frame structure design. CRC and/or checksums can be added to the frame structure.

Changing parameters on the fly. AXI peripheral [8] can be used to change the parameters in the `Constant_Config` on the fly, if the board allows.

B. Possible Extensions Beyond the Experiment

The training (TRN) field can be better utilized for additional experiments. For example, signal-to-noise (SNR) can be estimated at the TRN field.

Channel estimation algorithm [9], [10].

Auto generation [11].

IX. Conclusion

In this paper,

Appendix A
Block Diagrams

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, "BPSK/QPSK modulation and demodulation on FPGA with SDR," GitHub repository, 2024. [Online]. Available: https://github.com/Teddy-van-Jerry/sdr-psk-fpga

[4] 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

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

[6] 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.

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

[8] 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

[9] 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.

[10] 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.

[11] 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.