UDC 654.1:621.396.7
DOI https://doi.org/10.32782/2663-5941/2025.4.1/06

Ikhsanov Sh.M.
Admiral Makarov National University of Shipbuilding

Diakonov O.S.
Admiral Makarov National University of Shipbuilding

IMPLEMENTATION OF A REAL DATA TRANSMISSION CHANNEL BASED ON A SINGLE ADALM-PLUTO SDR PLATFORM

The paper analyzes the challenges of incorporating hardware platforms for software-defined radio (SDR) systems into the educational process while studying a range of disciplines related to electrical communication theory and the design of digital wireless systems.

Clearly, the use of these hardware platforms for educational purposes when studying wireless communication systems offers several advantages, such as a visual demonstration of how a radio system functions, the ability to conduct supplementary experiments related to radio wave propagation taking into account the hardware's drift characteristics, and so on.

The active learning module from Analog Devices, ADALM PLUTO SDR, was chosen as the hardware platform for the laboratory stand.

It is noted that in order to effectively study digital signal processing principles, a program framework should be created that includes the initialization code for the hardware platform, as well as the basic functionality of digital processing units used in real-world communication systems.

Using the demo programs provided with the MATLAB software, it has been demonstrated that they require significant refinement in order to create a working radio communication system.

Two signal processing approaches have been developed during the implementation of a QPSK-modulated data transmission system in order to address time synchronization issues.

A software implementation is provided for determining an additional shift in the signal constellation using Barker codes. This functionality can be used after initial synchronization and decoding to make a decision on whether to shift the signal constellation by a multiple of 90 degrees.

The results of the operation of the developed programs are presented, which demonstrate acceptable frequency shift reduction when synchronizing the receiver and transmitter within the same hardware platform ADALM PLUTO SDR. The shift estimation error is 0.01–0.03 Hz.

An analysis of the operation of an algorithm for estimating the frequency shift in the phase of a signal at different signal-to-noise ratios is presented. The operation of two synchronization algorithms (the algorithm for choosing between even and odd samples and the symbolic synchronization algorithm) for several transmitter power levels is also analyzed.

The data obtained allows us to conclude that error-free information transmission is possible in the data transmission channel under consideration with a signal-to-noise ratio of approximately 10 dB.

Key words: *ADALM PLUTO SDR, Software-Defined Radio, QPSK, symbol synchronization, phase synchronization, Education, Communication systems, Modulation.*

Formulation of the problem. To study wireless data transmission technologies more deeply, it is necessary to introduce software-defined radio (SDR) hardware platforms into the educational process, allowing for the implementation of a real communication channel with all digital signal processing units.

Analysis of recent research and publications. The recent development of SDR receivers and transmitters has made it possible to approach the study of digital wireless communications on a more advanced level. In [6], the authors shared their experience in creating a course that focuses on the practical aspects of synchronization, including automatic gain control, frequency offset, and synchronization – common challenges in real-world communication systems. The article [5] explores the technical specifications of the ADALM-PLUTO SDR hardware platform and its suitability for setting up laboratory exercises on

receiving FM signals and designing radar systems. The authors in paper [4] propose a methodology for learning the fundamentals of radio transmitters and receivers through practical experiments with commercial SDR modules. Therefore, it can be concluded that, for a deeper understanding of the fundamentals of wireless communication system operation, the use of computer simulations has a less tangible benefit than the use of dedicated SDR hardware platforms.

Task statement. The aim of the work is to develop a software-defined radio data transmission system based on the MATLAB software product and the ADALM-PLUTO SDR hardware platform from Analog Devices. This lab setup can be used for studying advanced topics in wireless data transmission.

Outline of the main material of the study. The ADALM-PLUTO SDR Active Learning Module is an easy-to-use tool that is based on the AD9363 chip [1, 2]. It has one receiving channel and one data transmission channel that operate in the frequency range of 325 MHz to 3.8 GHz with a sampling frequency of 65 kHz to 61 MHz, and a bandwidth of 200 kHz to 20 MHz for the tunable channel. Quadrature information is transmitted between the platform and PC via USB 2.0 interface. A 12-bit analog-to-digital converter (ADC) is used. With special settings, the operating frequency range can be expanded from 70 MHz to 6 GHz [2]. The external and internal views of the platform are shown in Figure 1.

Support for the ADALM-PLUTO software-defined radio (SDR) hardware platform is available in MATLAB [3]. The following MATLAB demonstration programs are available:

– plutoradioQPSKTransmitterExample.m;

– plutoradioQPSKReceiverExample.m;

– plutoradioWLANTransmitReceiveExample.m.

In turn, these projects are based on the use of the Communications System Toolbox Support Package for ADALM-PLUTO Radio [3]. The programs can be used with Matlab versions R2017 and above. The plutoradioWLANTransmitReceiveExample project was designed to run on a single ADALM-PLUTO SDR and functions normally. As indicated by the names of the first two programs, they enable information exchange between two ADALM-PLUTO SDR platforms. Unfortunately, this project was initially introduced in Matlab R2017 and not operated properly. There are software errors when connecting to the hardware platform, and information is almost always completely distorted on the receiver. The project version presented in Matlab R2019 performs somewhat better, but a reliable channel between the two ADALM-PLUTOs is still not established. Typically, bit error rate (BER) ranges from 0.1 to 0.3 regardless of the signal-to-noise ratio (SNR). The program transmits the phrase “Hello, world” with numbers from 000 to 999.

The final result of the program is as follows:

Error rate is = 0.162003.

Number of detected errors = 444083.

Total number of compared samples = 2741200.

It can be assumed that the final phase correction of the signal with a multiplicity of 90 degrees was not implemented or was implemented with errors. This type of signal processing will be discussed below.

Implementation of the transmitting program and organization of information reception. This is one of the initializations of the ADALM-PLUTO SDR platform used in the plutoradioWLANTransmitReceiveExample.m program:

a) b)

Fig. 1. The external (a) and internal (b) views of ADALM-PLUTO SDR

```
% Init ADALM PLUTO SDR:
USB = 'usb:0';
deviceNameSDR = 'Pluto'; % Set SDR Device
radio = sdrdev(deviceNameSDR); % Create SDR
device object
radio.RadioID = USB;
% Init transmitter:
sdrTransmitter = sdrtx(deviceNameSDR); % Create
transmitter object
sdrTransmitter.RadioID = USB;
% Init receiver:
sdrReceiver = sdrrx(deviceNameSDR); % Create
receiver object
sdrReceiver.RadioID = USB;
```

In most real-world information transmission channels, signals are sent in bursts of a relatively short duration, at the start of which a synchronization pulse is usually transmitted for receiver time synchronization. The Barker code, with a maximum length of 13, is often used for this purpose. Barker codes have a minimum side lobe level in the autocorrelation function of 1/N, where N is the length of the code.

```
% Create random data :
M=4;
infSize = 2^10;
txInf = randi([0 M-1],1,infSize);
% Insert Barker code before data packet:
Barker0 = [1,1,1,1,1,-1,-1,1,1,-1,1,-1,1];
% Bipolar Barker Code
BL=length(Barker0);
Barker=3*(1-Barker0)/2;   % coding [1,-1]
using symbols [0,3]
txInf = [Barker,txInf]';
% Generate QPSK Signal:
symbol_order='gray';% 'bin'(default)/'gray'
ini_phase0=pi/4; % initial phase in rad before
transmitting
txWaveform = pskmod(txInf,M,ini_phase0,symbol_
order);
% Set the signal parameters^
% Resample the transmit waveform at 400 KHz
fs = 100e3; % Transmit sample rate in Hz
osf = 4;    % Oversampling factor - QPSK
sdrTransmitter.BasebandSampleRate = fs*osf;
sdrTransmitter.CenterFrequency = 1.8500e9;
sdrTransmitter.ShowAdvancedProperties = true;
sdrTransmitter.Gain = -40;
```

The only thing that is more or less fundamental in this context is the value of the quantization frequency of the signal. This determines the duration of a single symbol, which is set to 10 microseconds in this example. Other parameters can be adjusted within acceptable limits for the ADALM-PLUTO SDR platform. The duration of a symbol can also be changed, but this may significantly affect the characteristics of the channel.

A prepared signal must pass through a Square-Root Raised Cosine Filter, implementing as an upsampling FIR-filter:

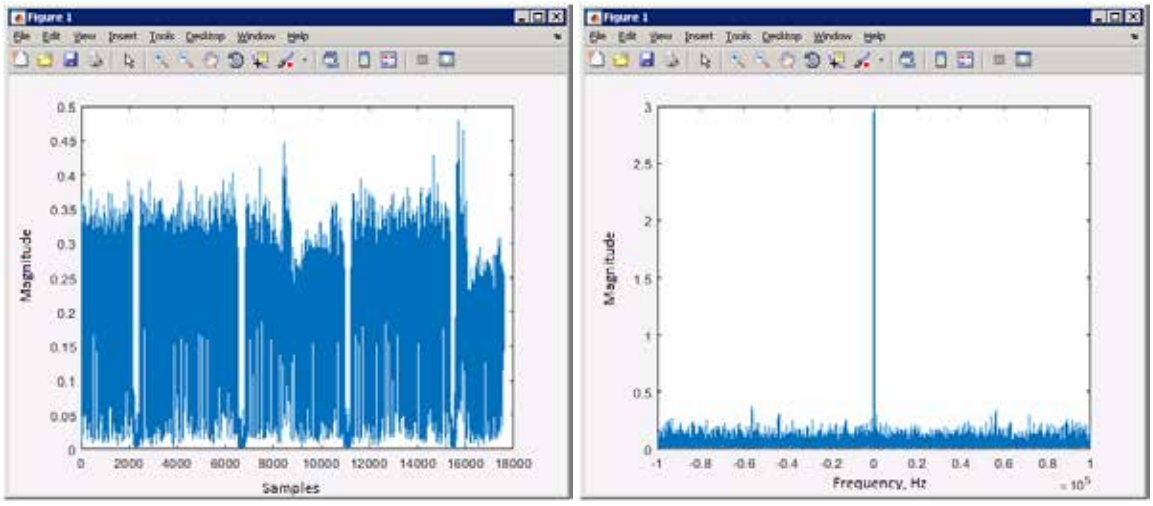
```
% Generate square root raised cosine transmit
filter:
RCFiltSpan = 10; % Filter span of Raised
Cosine Tx Rx filters (in symbols)
SquareRootRaisedCosineFilterOrder =
osf*RCFiltSpan;
RollOff = 0.5;
hTxFilt = fdesign.interpolator(osf, ...
          'Square Root Raised Cosine', osf, ...
          'N,Beta', SquareRootRaisedCosineFilt
erOrder, RollOff);
hDTxFilt = design(hTxFilt, 'SystemObject', true);
TransmitterFilterCoefficients = hDTxFilt.
Numerator/2;
% Filter and resample transmit waveform:
TransmitterFilter = dsp.FIRInterpolator(osf,Tr
ansmitterFilterCoefficients);
txWaveform = TransmitterFilter(txWaveform);
```

The filter used in this experiment has a length of 41 samples. The filter coefficients for this filter are shown in the following graph (Fig. 2).

The filter increases the quantization frequency by a factor of 4. Therefore, for each of the 1037 bits (13 bits from the Barker code and 1024 information bits), four complex samples are generated in the program. The total number of samples prepared is therefore 4148. Additionally, the signal payload is padded with a pause equal to 1/16th of the signal's length payload and is transmitted continuously (in a cyclic manner):

```
% Add pause to signal payload:
kPause=infSize/4; % Since there are 4 samples
per symbol, the pause is 1/16 of the duration
of the signal
txWaveform(end+1:end+kPause)=0;
PacketLen=length(txWaveform);
% Cyclic infinite signal transmission:
sdrTransmitter.transmitRepeat();
```

At the receiver, the signal is received within the time required to receive a certain number of copies of the transmitted message. We will set the reception time so that at least three packets of information are received. Now, we do not need to artificially introduce a frequency shift. It is enough to separate the frequencies of the transmitter and receiver. This is done in the program using the df parameter. The receiver uses the same quantization frequency as the transmitter.



a) b)

Fig. 2. Upsampling FIR-filter unnormalized impulse response

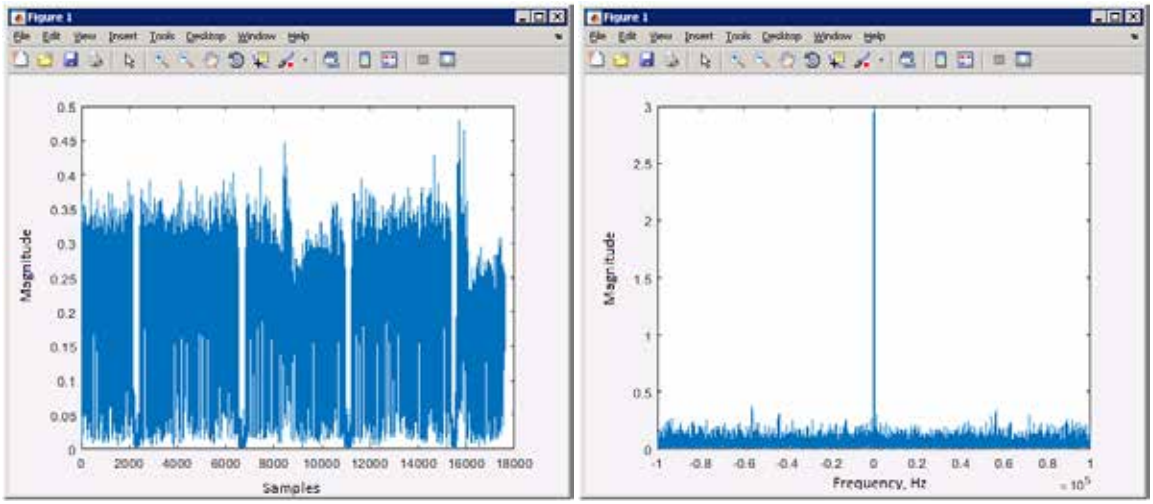
```
df=20000; % Shift of the receiver carrier
frequency relative to the transmitter (Hz)
sdrReceiver = sdrrx(deviceNameSDR);
sdrReceiver.RadioID = USB;
sdrReceiver.BasebandSampleRate =
sdrTransmitter.BasebandSampleRate;
sdrReceiver.CenterFrequency = sdrTransmitter.
CenterFrequency+df;
sdrReceiver.GainSource = 'AGC Slow Attack'
sdrReceiver.OutputDataType = 'double';
kPacket=4; %
sdrReceiver.SamplesPerFrame =
PacketLen*kPacket;
% The transmitted waveform is captured using
the PlutoSDR
burstCaptures = sdrReceiver();
bCaptAbs=abs(burstCaptures);
% Received signal magnitude:
plot(bCaptAbs);
```

Automatic gain control slow attack is set for the received signal. The length of the received frame is 17616 complex samples, which includes the pause. The emitted signal has a length of 4404 samples. The magnitude of the received signal is shown in Fig. 3a.

The figure shows the pauses between message packets and significant unevenness in the magnitudes of the received signal, even within a single packet.

Primary elimination of the frequency shift across the signal spectrum. The received signal is passed through a 2-fold step-down receiving FIR-filter and, before the zones of individual packets are allocated, it enters the first stage of eliminating frequency shift. At this stage, the frequency shift will be estimated from the signal spectrum using the FFTFreqSynchr() function:

```
function [ dfest ] = FFTFreqSynchr(yM,
SampleRate)
% yM - the signal after removing information
phase jumps %(usually by exponentiating the
signal)
% SampleRate - quantization frequency in Hz
% dfest - frequency shift in Hz
Fs = SampleRate;
LyM=length(yM); % Signal spectrum
NFFT = 2^nextpow2(LyM); % number of FFT
filters
FM=fft(yM,NFFT); % spectrum
dFFT=Fs/NFFT;  % FFT Filter Arrangement (Hz)
f=(0:dFFT:Fs-dFFT)'; % frequency axis marking
FMmod=abs(FM); % Magnitude spectrum of the
signal
% Define filter with maximum magnitude
[FMmodmax,kmax]=max(FMmod);
FMmod=FMmod/FMmodmax; % Normalization of the
signal to the
% maximum
% Draw a parabola through the maximum point of
the spectrum and
% two adjacent samples (filters f(1) and
f(NFFT) are adjacent)
if kmax == 1
   p=polyfit([-dFFT f(1:2)']',[FMmod(NFFT)
FMmod(1:2)']',2);
else
   f1=f-f(kmax-1); % shifting the frequency
axis to facilitate
% calculations in the least squares method
   if kmax == NFFT
      p=polyfit([f1(NFFT-1:NFFT)'
f1(NFFT)+dFFT]',...
                [FMmod(NFFT-1:NFFT)'
FMmod(1)]',2);
```



a) b)

Fig. 3. Received signal magnitude (a), The spectrum of the fourth degree of the received signal (b)

```
   else
      p=polyfit(f1(kmax-1:kmax+1),FMmod(kmax-
1:kmax+1),2);
   end
end
dfest=-p(2)/(2*p(1)); % Estimation of
displacement as
% the vertex of a parabola:
if kmax > 1
    dfest=dfest+f(kmax-1);
end
if dfest > Fs/2
    dfest=dfest-Fs;
end
end
```

All functions used in this article were developed by the authors. In each specific case, the use of MATLAB library functions is described.

```
% Measuring and eliminating frequency shift:
SampleRate=sdrReceiver.BasebandSampleRate/2;
RxLen=length(RCRxSignal);
dt=1/SampleRate;
t=(0:dt:dt*RxLen-dt)';
RCRxSignalM = RCRxSignal.^M;
dfestSP = FFTFreqSynchr(RCRxSignalM,
SampleRate)/M;
fprintf('dfestSP0=%.4f.\n',dfestSP);
omeg=-2*pi*dfestSP;
RCRxSignal = RCRxSignal.*exp(1i*omeg*t);
FrameLen = length(RCRxSignal);
```

The spectrum of the fourth degree of the signal for the entire frame is shown in Figure 3b. Instead of the interval [0, Fs], the spectrum is now linked to the interval [-Fs/2, Fs/2].

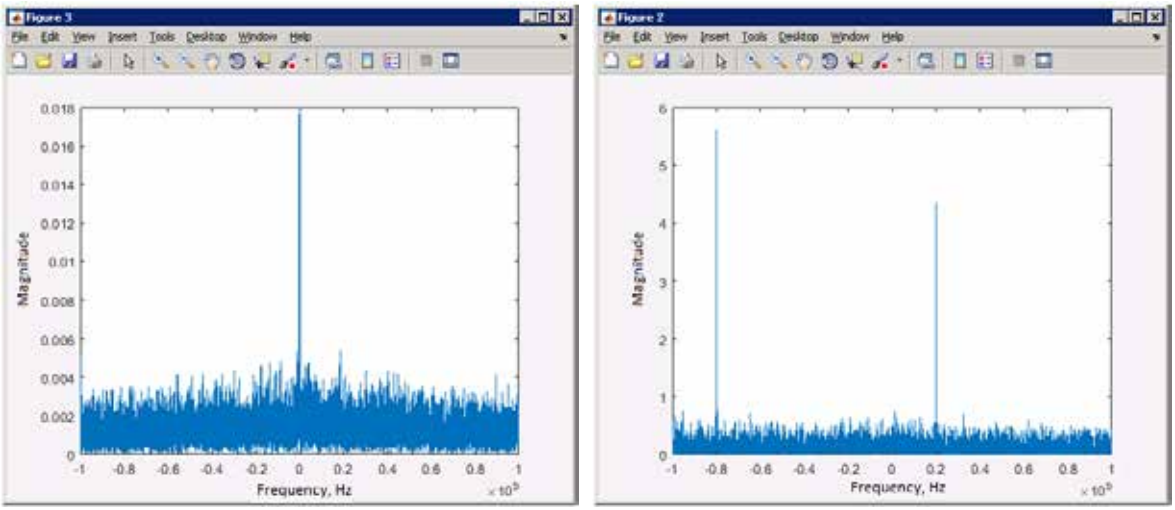
The shift estimate was -0.0138 Hz, which indicates a good match between the frequencies of the transmitter and receiver in the same ADALM PLUTO SDR device. With this signal-to-noise ratio, there are no bit errors in the channel. As with modeling, the estimate is practically independent of the signal-to-noise ratio.

Reducing the power of the transmitted signal by 14 dB leads to a spectrum as shown in Fig. 4a, with a BER of 10^{-1}.

For this spectrum, the estimated offset was -0.162 Hz. Adjusting the receiver frequency to the right should lead to the observation of a negative offset four times the difference, while adjusting to the left would lead to a positive offset. Indeed, when shifting to +20 kHz with good signal power (transmitter signal attenuation of 40 dB), we see the following spectrum of the fourth degree of the signal (Fig. 4b).

Our signal is observed at -4·20 kHz, as expected. The appearance of a signal at 20 kHz is associated with pauses between packets, and it is located 100 kHz away from the main signal. In the spectrum of the signal, this distance is 25 kHz, in accordance with the ratio of packet length and pause. The estimated frequency offset was -20000.96 Hz, with a slight increase in discrepancy relative to the previous case, probably due to nonlinearities in the hardware implementation of ADALM PLUTO SDR.

Allocation of zones using a convolution signal with a Barker code. Zones are assigned using the ConvBarker function.



a) b)

Fig. 4. The spectrum of the 4th degree of the received signal when the power is reduced by 14 dB (a), the spectrum of the 4th degree of the received signal when the receiver frequency is shifted by 20 kHz (b)

```
% Convolution with a Barker code:
ConvSignal = ConvBarker(RCRxSignal,ReceivOsf);
```

The text of the ConvBarker()function is as follows:

```
function ConvSignal = ConvBarker(RCRxSignal,
ReceivOsf)
%% Convolution function of a complex signal
with a Barker code
% ReceivOsf - the number of samples per symbol
in the received signal
% (Oversampling factor for received signal)
Barker0 = [1,1,1,1,1,-1,-1,1,1,-1,1,-1,1]; %
Bipolar Barker Code
BL=length(Barker0);
BL2 = ReceivOsf*BL;
Barker2=1:BL2; % Multiplying the Barker code
by the number of samples per symbol:
for i=1:BL
    Barker2((i-1)*ReceivOsf+1:i*ReceivOsf)=
Barker0(i);
end
% Convolution as the product of a row per
column:
FrameLen = length(RCRxSignal);
ConvSignal=(1:FrameLen-BL2+1)';
for i=1:FrameLen-BL2+1
    ConvSignal(i) =
Barker2*RCRxSignal(i:i+BL2-1);
end
end
```

The result of the convolution process is shown in Figure 5. Now, the beginning of each packet is clearly visible, and we can easily determine the number of packets. To do so, it is sufficient to call the specially designed SearchPackets() function.

```
[Mind,kFoundPackets] = SearchPackets(ConvAbs,
kPacket,PacketLen,kPause);
```

The text of the package search function is as follows:

```
function [Mind,kFoundPackets] = SearchPackets
(ConvAbs,n,PacketLen,kPause)
%% A function for searching for packets with
the Barker code at
% the beginning ConvAbs - the amplitude of the
complex signal
```

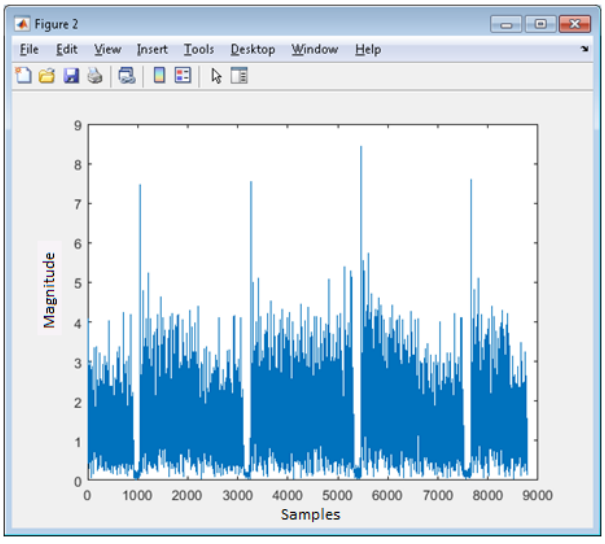


Fig. 5. The result of the convolution of the received information packets with the Barker code

```
% convoluted with the Barker code
% n - number of packets to search for
% PacketLen – packet length, kPause - pause
between packets
% kFoundPackets - number of packets found
% Mind - packets start indexes
Pqs=8; % threshold for the Barker convolution
relative to the
% signal area (dB)
Pqp=15; % threshold for the Barker convolution
relative to the
% pause zone (dB)
ConvLen=length(ConvAbs);
% Searching for a set number of maximum
samples in a signal:
[MConvSignal,Mind] =
fewmax(ConvAbs,n,PacketLen);
qs(1:n)=0; % the ratio of the maximum reading
to the average
% level in a packet (dB)
qp(1:n)=0; % the ratio of the maximum count to
the average level
% in the pause (dB
kFoundPackets=0;
for i=1:n
    i1=Mind(i)+1; i2=i1+PacketLen-2;
    j2=i1-2; j1=max(j2-kPause/2+1,1);
    if i2 <= ConvLen
        qs(i)=20*log10(MConvSignal(i)/
mean(ConvAbs(i1:i2)));
    else
        qs(i)=0;
    end
    if j2-j1 >= 0
        qp(i)=20*log10(MConvSignal(i)/
mean(ConvAbs(j1:j2)));
    else
        qp(i)=0;
    end
    fprintf('qs=%0.2f, qp=%0.2f\
n',qs(i),qp(i));
    if qp(i) >= Pqp && qs(i) >= Pqs
        kFoundPackets = kFoundPackets+1;
        Mind(kFoundPackets) = Mind(i);
    end
end
Mind = Mind(1:kFoundPackets); Mind =
sort(Mind);
End
```

First of all, we note that the function searches for packets of known length with a given pause length between them. The program input receives the amplitude of the signal convoluted with the Barker code. First, the beginning of each packet is determined by the maximum samples. To do this, the fewmax() function is written, which searches for a given number of maximum elements in a valid array. To avoid a situation where there may be several maxima on one packet, the maximum found is excluded from further search along with PacketLen samples on the right and dl (assumed to be 5) samples on the left. The number of required maxima corresponds to the maximum number of packets that can be received in one frame. The main criterion for selecting a packet is a sufficient level of convolution with the Barker code relative to the average level of packet magnitudes and the presence of a sufficiently deep pause after the packet (8 and 15 dB, respectively, recall that for the Barker code used, the theoretical level of the side lobes does not exceed 20lg(1/13)= -22 dB).

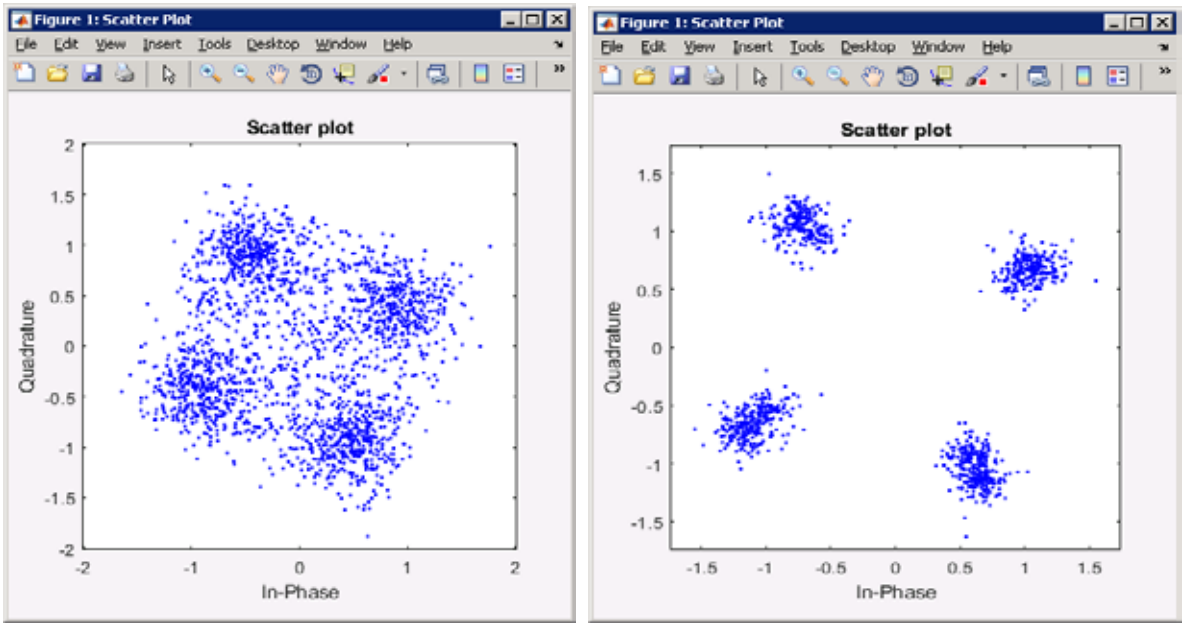
After selecting packet zones to prevent exceeding the dynamic range, the signal is normalized to the average as well as calculating the average signal-to-noise ratio in the pauses between packets. The last characters in each packet are distorted by convolution with a pause during filtering and so they should be removed.

```
% Normalization of the signal to the average
level
% and deletion of the last characters:
RCRxSignal=RCRxSignal/mean(abs(RCRxSignal));
SignalM=RCRxSignal.^M;
NFFT=PacketLen0/4-5; % the last ~10 samples in
each packet of the RCRxSignal filtered array
are distorted
due to convolution with pause
NFFT1=2*NFFT;
SampleRate=sdrReceiver.BasebandSampleRate/4;
% Calculating the average signal-to-noise
ratio:
MeanNoise = CalcNoise(abs(RCRxSignal),Mind,kFo
undPackets,PacketLen,kPause);
```

Packet time synchronization (symbol synchronization). Further processing is carried out separately for each packet. If we build a signal constellation for individual packets at this stage, we will see that the signal, in addition to the resulting phase shift, is practically scattered over the entire complex plane, despite a significant SNR of about 14 dB (Fig. 6).

It should be noted that the program uses a 45-degree rotation of the phase plane (in_phase0 parameter), so the centers of the symbols of the transmitted signal are located on the bisectors of the coordinate quarters.

Let's consider two signal processing options that are aimed at eliminating time synchronization violations. In the first variant, we will use the MATLAB comm.SymbolSynchronizer() function, in the second variant, we will consider the method widely used in



a) b)

Fig. 6. The signal constellation of one of the packets before phase shift compensation and time synchronization (a), the signal constellation of one of the packets after time synchronization by the Matlab function comm.SymbolSynchronizer() (b)

practice to select the best of the 2 available samples per symbol.

The time synchronization function (symbol synchronization) is initialized as follows:

```
% Create symbol synchronizer
symSync = comm.SymbolSynchronizer( ...
      'SamplesPerSymbol', 2, ...
      'DampingFactor', sqrt(2)/2, ...
      'NormalizedLoopBandwidth', 0.02);
```

We use it when there are two samples for each symbol, and the function returns a signal with one sample per symbol. Unfortunately, there are two minor flaws with the function, which are easily fixed, fortunately. The function does not work well on the first call, so you will need to call it first before using its results. Additionally, it gives you an extra first sample, which we will remove. Working with the function is as follows:

```
% Symbol synchronizer:
    if prRepead % the result is bad
        PacketOpt0 = symSync(Packet); % on the
first call.
        prRepead=0;
    end
    PacketOpt1 = symSync(Packet);  % removing
the shift
    PacketOpt1=PacketOpt1(2:end);  % by 1
symbol
```

Here, **Packet** is the complex samples in a packet obtained after determining the boundaries of each packet using the Barker code:

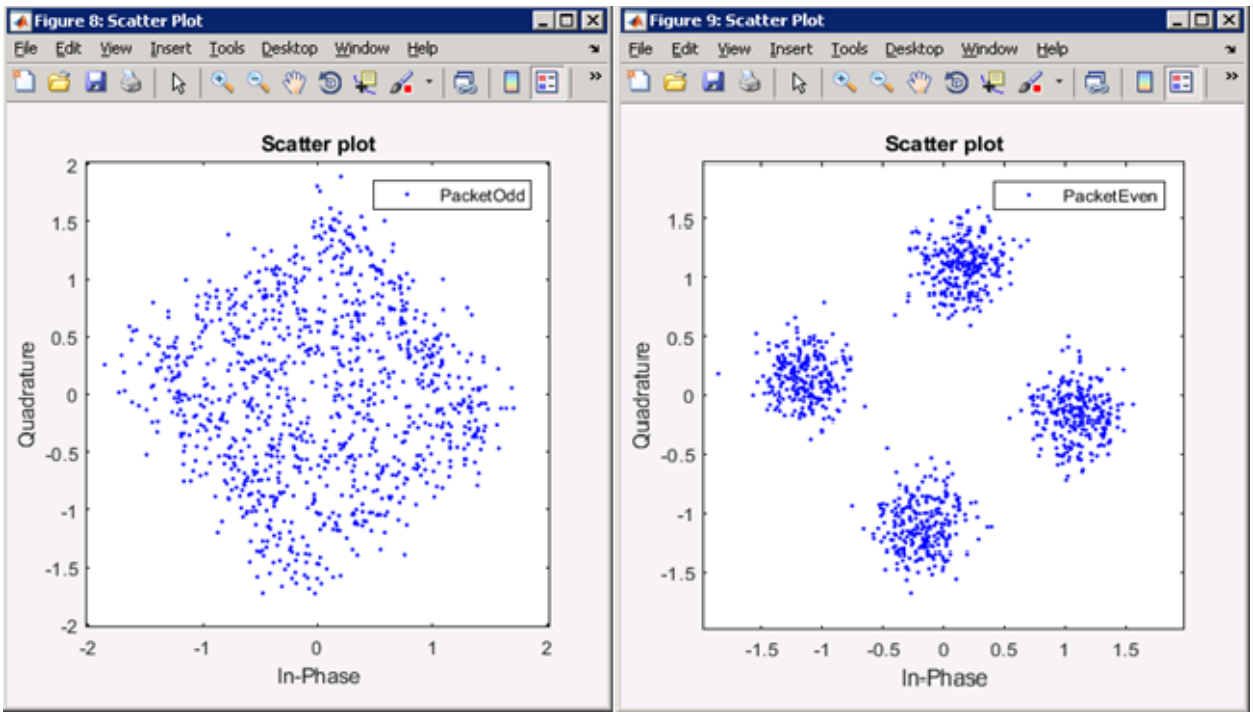
```
Packet=RCRxSignal(Mind(i):Mind(i)+NFFT1-1);
```

The result of the function is clearly visible on the signal constellation (Fig. 6, b). All the symbols are compactly assembled, leaving only a phase shift of about 10 degrees.

To explain the second method, we will present the signal constellation separately for even and odd samples. Since we have two samples per symbol, one of them should be closer to the center of the transmitted symbol, and therefore, more accurately convey information about the symbol. This is clearly visible in the signal constellation of even and odd samples (Fig. 7).

Here, all four symbols are compactly arranged for even samples. The phase shift is slightly larger, if we mean by the phase shift an angle of up to 45 degrees, the rotation of which places the centers of gravity of the groups of symbols along the bisectors of the coordinate quarters. It is clear that this does not guarantee the location of the groups of symbols in their quarters, you may have to rotate the phase by an angle of 90, 180 or 270 degrees. At the same time, odd samples carry practically no information.

It remains for each packet to choose between even and odd samples. In principle, this can be done using



a) b)

Fig. 7. The signal constellation of one of the packets for odd (a) and even (b) samples

the signal constellation or the unwrapped phase. We believe that the simplest and most reliable approach is to use the magnitude of the signal after removing the phase wrap. Figure 8 shows the magnitude of the 4th degree of the signal for odd and even samples separately for one of the packets. Note that different scales were chosen for even and odd samples in this figure.

To measure the preference for odd or even samples, we can look at the minimality of their variances. In Figure 8, the ratio of odd symbol variance to even sample variance was 3.45 for the magnitudes.

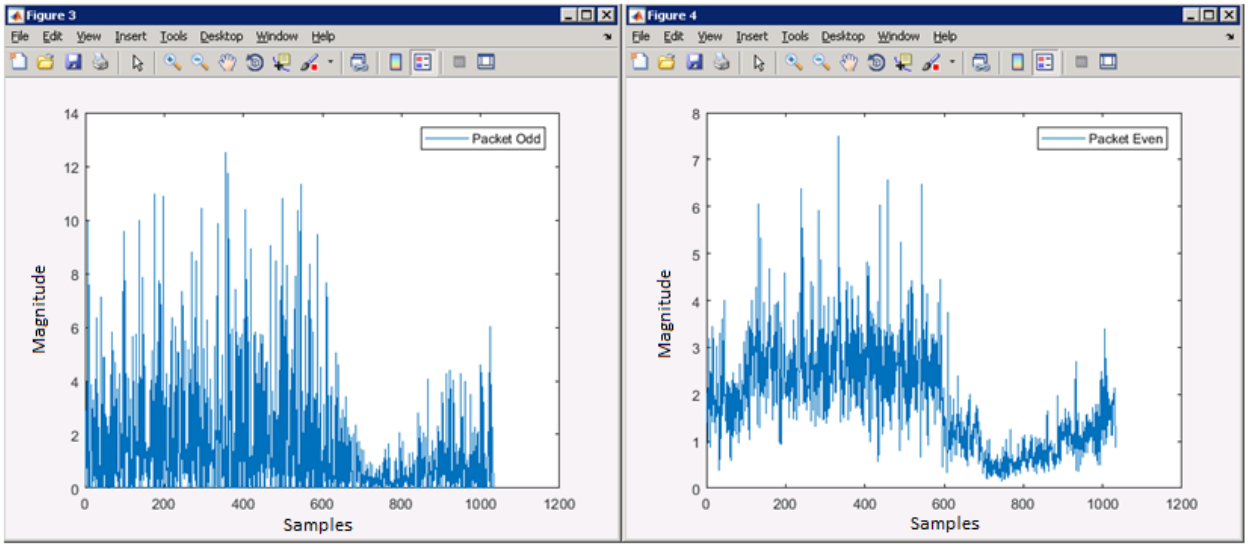
The software implementation of the decision is as follows:

```
% Choosing between even and odd samples:
    PacketM=SignalM(Mind(i):Mind(i)+NFFT1-1);
    PacketOddM=PacketM(1:2:end);
    PacketEvenM=PacketM(2:2:end);
    varOdd=var(abs(PacketOddM));
    varEven=var(abs(PacketEvenM));
    if varOdd <= varEven
        prOdd=1;
    else
        prOdd=0;
    end
    Packet=RCRxSignal(Mind(i):Mind(i)+N
FFT1-1);
    if prOdd
        PacketMopt=PacketOddM;
PacketOpt=Packet(1:2:end);
        Kvar=varEven/varOdd;
    else
        PacketMopt=PacketEvenM;
PacketOpt=Packet(2:2:end);
        Kvar=varOdd/varEven;
    End
```

Phase synchronization. After synchronizing the symbols, we will once again estimate the residual frequency shift for each packet, and for comparison, we will do this both for the spectrum and for the phase:

```
% Calculating frequency shift from spectrum:
  dfestSP = FFTFreqSynchr(PacketMopt,
SampleRate)/M;
% Calculating the frequency shift from the
unwrapped phase:
  PacketOptM = PacketOpt.^M;
  PhaseOpt = phaseCalc(PacketOptM);
  dfestPH = PhaseFreqSynchr(PhaseOpt,
SampleRate)/M;
```

The phase rotation angle can be calculated based on the center of gravity of the 4 groups of transmitted symbols. We will use the MATLAB function comm. CarrierSynchronizer(), which, in addition to the phase synchronization, performs synchronization based on



a) b)

Fig. 8. Magnitude of one of the packets for odd (a) and even (b) samples

the carrier frequency shift. The function descriptor is created as follows:

```
% Create frequency and phase synchronizer
    carrSync = comm.CarrierSynchronizer( ...
    'DampingFactor', 0.707, ...
    'NormalizedLoopBandwidth', 0.01, ...
    'SamplesPerSymbol', 1,...
    'Modulation','QPSK',...
    'ModulationPhaseOffset','Auto');
```

To compare the symbol synchronization options, the program implements the ability to process a frame in parallel with both options. Therefore, the use of the comm.CarrierSynchronizer() function looks like this:

```
    if prOpt
        fineCompPacketOpt =
carrSync(PacketOpt);
        Legend='Odd/Even';
    else
        fineCompPacketOpt = carrSync(Packet
Opt1);
        Legend='Symbol Synchronizer';
    End
```

Figure 9 shows the packet signal constellations for two implementations. If in the first implementation a slightly better result was obtained using the Matlab function comm.SymbolSynchronizer(), then in the second implementation we see that its use led to the scattering of some symbols, which is not observed when using the choice of even/odd samples.

As mentioned above, the results obtained do not guarantee that the phase shift has been completely eliminated. Rotating the phase plane by multiples of 90 degrees would lead to the same qualitative signal constellation. Using only the samples of the signal alone, we would not be able to detect such rotations in any way. This is where the Barker code, which is placed at the beginning of each packet, comes to our aid. To do this, we will need to decode the first 13 bits using additional phase rotations of 0, 90, 180, and 270 degrees, and compare them to the Barker code in use. The rest of the packet will be decoded using a phase rotation that will result in the fewest errors. The function that implements this process looks like this:

```
function [PhSp,mind] =
PhaseByBarker(BarkerIn,Barker)
%% The function of searching for phase
rotation
% by matching the Barker code:
% BarkerIn - received Barker code in QPSK
symbols
% Barker – initial Barker code in QPSK symbols
% PhSp - required phase rotation in radians
% mind - the number of error bits in the
Barker code
% for the selected rotation
M=4; % for QPSK
% Barker code conversion table for
0,pi/2,pi,3*pi/2 degree
% phase rotation
BarkerSp=[0,1,2,3; 1 3 0 2; 3 2 1 0; 2 0 3 1];
BL=length(BarkerIn); BarkerInSp(1:M,1:BL)=0;
```

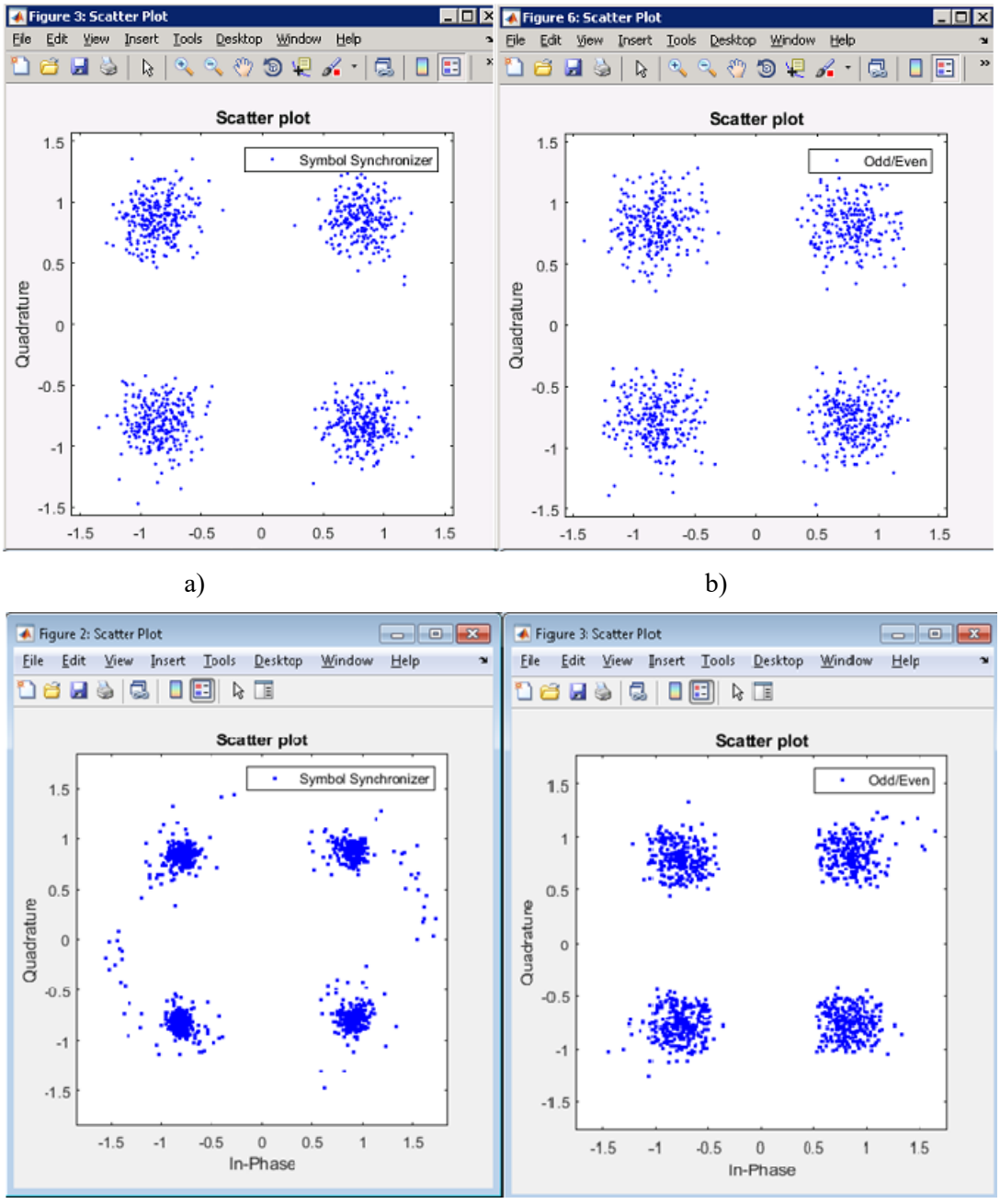


Fig. 9. The signal constellation of the packet after phase synchronization by the comm.SymbolSynchronizer() function (c) and the selection of even/odd samples (d); (a), (b) – before synchronization, respectively

```
dSp(1:M)=0;
for i1=1:M
    for j1=1:BL
        BarkerInSp(i1,j1)=BarkerSp(i1,BarkerI
n(j1)+1);
    end
[dSp(i1),~] = biterr(BarkerInSp(i1,:),Barker);
end
[mind,indPhSp]=min(dSp); PhSp=(indPhSp-
1)*pi/2;
End
```

Recall that the original Barker bipolar code [1,1,1,1,1,-1,-1,1,1,-1,1,-1,1] we encode with QPSK symbols 0, 3 and get the following sequence [0,0,0,0,0,3,3,0,0,3,3,0] (0 corresponds to the 1st quarter, 1–2nd quarter, 3–3rd quarter and 2–4th quarter in the Gray encoding that we use). Now we need to create a table of symbol transformations during phase rotations. In the program, it is represented by the BarkerSp matrix, the i^{th} row of which corresponds to the rotation of the phase counterclockwise by an angle of $\pi/2*(i-1)$ radians.

In the function, we count not symbol failures, but bit failures for each rotation.

The function is called in the program after the initial phase synchronization and decoding with the following lines:

```
% Packet Demodulation:
rxInf = pskdemod(fineCompPacketOpt,M,ini_
phase0,symbol_order);
% Searching for phase rotation by matching the
Barker code:
 BarkerIn=rxInf(1:BL);
 [PhSp,mind] = PhaseByBarker(BarkerIn,Barker);
```

Now we can decode the packets taking into account the phase rotation found:

```
% Demodulation of a packet after phase
rotation:
ini_phase=ini_phase0-PhSp;
rxInf = pskdemod(fineCompPacketOpt,M,ini_
phase,symbol_order);
```

Processing of received information is completed by calculating the BER for each information packet:

```
NN = length(rxInf);
[numErr,BER] = biterr(txInf(1:NN),rxInf);
fprintf('PhSp=%.0f градусов, min_d=%3d\
n',PhSp*180/pi,mind);
fprintf('numErr=%4d,BER=%7.5f\n\
n',numErr,BER);
```

Before finishing the program we need to release the ADALM PLUTO SDR platform and the objects used:

```
% Release the state of sdrTransmitter,
sdrReceiver and objects:
release(sdrTransmitter);
release(sdrReceiver);
release(symSync);
release(carrSync);
```

Analysis of the program operation. To analyze the operation of various blocks, we will run the program twice with the same parameters at close time intervals (Table 1).

The established frequency offset on the receiver of 50 Hz relative to the transmitter was estimated with good accuracy (at the first launch the error was about 0.03 Hz, in the second case – less than 0.01 Hz). This indicates good frequency matching between the receiver and transmitter within the Pluto platform due to the use of a single local oscillator for both the receiver and transmitter. However, after reducing the sampling frequency to 100 kHz, a small frequency offset is recorded, not exceeding 0.4 Hz. Such a frequency shift with the accepted symbol length and packet duration has practically no effect on the characteristics of information transmission. Moreover, it will be further reduced by the function comm.CarrierSynchronizer(), which adjusts the frequency and phase. To verify this, we will additionally measure the frequency shift after the specified function. The results of such a measurement in one of the program launches are given below (dfestSP is the shift before re-tuning, dfestSP1 is after) – Table 2.

The residual offset does not exceed 0.45 Hz.

The algorithm for estimating the frequency shift in the signal phase with large SNR, which were implemented in the above launches, gives approximately the same shifts, however, with a decrease in signal power, as in modeling, it produces unacceptable results (the dfestPH parameter) – Table 3.

The energy of the signal during the second launch was approximately 2 dB lower than the first, although the time interval between the two launches was no more than 10 seconds. This could be due to interference or instability on the Adalm Pluto SDR platform. Using the SymbolSynchronizer() function and an algorithm to choose between even and odd sample values resulted in completely error-free data transmission. Additionally, odd samples were found to be more effective for the first launch and even samples for the second launch. After the function comm. CarrierSynchronizer() it had to rotate the phase by 180 degrees at the first start, and by 270 degrees in the second case, regardless of the algorithm used. This phase shift is the same for all packages within a single launch.

In conclusion, it remains for us to compare the two synchronization algorithms with lower SNRs. The results are shown in Table 4.

As can be seen from the table above, in two starts when the transmitter signal was attenuated to -48 dB, the algorithm for choosing between even and odd samples turned out to be about an order of magnitude worse in terms of BER, however, in the next two starts when attenuating to -50 dB, the SNR dropped significantly (about 9 and 12 dB). At 12 dB, the algorithms equaled the BER level, and at 9 dB using the SymbolSynchronizer() function led to the complete destruction of the channel, while the automatic selection of even or odd samples provided an acceptable level of failures that can be eliminated by other methods, in particular, using error correction codes. The data obtained also allows us to conclude that error-

Table 1

Program operation log

First launch	Second launch
Generating QPSK transmit waveform at: Transmitter.SampleRate=400000 Hz Transmitter.CenterFrequency=1850000000 Hz Transmitter.Gain=-45 dB Starting a new RF capture at: Receiver.CenterFrequency=1850000050 Hz Receiver.GainSource=AGC Slow Attack dfestSP0=-49.9704. Number of found packets = 3 prOpt=0. % **0 - Symbol Synchronizer** Packet #1,SNR=18.0 dB,prOdd=1,Kvar=3.32. dfestSP=0.3391 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #2,SNR=18.2 dB,prOdd=1,Kvar=2.97. dfestSP=0.5545 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #3,SNR=17.8 dB,prOdd=1,Kvar=3.16. dfestSP=0.4120 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000 prOpt=1. % **1 - Odd/Even choice** Packet #1,SNR=18.0 dB,prOdd=1,Kvar=3.32. dfestSP=0.3391 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #2,SNR=18.2 dB,prOdd=1,Kvar=2.97. dfestSP=0.5545 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #3,SNR=17.8 dB,prOdd=1,Kvar=3.16. dfestSP=0.4120 PhSp=180 degrees, min_d= 0 numErr= 0,BER=0.00000	Generating QPSK transmit waveform at: Transmitter.SampleRate=400000 Hz Transmitter.CenterFrequency=1850000000 Hz Transmitter.Gain=-45 dB Starting a new RF capture at: Receiver.CenterFrequency=1850000050 Hz Receiver.GainSource=AGC Slow Attack dfestSP0=-49.9924. Number of found packets = 3 prOpt=0. % **0 - Symbol Synchronizer** Packet #1,SNR=16.9 dB,prOdd=0,Kvar=2.62. dfestSP=-0.0116 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #2,SNR=16.2 dB,prOdd=0,Kvar=3.93. dfestSP=0.3297 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #3,SNR=16.1 dB,prOdd=0,Kvar=2.60. dfestSP=-0.1924 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000 prOpt=1. % **1 - Odd/Even choice** Packet #1,SNR=16.9 dB,prOdd=0,Kvar=2.62. dfestSP=-0.0116 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #2,SNR=16.2 dB,prOdd=0,Kvar=3.93. dfestSP=0.3297 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000 Packet #3,SNR=16.1 dB,prOdd=0,Kvar=2.60. dfestSP=-0.1924 PhSp=270 degrees, min_d= 0 numErr= 0,BER=0.00000

Table 2

Program operation log

prOpt=0. Packet #1,SNR=19.0 dB,prOdd=0,Kvar=4.19. dfestSP=3.9517, dfestSP1=0.0966. PhSp=270 **degrees**, min_d= 0 numErr= 0,BER=0.00000 Packet #2,SNR=18.7 dB,prOdd=0,Kvar=5.84. dfestSP=1.8721, dfestSP=0.3524. PhSp=270 **degrees**, min_d= 0 numErr= 0,BER=0.00000	Packet #3,SNR=18.7 dB,prOdd=0,Kvar=5.12. dfestSP=1.7735, dfestSP=0.4296. PhSp=270 **degrees**, min_d= 0 numErr= 0,BER=0.00000

Table 3

Program operation log

SNR 16 dB	SNR 9 dB
Packet #1,SNR=16.8 dB,prOdd=0,Kvar=3.56. dfestSP=-0.6092, dfestPH=-0.9937. PhSp=180 **degrees**, min_d= 0 numErr= 0,BER=0.00000	Packet #1,SNR=9.0 dB,prOdd=0,Kvar=1.32. dfestSP=-0.0094, dfestPH=82.6901. PhSp=270 **degrees**, min_d= 2 numErr= 20,BER=0.00970
Packet #2,SNR=16.6 dB,prOdd=0,Kvar=3.58. dfestSP=0.0151, dfestPH=1.0546. PhSp=180 **degrees**, min_d= 0 numErr= 0,BER=0.00000	Packet #2,SNR=9.5 dB,prOdd=0,Kvar=1.39. dfestSP=-1.0198, dfestPH=-10.3491. PhSp=270 **degrees**, min_d= 0 numErr= 20,BER=0.00970
Packet #3,SNR=16.7 dB,prOdd=0,Kvar=3.80. dfestSP=-0.6271, dfestPH=-0.4760. PhSp=180 **degrees**, min_d= 0 numErr= 8,BER=0.00388	Packet #3,SNR=9.5 dB,prOdd=0,Kvar=1.53. dfestSP=-0.4988, dfestPH=6.4755. PhSp=270 **degrees**, min_d= 0 numErr= 2,BER=0.00097

Table 4

Results of comparison of synchronization algorithms

	Number of errors			Number of errors	
SNR (dB)	**Symbol Synchronizer**	**Odd/Even Choice**	**SNR (dB)**	**Symbol Synchronizer**	**Odd/Even Choice**
Transmitter.Gain=-48 dB					
17,0	0	0	14,7	0	5
16,9	0	1	15,8	1	1
16,3	0	7	15,6	0	3
Total:	**0**	**8**	**Total:**	**1**	**9**
Transmitter.Gain=-50 dB					
9,3	1062	17	12,3	9	11
9,4	1061	7	12,3	6	6
8,2	1058	15	12,5	14	13
Total:	**3181**	**39**	**Total:**	**29**	**30**

free information transmission is possible in the considered data transmission channel with a SNR of about 10 dB. Remind that in this case we are talking about the SNR over the entire quantization frequency range, so the SNR measured at the maximum of the amplitude spectrum will be significantly higher.

Conclusions. The use of the ADALM PLUTO SDR hardware and MATLAB software as a laboratory stand allows us to explore all stages of digital signal processing in the creation of modern software-defined wireless radio systems. This paper demonstrates the creation of a QPSK-modulated data transmission system. Synchronization algorithms for real data transmission systems are presented. This technique can be used in the education of university students in the framework of educational programs in the field of telecommunications.

Bibliography:

1. Introduction to ADALM-PLUTO. URL: https://wiki.analog.com/university/tools/pluto/users/intro (дата звернення 15.07.2025).

2. Hands-on workshop: Software-defined radio. Analog Devices. URL: https://wiki.analog.com/_media/richardsonmathworksseminars.pdf (дата звернення 15.07.2025).

3. ADALM-PLUTO Radio Support from Communications Toolbox. MathWorks. URL: https://surl.li/rgeryw (дата звернення 15.07.2025).

4. L. A. Camuñas-Mesa, J. M. De La Rosa. Using Software-Defined Radio Learning Modules for Communication Systems. 2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference), Teruel, Spain. 2022. P. 1–6. DOI: 10.1109/TAEE54169.2022.9840692

5. S. B. S. Hanbali. Low-cost software-defined radio for electrical engineering education. IEEE Potentials. 2023. Vol. 42. №. 5. P. 13–19. DOI: 10.1109/MPOT.2022.3223788

6. M. Rice, M. McLernon. Teaching communications with SDRs: Making it real for students. IEEE Commun. Mag. 2019. Vol. 57. №. 11. P. 14–19. DOI: 10.1109/MCOM.001.1900185.

7. S. Yogitha and D. Raju. Design And Implementation Of QPSK Modulation System. 2017. URL: https://api.semanticscholar.org/CorpusID:128355114 (дата звернення 15.07.2025).

8. Іхсанов Ш.М., Рябенький В.М., Дьяконов О.С. Дослідження сигналів реальних інформаційних систем з використанням приймачів RTL-SDR: навч. посіб. Миколаїв, 2019. 184 с.

9. W. Stewart, W. Barlee, S. W. Atkinson, H. Crockett. Software Defined Radio using MATLAB & Simulink and the RTL-SDR. 1st Edition. UK: University of Strathclyde, 2015.

Іхсанов Ш.М., Дьяконов О.С. РЕАЛІЗАЦІЯ РЕАЛЬНОГО КАНАЛУ ПЕРЕДАЧІ ДАНИХ НА БАЗІ ОДНІЄЇ ПЛАТФОРМИ ADALM-PLUTO SDR

У роботі аналізуються питання впровадження в навчальний процес апаратних платформ програмно-визначених радіосистем при вивченні циклу дисциплін, пов'язаних з теорією електричного зв'язку і побудовою цифрових бездротових систем.

Очевидно, що використання в навчальних цілях апаратних платформ при вивченні систем бездротового зв'язку має ряд переваг, серед яких – наочна демонстрація роботи радіосистеми, можливість проводити додаткові експерименти, пов'язані з дослідженням поширення радіохвиль, врахування дрейфу характеристик апаратної частини і т. п.

У якості апаратної платформи для лабораторного стенду обраний модуль активного навчання від компанії Analog Devices – ADALM PLUTO SDR.

Зазначено, що для ефективного вивчення принципів цифрової обробки сигналів повинен бути створений скелет програми, що включає в себе частини коду з ініціалізації апаратної платформи, а також мінімальний функціонал блоків цифрової обробки, які використовуються в реальних системах.

На прикладі демо-програм, які поставляються з пакетом MATLAB, показано, що вони потребують суттєвого доопрацювання для реалізації працюючої радіодистеми.

В ході реалізації реальної системи передачі даних з модуляцією QPSK розроблені два варіанти обробки сигналу, спрямовані на усунення порушення часової синхронізації.

Наведено приклад програмної реалізації щодо прийняття рішення в частині визначення додаткового зсуву сигнального сузір'я на кут кратний 90 градусів з використанням кодів Баркера. Даний функціонал використовується після первинної синхронізації і декодування.

Представлені результати роботи розроблених програм, які демонструють прийнятне зведення частот при синхронізації приймача і передавача в рамках однієї апаратної платформи ADALM PLUTO SDR. Помилка оцінки даного зміщення становить 0,01–0,03 Гц. Наведено аналіз роботи алгоритму оцінки зміщення частота по фазі сигналу при різних співвідношеннях сигнал/шум. Також проаналізовано роботу двох алгоритмів синхронізації (алгоритм вибору між парними і непарними відліками і алгоритм символьної синхронізації) для декількох рівнів потужності передавача. Отримані дані також дозволяють зробити висновок про можливість безпомилкової передачі інформації в розглянутому каналі передачі даних при відношенні сигнал/перешкода близько 10 дБ.

Ключові слова: *ADALM PLUTO SDR, програмно-визначене радіо, QPSK, символьна синхронізація, фазова синхронізація, освіта, комунікаційні системи, модуляція.*

Дата надходження статті: 13.07.2025
Дата прийняття статті: 20.07.2025
Опубліковано: 27.10.2025