Laboratory 5: Spread Spectrum Communications

Cory J. Prust, Ph.D.
Electrical Engineering and Computer Science Department
Milwaukee School of Engineering
Last Update: 19 September 2018

Contents

0 Laboratory Objectives and Student Outcomes

0.1 Laboratory Objectives

This laboratory will assist students in understanding fundamental concepts of spread spectrum communication systems. The activities focus on direct sequence spread spectrum (DSSS) techniques such as those used in the Global Positioning System (GPS) and 3G Code Division Multiple Access (CDMA) cellular phone systems. Students will use simulation to experiment with and verify the operation of DSSS systems. Students will then utilize SDR hardware to implement a spread-spectrum receiver capable of decoding a DSSS signal sent over a wireless channel.

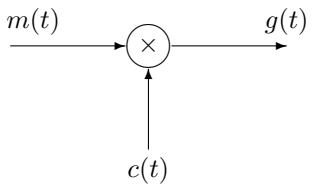
0.2 Student Outcomes

Upon successful completion of this laboratory, the student will:

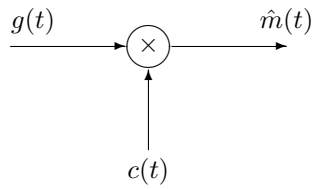
- Explain how spreading codes are used in direct sequence spread spectrum communication systems.
- Explain the importance of code synchronization in a spread spectrum communication system.
- Simulate a direct sequence spread spectrum communication system.
- Simulate a multi-user communication system that uses code-division multiplexing.
- Implement a spread-spectrum receiver using SDR hardware and verify its operation over a wireless channel.

1 Background

This laboratory concerns direct sequence spread spectrum (DSSS) communication systems. We will consider the baseband modulator (transmitter) given by the following block diagram



where $m(t)$ is the message signal and $c(t)$ is the spreading signal. The baseband demodulator (receiver) is given by the following block diagram



We will consider DSSS systems described as follows:

- The message signal $m(t)$ represents binary digital information and encodes the message bits as having values ± 1 over the bit interval, T_b. Therefore, $m(t)$ is the complex envelope of a BPSK communication signal.
- The spreading waveform $c(t)$ has values ± 1 over the chip interval, T_c. The sequence of values in $c(t)$ is given by a pseudo-noise (PN) code of length M. We define $c(t)$ to be a periodic repetition of the PN sequence, and therefore $c(t)$ has period MT_c.
- We will assume $T_c << T_b$. That is, the chip rate $R_c = 1/T_c$ is much higher than the bit rate $R_b = 1/T_b$. The multiplication by $c(t)$ in the modulator causes a *spreading* of the message signal spectrum. We define the spreading factor, L, to be

$$L = \frac{R_c}{R_b} \tag{1}$$

  The bandwidth of the spread spectrum signal $g(t)$ is approximately L times larger than the bandwidth of the message signal $m(t)$.
- Because both $m(t)$ and $c(t)$ have values ± 1, $g(t)$ also has values ± 1. Therefore, we can consider $g(t)$ to be the complex envelope of a BSPK communication signal. The passband communication signal can then be expressed as

$$\begin{aligned} s(t) &= \Re\{g(t)e^{j2\pi f_c t}\} \\ &= m(t)c(t)\cos(2\pi f_c t) \end{aligned}$$

- Note that the output of the multiplier in the receiver is

$$\begin{aligned} \hat{m}(t) &= g(t)c(t) \\ &= m(t)c^2(t) \\ &= m(t) \end{aligned}$$

  since the spreading waveform $c(t)$ takes on values ± 1. The multiplication in the receiver is often referred to as a *despreading* operation.

1.1 Synchronization

The DSSS receiver must have knowledge of the PN code in order to generate $c(t)$ for the despreading operation. Furthermore, the locally generated PN code in the receiver must be synchronized to the PN code used in the transmitter. Details of the synchronization processes used in real-world DSSS systems are beyond the scope of this laboratory. However, many of the synchronization processes involve *correlating* the received data with the known PN code. These techniques leverage the fact that certain PN codes exhibit special correlation properties making them useful for synchronization[1]. These properties are revealed by computing the *autocorrelation* function of the PN code.

Let $c[n]$ denote the periodic spreading sequence having period M. Define the autocorrelation function of $c[n]$ as

$$R_c[k] = \frac{1}{M}\sum_{n=0}^{M-1} c[n]c[n-k]$$

The autocorrelation function is a measure of similarity between the code and time-shifted versions of the code. Note that for $c[n]$ taking on values ± 1, $R_c[0] = 1$ is the maximum value of the autocorrelation function. For purposes of synchronization, it then follows that a code having small autocorrelation values for $k \neq 0$ is desirable.

Figure 1 shows an example MATLAB script for computing the autocorrelation function of a given PN code. The script also demonstrates using the MATLAB function `xcorr` for correlating a received signal with its PN code. The issue of synchronization will be further explored in Sections 2 and 4.

1.2 Code-Division Multiplexing

Spread spectrum techniques can be used to transmit multiple message signals simultaneously in the same frequency band. Such techniques, referred to as *code-division multiplexing* (CDM), are often used in mobile phone communications. This form of multiplexing is made possible by assigning each user a unique spreading code from a set of codes. Then, multiple spread-spectrum signals can be transmitted in the same frequency band, and the messages can be decoded at each receiver using the unique (and properly synchronized) spreading code. In this way, the larger bandwidth used by spread-spectrum systems is offset by allowing multiple users to simultaneously access the frequency band.

One of the best-known sets of spreading codes used in CDM systems are the Walsh-Hadamard codes. These particular codes have the desirable property that they are mutually orthogonal. That is, when properly synchronized, the correlation between any one code and any other code is exactly zero. The matrix below shows the set of length 8 orthogonal Walsh-Hadamard codes, where each row of the matrix is a different spreading code.

$$H_8 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix}$$

Note that each code is orthogonal to every other code (i.e., the inner product of any one row with any other row is exactly zero). The concept of CDM using DSSS will be further explored in Section 3.

[1] The subject of designing and generating PN sequences is beyond the scope of this laboratory. Well-known codes include m-sequences, Gold codes, and Walsh-Hadamard codes. Students interested in learning more are encouraged to consult the course textbook or take a follow-on course in communication systems and/or coding theory.

```
%% example_correlation.m
%   Example script that computes the autocorrelation function of a
%   PN sequence.  The script also demonstrates using the MATLAB "xcorr"
%   function to correlate a received signal with a PN sequence.
%
%   Cory J. Prust, Ph.D.
%   Last Modified:  8/3/2018

close all
clear all

%% Specify PN code
code = [-1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1]';
M = length(code);

%% Compute Autocorrelation Function
% R[k] = 1/M sum_{n=1}^{M} c[n] c[n-k]
% Note: "circshift" is used because the sequence is assumed periodic
R = zeros(M,1)
k = 0:1:(M-1);
for ii=1:length(k)
    R(ii) = 1/M * (code' * circshift(code,k(ii))); % circshift
end
stem(k,R)

%% Simulate received DSSS data and use "xcorr" to correlate it with PN code
% Assume spreading factor of M
m = [1 1 -1 1]';
g = [m(1)*code; m(2)*code; m(3)*code; m(4)*code]; % or, use "g = kron(m,code);"

[r,lag] = xcorr(g,code);
figure
stem(lag,r)
```

Figure 1: MATLAB example showing calculation of the autocorrelation function of a PN sequence and use of MATLAB's `xcorr` function.

2 Direct Sequence Spread Spectrum Simulation

In this portion of the laboratory you will construct a Simulink model for simulating a direct sequence spread spectrum (DSSS) communication system. The complete model is shown in Figure 2.

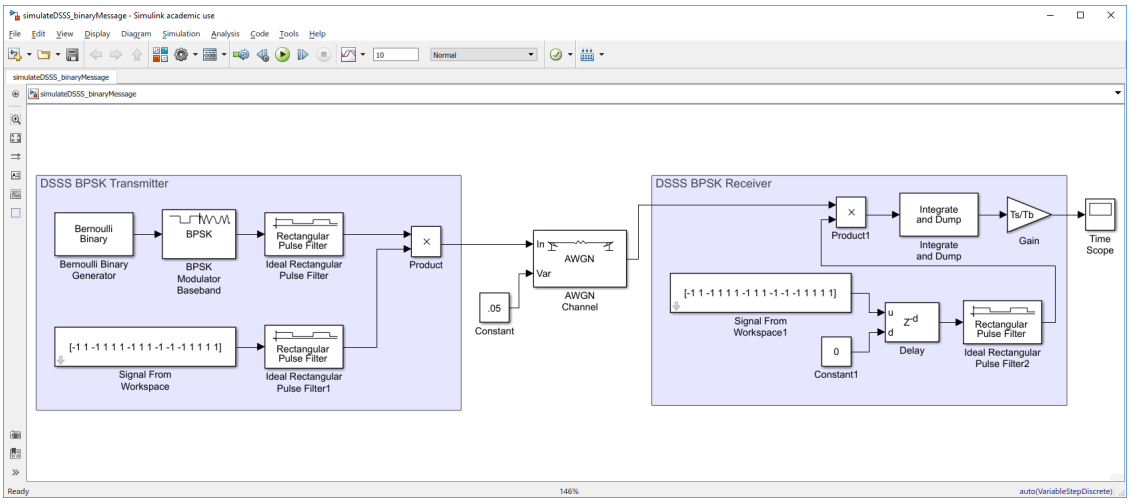


Figure 2: Simulink model for simulating a DSSS transceiver.

The following sequence of steps is suggested:

1. Open a new Simulink model and define the following in `Model Properties -> Callbacks -> InitFcn`.

   ```
   fs = 1e3; % sampling frequency
   Ts = 1/fs; % sample time
   Tb = 0.1; % bit time
   Tc = 0.01; % chip time
   frameSize = 1000; % size of each frame in source blocks
   ```

   The model will be specified in terms of these parameters.

2. Construct the *DSSS BPSK Transmitter* portion of the model.

   - The **Bernoulli Binary Generator** block generates random binary numbers using a Bernoulli distribution. This block acts as our message source in the simulation. Set `Sample time` to `Tb`. Set the `Samples per frame` to `frameSize`.
   - The **BPSK Baseband Modulation** block applies BSPK modulation to the message bits. The output of this block connects to a **Ideal Rectangular Filter** block, which translates the BPSK symbols into rectangular pulses. Set `Pulse length (number of samples)` to `Tb/Ts`. Set the `Input processing` for `Columns as frames` and set the `Rate options` for `Allow multirate processing`. The output of this block is the complex envelope of our BPSK waveform, $m(t)$.
   - The **Signal from Workspace** block specifies the spreading sequence. Set `Signal` to

     `[-1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1]`

     set `Sample time` to `Tc`, and set `Samples per frame` to `frameSize`. The output of this block connects to a **Ideal Rectangular Filter** block, which translates the sequence into rectangular pulses. Set `Pulse length (number of samples)` to `Tc/Ts`, set `Form output after final data value by` to `Cyclic repetition`. Set the `Input processing` for `Columns as frames` and

set the `Rate options` for `Allow multirate processing`. The output of this block is the spreading waveform, $c(t)$.

- The **Product** block creates the complex envelope of the DSSS BPSK signal, $m(t)c(t)$.

Experiment with this portion of the model by adding **Time Scope** and **Spectrum Analyzer** blocks.

Hint: You may find it helpful to change the **Time Scope** to a "stem" plot by selecting `View -> Style` and changing the `Plot type` setting.

3. Add the **AWGN Channel** block, specifying the `Mode` as `Variance from port` and using a **Constant** block to control the noise variance. As you experiment with the model, adjust the noise variance (i.e. the noise power) to see the effects of noise on the system.

4. Construct the *DSSS BPSK Receiver* portion of the model.

   - The **Signal from Workspace** block and **Ideal Rectangular Pulse** block create the spreading sequence (for purposes of despreading), and are identical to those used in the transmitter portion of the model.
   - The **Delay** block allows you to delay the receiver spreading sequence used relative to the one used in the transmitter. Set `Delay length` to `Input port` and use a **Constant** block to specify the delay. Set the `Input Processing` to `Columns as channels (frame based)`. The delay can be modified while the model is running. Use only integer values.
   - The **Product** block performs the despreading operation in the receiver.
   - The **Integrate and Dump** block integrates the received pulses, which can be thought of as part of the ideal correlation receiver (or the matched filtering process). Set the `Integration period (number of samples)` parameter to `Tb/Ts`. The **Gain** block, set to `Ts/Tb`, adjusts the amplitude of the resulting signal to the nominal range of ± 1.

   For simplicity, the additional blocks required to recover the binary message have been omitted from the model.

5. Experiment with the model by running it. Add **Time Scope** and **Spectrum Analyzer** blocks as needed. Be sure you understand how it operates before proceeding.

Question 2.1: Based on the parameters specified in Step 1 above, what is the spreading factor of the DSSS system?

Question 2.2: Use MATLAB to compute the autocorrelation function for the sequence given in Step 2 above. Comment on the properties of the autocorrelation function. Does the sequence exhibit desirable properties for use in a DSSS communication system? Submit your MATLAB code and plot of the autocorrelation function.

Question 2.3: In the `DSSS BPSK Transmitter`, connect a **Spectrum Analyzer** block to the BPSK message waveform $m(t)$ and the spread waveform $m(t)c(t)$. Submit a screen capture. Measure the bandwidth of each signal. Do these bandwidth measurements agree with the spreading factor of the DSSS system? *Hint:* Measure the bandwidth as the frequency of the first spectral null.

Question 2.4: In the `DSSS BPSK Receiver`, connect a **Time Scope** and **Spectrum Analyzer** block to the recovered message waveform (output of the **Product** block). Carefully examine these plots as you adjust the `Delay length` parameter of the **Delay** block. You should observe that the receiver properly recovers the message signal when the `Delay length` parameter is 0. Describe what happens to the recovered message when the `Delay length` parameter is 1. Describe what happens when the `Delay length` parameter is 16. Explain the results.

3 Code Division Multiplexing Simulation

In this portion of the laboratory you will construct a Simulink model for simulating a communication system that uses code division multiplexing to send multiple messages simultaneously in the same frequency channel. The complete model is shown in Figure 3.

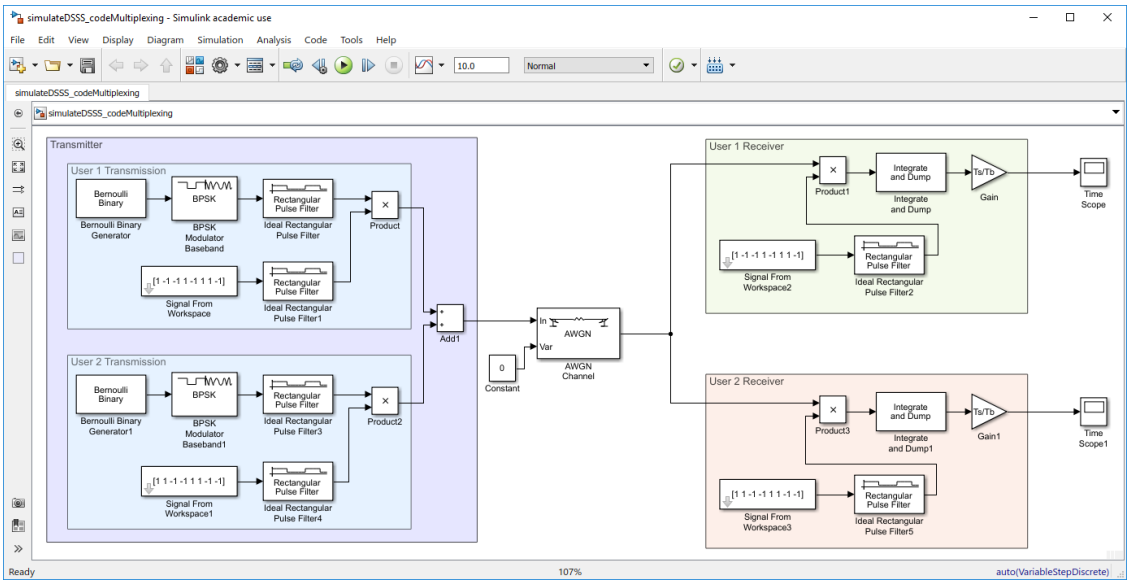


Figure 3: Simulink model for simulating code division multiplexing.

The following sequence of steps is suggested:

1. Open a new Simulink model and define the following in `Model Properties -> Callbacks -> InitFcn`.

   ```
   fs = 1e4; % sampling frequency
   Ts = 1/fs; % sample time
   Tb = 0.1; % bit time
   Tc = 0.0125; % chip time frameSize = 1000; % size of each frame in source blocks
   ```

   The model will be specified in terms of these parameters.

2. Construct the *Transmitter* portion of the model.

   - The **Bernoulli Binary Generator**, **BPSK Modulator Baseband**, and **Ideal Rectangular Filter** blocks are used in the same manner they were in Section 2.
   - The sequences

     `[1 -1 -1 1 -1 1 1 -1]`

     and

     `[1 1 -1 -1 1 1 -1 -1]`

     are the spreading codes for User 1 and User 2, respectively. Note that these codes are members of the length 8 Walsh-Hadamard codes.
   - The **Add** block is used to sum the User 1 and User 2 coded messages into a single transmission.

3. Construct the *User 1 Receiver* and *User 2 Receiver* portions of the model.

- The **Ideal Rectangular Filter**, **Product**, **Integrate and Dump**, and **Gain** blocks are used in the same manner they were in Section 2.
- Be sure to use the proper sequences for each receiver.

4. Experiment with the model by running it. Add **Time Scope** and **Spectrum Analyzer** blocks as needed. Be sure you understand how it operates before proceeding.

Question 3.1: In the *Transmitter*, use **Time Scope** and **Spectrum Analyzer** blocks to observe the inputs to and the output of the **Add** block. That is, observe the User 1 and User 2 DSSS signals, and their sum. Include screen captures and comment on what you observe. In particular, comment on the signal bandwidths.

Question 3.2: Use a **Time Scope** block to display the User 1 and User 2 messages in the transmitter, and the User 1 and User 2 messages recovered in the receiver. Submit screen captures for simulations where the AWGN noise variance is 0, 1, and 10. Comment on the results.

Question 3.3: Repeat the previous problem, however now change the *User 2 Receiver* sequence to

```
[1 -1 1 -1 -1 1 -1 1]
```

4 DSSS BSPK - Text Message Communication

In this portion of the laboratory you will receive a DSSS BPSK waveform using your personal SDR device. You will then despread the signal and recover an ASCII text message.

Construct the Simulink receiver model shown in Figure 4.

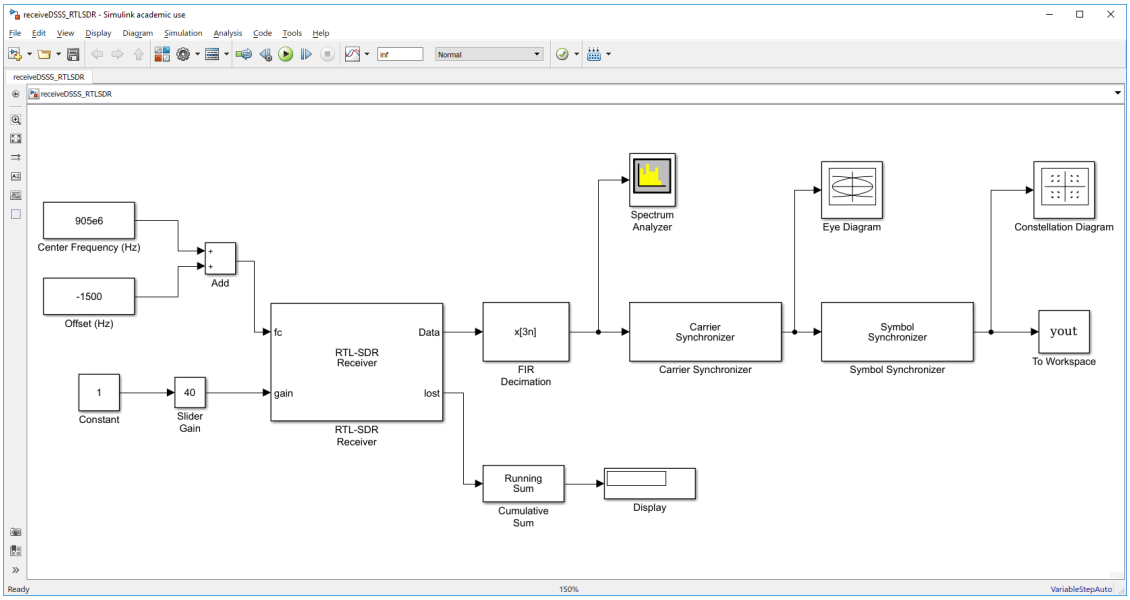


Figure 4: Simulink model for receiving a DSSS BPSK message.

Some salient points regarding the DSSS broadcast and the receiver model:

- The DSSS broadcast consists of BPSK symbols. The message information is encoded using BSPK modulation, giving symbols having values ± 1. The chip sequence also contains values of ± 1. Therefore, the complex envelope of the broadcast signal has values of ± 1, and can therefore be considered to be a BSPK waveform.
- The message bits are sent at 1000 bits per second, and the chip rate is 8000 chips per second.
- The chip sequence used in the transmitter is

```
[-1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1]
```

- The transmitter uses rectangular pulse shapes.
- **RTL-SDR Receiver** configuration:
  - Sampling rate: 240e3
  - Output data type: `single`
  - Samples per frame: 120000
  - "Lost samples output port" enabled. This signal will be non-zero when samples are lost. Connect this output as shown to **Cumulative Sum** and **Display** blocks, which will count any samples that are lost during the recording. It is important that no samples are lost. Consult your instructor if samples are being lost.

  - The center frequency must be corrected according to the calibration performed in our previous laboratory.

- The **FIR Decimation** block reduces the sample rate of the receive data by a factor of 3. The output of the **RTL-SDR Receiver** block is configured for $240e3$ samples per second, therefore the decimation reduces the rate to $240e3/3 = 80e3$ samples per second. This gives 10 samples per chip, or 80 samples per bit, in the receive data. Confirm the sample rate, and the frequency offset correction, by carefully viewing the **Spectrum Analyzer** plot.

- The **Carrier Synchronizer** block adjusts for carrier frequency and phase offsets. Configure the `Modulation` for `BPSK` and `Samples per symbol` to 10. This block will synchronize to the chip sequence.

  IMPORTANT: This block may synchronize to a constellation that is a 180 degree rotation of the transmitted signal. We will detect the presence of and correct for any such rotation in a later step.

- The **Symbol Synchronizer** adjusts for clock drift in the incoming signal, ensuring that the incoming pulses are sampled at the appropriate time. Configure the `Timing error detector` for `Zero-Crossing (decision directed)` and `Samples per symbol` to 10. The block will output exactly one sample per chip.

- The **Eye Diagram** can be used to confirm that carrier synchronization has occurred. Set `Samples per symbol` to 10.

- The **Constellation Diagram** can be used to confirm the received constellation. Set `Samples per symbol` to 1.

- The **To Workspace** block writes data to the MATLAB workspace. Configure `Limit data points to last` to 4000, and `Save format` to `Structure`.

  - After running the model, you will see a structure named `yout` in the MATLAB workspace. You can save the MATLAB workspace, and therefore your recorded data, using the `save` function in MATLAB. You can then restore the workspace at a later time using the `load` function.
  - The actual symbols are located within the structure at `yout.signals.values`

At this point, you should have a vector in MATLAB containing 4000 values, each of which corresponds to a chip in the DSSS sequence. Your task is to recover the ASCII text message. The following sequence of steps is suggested:

1. Confirm that the vector of data represents a baseband BSPK signal. You can do so by plotting the data (e.g., using `stem` or `scatterplot`).

2. You will notice that the vector of data is complex-valued. Because the data is BPSK modulated, we only need its real part. The command

   ```
   data = real(yout.signals.values);
   ```

   will extract just the real part of the vector.

3. You must implement the despreading operation. Doing so will require several steps:

   - Use the MATLAB `xcorr` function to correlate the data vector with the spreading sequence. For example

     ```
     code = [-1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1];
     [r,lag] = xcorr(code,data);
     stem(lag,r)
     ```

     will compute and plot the cross-correlation between the data vector and the spreading sequence.

- Examine the cross-correlation plot carefully. It will tell you how to shift the spreading sequence so as to properly despread the received data.
- You must create the despreading signal. This can be thought of as a vector where each value corresponds to a chip. You may find the MATLAB function `circshift` helpful for shifting the code sequence. See Figure 1 for an example of using this function. You may find the MATLAB function `repmat` helpful for creating the despreading signal. For example

  ```
  c = repmat(code',250,1);
  ```

  will create a column vector containing 250 consecutive copies of `code` representing the despreading signal. Note that the dimensions of `c` should exactly match your received data vector.
- The despreading operation is simply the element-by-element product of the received data and the despreading signal. Plot the resulting waveform. You should be able to clearly see the BSPK symbols, with 8 consecutive samples of nearly identical amplitude representing each bit.

4. Following proper despreading, it will be convenient to reduce the data vector to one sample per bit. It is up to you to decide how to achieve this.

   Hint: Because the transmitter used rectangular pulses, this operation could be as simple as retaining every 8th sample. A more robust approach would be to compute the mean of every 8 samples, which is effectively a matched filter operation for rectangular pulses.

5. You can now perform the BSPK demodulation to recover the binary data sequence.

6. Plot and carefully inspect the binary data sequence. In particular, locate a complete data frame by identifying the message *preamble* (described by your instructor). If you see an inverted version of the `preamble`, this means that the **Carrier Synchronizer** locked 180 degrees out of phase relative to the transmit carrier. You can correct for this by inverting your binary data sequence.

7. After identifying a complete data frame, convert the binary sequence to ASCII characters. This process is identical to the one used in a previous laboratory.

Question 4.1: Provide a plot of the cross-correlation between your received data and the spreading sequence. Explain what is seen in the plot. Explain how this information was used in the despreading operation.

Question 4.2: Provide time-domain `stem` plots showing the received data before and after the despreading operation. Explain what is seen in the plot.

Question 4.3: Provide frequency-domain plots showing the signal before and after the despreading operation. Explain what is seen in the plots. Comment on the relative bandwidth of the signals.

Question 4.4: Explain how you processed the data vector, after despreading, to retain one sample per bit (as described in Item 4 above). Include the MATLAB code you used to achieve this task.

Question 4.5: Provide a time-domain `stem` plot of the recovered message bit sequence. This plot must show the message preamble and a complete data frame. Annotate the plot, identifying the preamble.

Question 4.6: What is the text message? Provide the complete MATLAB code listing you used in this portion of the lab.

References

[1] Mathworks. Communications Toolbox. Online Resource. https://www.mathworks.com/products/communications.html.

[2] Mathworks. USRP Support Package from Communications Toolbox. Online Resource. http://www.mathworks.com/discovery/sdr/usrp.html.

[3] Mathworks. RTL-SDR Support Package from Communications Toolbox. Online Resource. https://www.mathworks.com/hardware-support/rtl-sdr.html.

[4] Leon W. Couch III. *Modern Communication Systems*. 1995.

[5] Simon Haykin. *Communication Systems*. 4th edition, 2001.

[6] B.P. Lathi and Zhi Ding. *Modern Digital and Analog Communication Systems*. 5th edition, 2019.