# Abstract

|  |  |
| --- | --- |
| ECE4305 Lab2 report  Receiver Structure and Waveform Synthesis of a Transmitter and a Receiver | Renato Iida  Le Wang  Rebecca Cooper |

In this lab we simulated the decoding of a signal generated from 4 symbols using several methods. Each signal contained such a number of samples that even a high degree of variance in the Gaussian white noise did not significantly change the error rate. Over the air we transmitted simple 1s and 0s. we used both DBPSK and DQPSK modulation in his part of the project but because of the application there was little difference. We used correlation and the Baker code to find the start of the frames in the next part of the lab. The way that Simulink operates prevented us from achieving good outputs for long.

# Table of Contents

Contents

[Abstract 0](#_Toc412537872)

[Table of Contents 2](#_Toc412537873)

[Introduction Chapter 4](#_Toc412537874)

[Description of all steps taken throughout the experiment 5](#_Toc412537875)

[Software Implementation 5](#_Toc412537876)

[Observation Vector Construction 5](#_Toc412537877)

[Maximum-Likelihood Decoder Implementation 5](#_Toc412537878)

[Correlator Realization of a Receiver in Simulink 5](#_Toc412537879)

[USRP Hardware Implementation 5](#_Toc412537880)

[DBPSK Transmitter 5](#_Toc412537881)

[DBPSK Receiver 5](#_Toc412537882)

[DQPSK Transmitter and Receiver 5](#_Toc412537883)

[Frame Synchronization 5](#_Toc412537884)

[Observations of the performance and behavior of the communication system 7](#_Toc412537885)

[Software Implementation 7](#_Toc412537886)

[Maximum-Likelihood Decoder Implementation 7](#_Toc412537887)

[Correlator Realization of a Receiver in Simulink 7](#_Toc412537888)

[USRP Hardware Implementation 7](#_Toc412537889)

[DBPSK Transmitter versus Receiver 7](#_Toc412537890)

[Frame Synchronization 7](#_Toc412537891)

[Questions 8](#_Toc412537892)

[Software Implementation 8](#_Toc412537893)

[Observation Vector Construction 8](#_Toc412537894)

[Maximum-Likelihood Decoder Implementation 11](#_Toc412537895)

[Correlator Realization of a Receiver in Simulink 12](#_Toc412537896)

[USRP Hardware Implementation 18](#_Toc412537897)

[Experimental results demonstrating the operation of the system 19](#_Toc412537898)

[Software Implementation 19](#_Toc412537899)

[Observation Vector Construction 19](#_Toc412537900)

[Maximum-Likelihood Decoder Implementation 19](#_Toc412537901)

[Correlator Realization of a Receiver in Simulink 20](#_Toc412537902)

[USRP Hardware Implementation 20](#_Toc412537903)

[Frame Synchronization 22](#_Toc412537904)

[Conclusion 23](#_Toc412537905)

[Appendix 1: Matlab code for the Software Implementation 24](#_Toc412537906)

[correlator.m 24](#_Toc412537907)

[decoder.m 27](#_Toc412537908)

[Correlator Realization of a Receiver in Simulink 30](#_Toc412537909)

[DBPSK/DQPSK transmitter 31](#_Toc412537910)

[DBPSK Receiver 31](#_Toc412537911)

[Frame Synchronization Receiver 32](#_Toc412537912)

# Introduction Chapter

This lab includes two sections: the software implementation and hardware implementation.

In the software implementation of the lab, we will design and implement two different receivers, the matched filter the correlator receivers. We start by creating the observation vector X, which is the transformation of the received signal waveforms s(t) into vector space. To find the best match we then proceed by comparing the observation vector to the signal space representation of the transmitted waveforms, which is known as the maximum-likelihood (ML) decoder. In addition to the matched filter receiver, we also implement another type of the receiver structure, i.e., the correlator-based receiver.

In the hardware implementation section, we will design two modulation schemes including a basic transmitter and receiver. We first use DBPSK modulation in the system, and we change to DQPSK. Then, we compare the error rate to evaluate the performance between DBPSK and DQPSK.

The last part of the lab is focusing on the frame synchronization. In real-world situation, it is crucial to know where each frame starts in order to perform any format of data reception. So the receiver needs to frame implementation to know the start point of a specific frame. In the lab, the Barker code is attached to the frame to realize the purpose. We first test the frame synchronization function in Simulink and then transmit the signal through USRP N210. Finally, we successfully transmit messages, e.g., ‘hello world’, to another station.

# 

# Description of all steps taken throughout the experiment

## Software Implementation

### Observation Vector Construction

The correlator.m creates the reference of 4 symbols. Each symbol is selected randomly. This symbol is added to a random noise with 0.5 variance. It runs for 100 times. This is the received signal of a AWGN signal. In the receiving process, this signal is multiplied separately for each reference symbol. It in summed over the symbol period that is 3 seconds in this case.

The result of this sum is plotted against time to see the results.

### Maximum-Likelihood Decoder Implementation

The Maximum-Likelihood Decoder (MLD) is based on the probabilities that the noisy unknown symbol is equivalent to the each of the known symbols that makes up the original signal. Simplifying the probability equations results in a simple algorithm where we take the dot product of the noisy unknown symbol and each of the known symbols. Half the energy of the corresponding symbol is removed. For each unknown symbol the maximum value obtained after the energy normalization corresponds to the symbol with the highest probability of being the unknown symbol.

The energy of each symbol is found using the energy of the signal vectors found in problem 1 of 6.5.

### Correlator Realization of a Receiver in Simulink

This correlator reuses the basic blocks of correlator,m to create the reference signal.  After, the Gaussian Noise Generator block create the behavior of the AWGN imperfection to this signal. The receiver part split the signal in for path and each path is multiplied by one of the reference symbols s1,s2,s3 and s4. Each path have an integrated and dump block with 3000 samples that it is the amount of samples per symbol. Each path is a input for a function to decide which is more probably transmitted symbol. Each path is divide by 1000 that it the sample rate and after that is subtracted by the half of the energy of the correspondent symbol. The highest result is found and it corresponded path is the symbol transmitted.

## USRP Hardware Implementation

### DBPSK Transmitter

We start the design of the basic transmitter. The signal to be transmitted was from a vector [1 0]’ that was set to the output type “Cyclic Repetition. This will generate a constant stream of ‘1 0 1 0’. This pattern is immediately obvious at the receiver, and any errors will be easy to spot. We set the Sample time to 179 via **callback functions** in ‘*InitFcn’*. Using DBPSK avoided the need for a coherent reference signal at the receiver. We transmitted this signal at 0.902e9 Hz continuously.

### DBPSK Receiver

At the receiver, the frequency of the signal was so close to 0.902e9 Hz that it was unnecessary to input the compensation of the frequency offset manually.

### DQPSK Transmitter and Receiver

We constructed the DQPSK system with DBPSK Transmitter and Receiver system. Instead of the DBPSK Modulator and Demodulator blocks we used the DQPSK versions of those blocks.

## Frame Synchronization

We input ‘hello world’ into *charToBitsAndBack.m*, and run the section that initializes our system and converts our message into bits. Those bits were stored in *sBitTx* and the output type was modified to “Cyclic Repetition”. Also, pad of 10 zeros was added to the cyclic repetition to keep the message aligned. The message was repeated 100 times. The received variable is changed to a new Matlab block to verify the message received is ‘hello world’. This block used the received part of original *charToBitsAndBack.m*. To verify the delay calculation was correct, we created a new variable to show this value in workspace. The delay was changed to 10 and the delay was calculated correctly with 5 and 10. Create a new transmitter based on this blocks and change the BPSK to DPSK and add the raised cosine transmitter filter and the USRP transmitter

1. The parameters of filter in the transmitter:
   1. Group delay :10
   2. Rolloff factor: 0.2
   3. Upsampling Factor: 4
2. The receiver used the same filter after the BPSK demodulator and the parameters are:
   1. Input samples:4
   2. Group Delay: 10
   3. Rolloff factor: 0.2
   4. Downsampling: 4
   5. Sample offset: 0

# Observations of the performance and behavior of the communication system

## Software Implementation

### Maximum-Likelihood Decoder Implementation

The output of the accumulators of symbols 1 and 3 are nearly identical when the symbol 1 was transmitted. The difference between the two symbols exists only when symbol 1 is 0. In element by element multiplication there will be little difference between 0\*0 and -1\*0 even with noise. The difference in energy levels between symbol 1 and symbol 3 is only 1. As long as the noise on the transmitted 0 is low, the sum of the multiplication over this time period will most likely be less than 1. When the accumulator is used over thousands of samples this difference is not very significant. Normalizing the accumulators for the number of samples per second before subtracting the energy results in 0% error over 10,000 symbols.

### Correlator Realization of a Receiver in Simulink

Simulating a receiver using 3000 samples per symbol made the symbol detection very robust to the AWGN channel. The error rate increased with very low number of SNR. The initial model didn’t correctly remove the energy of the result, it added errors with high SNR and the error was not change with SNR until -20 dB. This strange behavior made a whole verification of all the blocks and the correction of the normalization made the BER goes to zero with SNR around zero.

## USRP Hardware Implementation

The sample time and the frame length must have an integer proportionality to avoid problem in the delay line block.

### DBPSK Transmitter versus Receiver

During the experiments, the problem we met is we kept receiving ‘111111’ or ‘00000’. The reason is in the Transmitter, we set the upsampling factor of the Raised Cosine Transmit Filter to 2. In the Receiver, we also set the downsampling factor to 2. However, due to the existence of the Mueller-Muller Timing Recovery block, which already take downsampling one time, so if the downsampling factor of the Raised Cosine Receive Filter is still 2, we will lost half of the frames, which is why we kept receiving only ‘1’s or ‘0’s. We solve this problem by changing the downsampling rate into 1 within the raised cosine block and we get the correct signals as ‘10101010101010’.

## Frame Synchronization

This module took some time to initialize. It would produce no messages for many repetitions of the program. After some time the buffer would fill up and we would see ‘udh:0‘ output in Matlab. After this point there would be periods of properly decoded and synchronized ‘Hello World’ among periods of garbage.

# Questions

## Software Implementation

### Observation Vector Construction

1. *Plot a randomly generated stream of these waveforms s(t) in MATLAB. What do you observe? For example, a sample plot of s(t) generated by 10 transmitted symbols is shown in Figure 6.4.*

We observed a sequence of random symbols which could visually be decoded into the 4 symbols s(t) as shown in Figure 1. The first 5 symbols are as follows, s3, s1, s3, s4, s4.

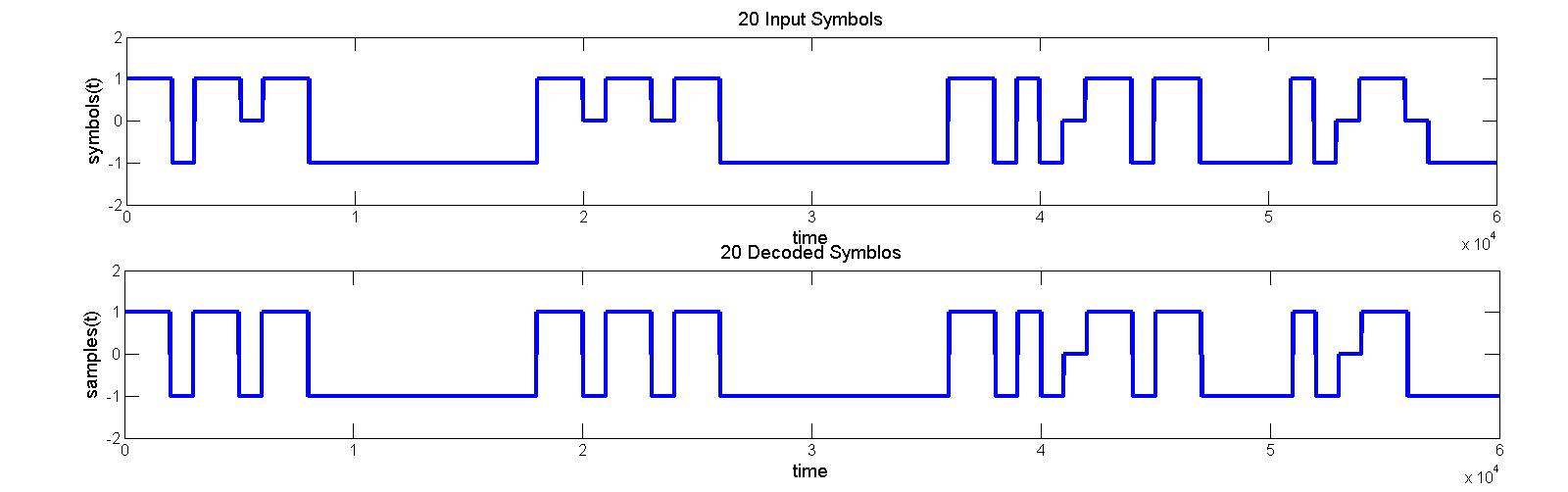


Figure : 20 input symbols over 60 seconds

1. *Plot the time domain representation of the input signals s(t) and output signals x(t) for the channel for several different noise variances. Explain how the noise could potentially impair the successful decoding of the intercepted signal at the receiver.*

The noise is mixed into the sending signal, if the noise is too heavy, the decoder would be unable to distinguish the original signal. The higher the variance, it is more difficult to distinguish the different symbols from each other. Figure 2 compares the signal with the signal that has an AWGN of variance 0.1. Figure 3 compares the signal and the same signal with an AWGN of variance 0.5. Figure 4 compares the signal and the same signal with an AWGN of variance 1.

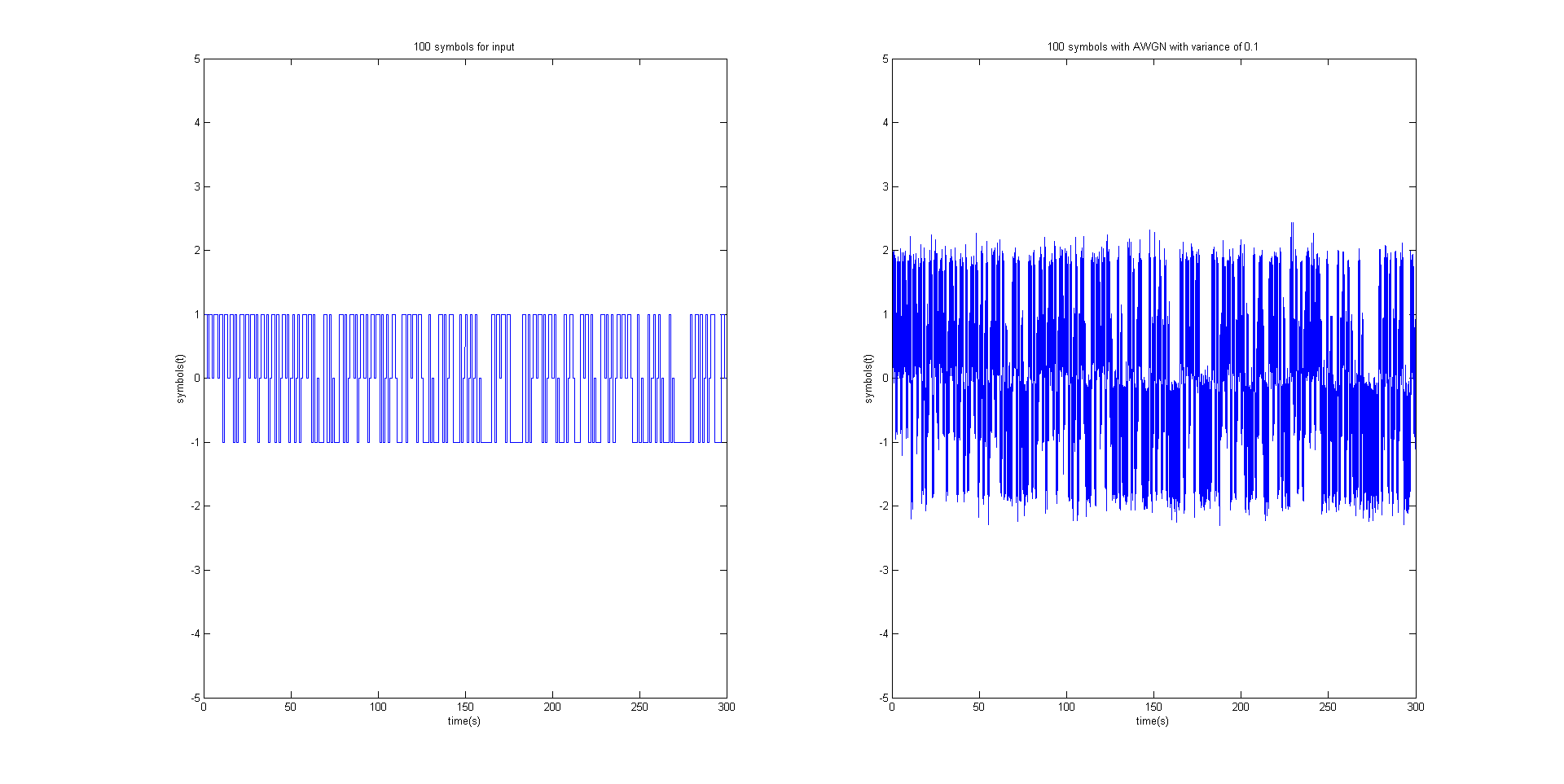


Figure : Random signal generated from s1, s2, s3, s4 and the same signal with AWGN variance of 0.1



Figure Random generated symbols from s1, s2, s3 and s4. Same symbols with AWGN with variance 0.5



Figure Random generated symbols from s1, s2, s3 and s4. Same symbols with AWGN with variance 1

*3. When you have done all these steps test your implementation with a transmission of duration 300s, randomly consisting of one of the 4 equiprobable signals of duration T = 3s each with zero-mean white Gaussian noise of variance 0.5 added. Assume perfect synchronization between the received signal and the communications system. Plot each of the elements of the observation vector X using the stem command.*

The results of the sum without normalizing by the energy and divide the number of samples is shown in Figure 5.



Figure Results of integration of the received signal with each of the original signal. This value isnot normalized by the number of samples or the energy per symbol

### Maximum-Likelihood Decoder Implementation

*1. Plot the output of the accumulator for each of the branches. Provide your observations and explain.*

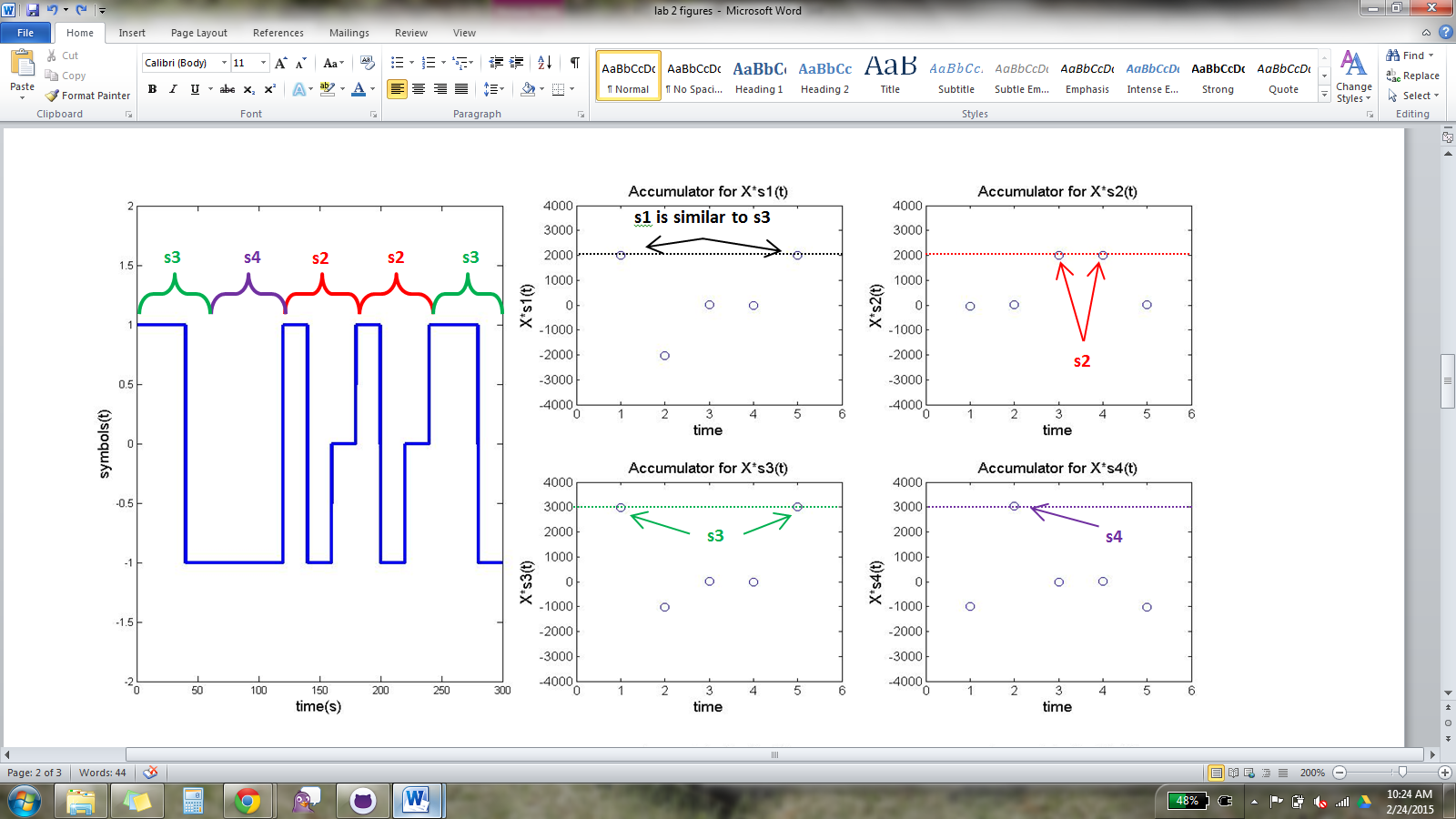


Figure .....

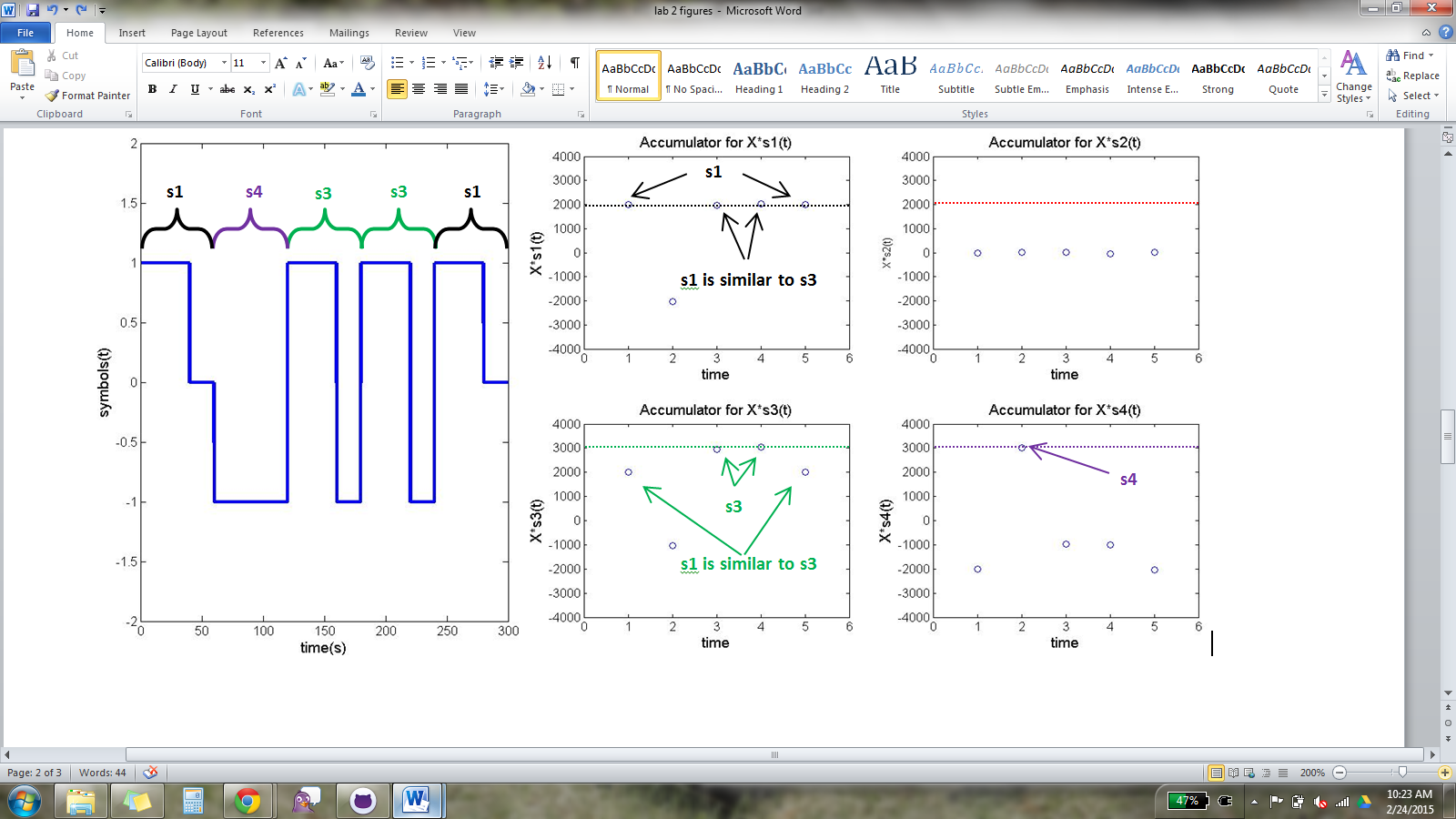


Figure .....

The accumulator combines every sample of into one point per symbol. When X(t) and are similar this number will be high, when it is low, the accumulator value will be low. The symbols s1 and s3 are very similar and their accumulator responses are nearly identical.

2. *Remove the energy subtraction from each of the branches and repeat your experiment.*

*Do you get the same results as before?*

No, our results improved by ~12.5% over 10,000 symbols by including the energy subtraction from each of the branches instead of only relying upon the accumulators to make the decision. The plots of the accumulator above show how close the accumulations of s1 and s3 are when the transmitted signal is s3. After normalizing for the number of samples the output of the s3 branch or the s1 branch will be ~2 if the noisy signal is s1. By subtracting half the energy the proper symbol is decoded.

*What would be required for the results with and without energy subtraction to be equivalent?*

The energy for each group of symbol coefficients would have to be the same for the energy subtraction to make no difference.

### Correlator Realization of a Receiver in Simulink

1. *In Software Implementation 1 do not integrate the entire period Integrate until 0.75T. Plot both the estimate m and the actual transmitted symbol mi and compare it with the actual plot.*
2. *In Software Implementation.2, do not subtract the energy of s(t) from each branch. Plot both the estimate m and the actual transmitted symbol mi and compare it with the original plot.*
   1. *I lost this plot, whoops.*
3. *Combine correlator.m and decode.m, SNR = 10-1 to get the BER*

The error rate over 10000 symbols was 0% when the variance was 0.5because of the great number of samples the error rate did not change significantly.

1. *Explain the end-to-end operation of the basic Simulink model, from the binary PAM transmitter all the way across to the output of the integrate-and-dump block. Use time domain plots to help illustrate your explanations.*
   1. The Bernoulli Binary Generator creates bits with values of 1 and 0
   2. The add and subtract blocks make the bit values 1 and -1 as shown in Figure 8.
   3. The Gaussian Noise Generator block has a variance of 0.5 in Figure 9 and was added to the generated signal to simulate the AWGN behavior, the result signal is shown in Figure 10.
   4. The ‘*integrate and dump’* block removes the imperfection of AWGN channel by calculating the integral of the received signal in each symbol time. The results in Figure 11 are similar to the original signal of Figure 8.

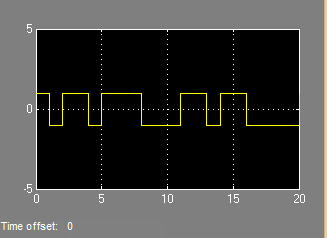


Figure Bits generated by the Bernoulli Binary Generator

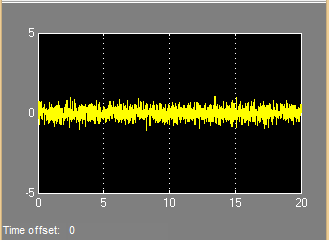


Figure Noise generated by the Gaussian Noise Generator with a variance of 0.5

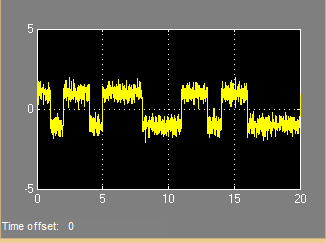


Figure The Generated Signal plus Noise

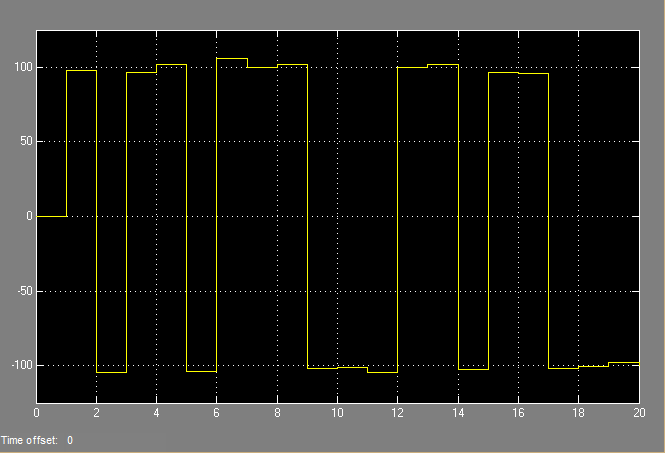


Figure Result after the integrate and dump descsion

5. Plot the output of the Transmitter to show the randomly generated sequence of waveforms.



Figure The Input Signal in Simulink and the signal with AWGN of variance 0.5 or SNR of 2.2627 dB

6. How much noise must be added to make the signal stream unrecognizable?

The higher the variance of the AWGN added more unrecognizable the symbols became. Because it has 3000 samples to recreate the information, need a very low SNR to have a more than 50% of error. An extreme case of SNR -38.5 dB was used to get a 50% of error is shown in Figure 13.



Figure Symbols transmitted and the symbols received with SNR of -38.5 dB and a CBER of 50% in the detection with the energy normalized

*7. What did you set as the number of samples to be integrated?*

The number of samples is 3000.

*8. What did you notice about the relative energy levels for each of the branches?*

The s1 and s2 have the same energy and lower than s3 and s4 which also have the same energy.

*9. Plot the output of the decision making process, indicating which of the four waveforms were selected. Why is there a delay in the decoded signal relative to the originally transmitted signal?*

The signal was processed offline after the simulation finished so the delay to input and output not happen.



Figure 14 Symbols transmitted and the symbols received and no errors in the detection with the energy normalized

*10. Suppose you removed the energy subtraction for all the branches. Does the BER performance of the system change?*

Yes. Using the same simulation of Figure 14 that have BER zero, it goes to BER around 10% because symbols 1and 3 because very similar as shown in Figure 1.

5

Figure Symbols transmitted and the symbols received and error of 10% in the detection without energy normalization

## USRP Hardware Implementation

1. Compare the performance between DBPSK and DQPSK. Which one is better? Justify your answer.

Both of DBPSK and DQPSK could successfully receive the signal in the form of ‘1010101010101010’ (See Figure in USRP Hardware Implementation). However, considering DQPSK has four phases while DBPSK has only two, so DQPSK has a higher efficiency.

# Experimental results demonstrating the operation of the system

## Software Implementation

### Observation Vector Construction

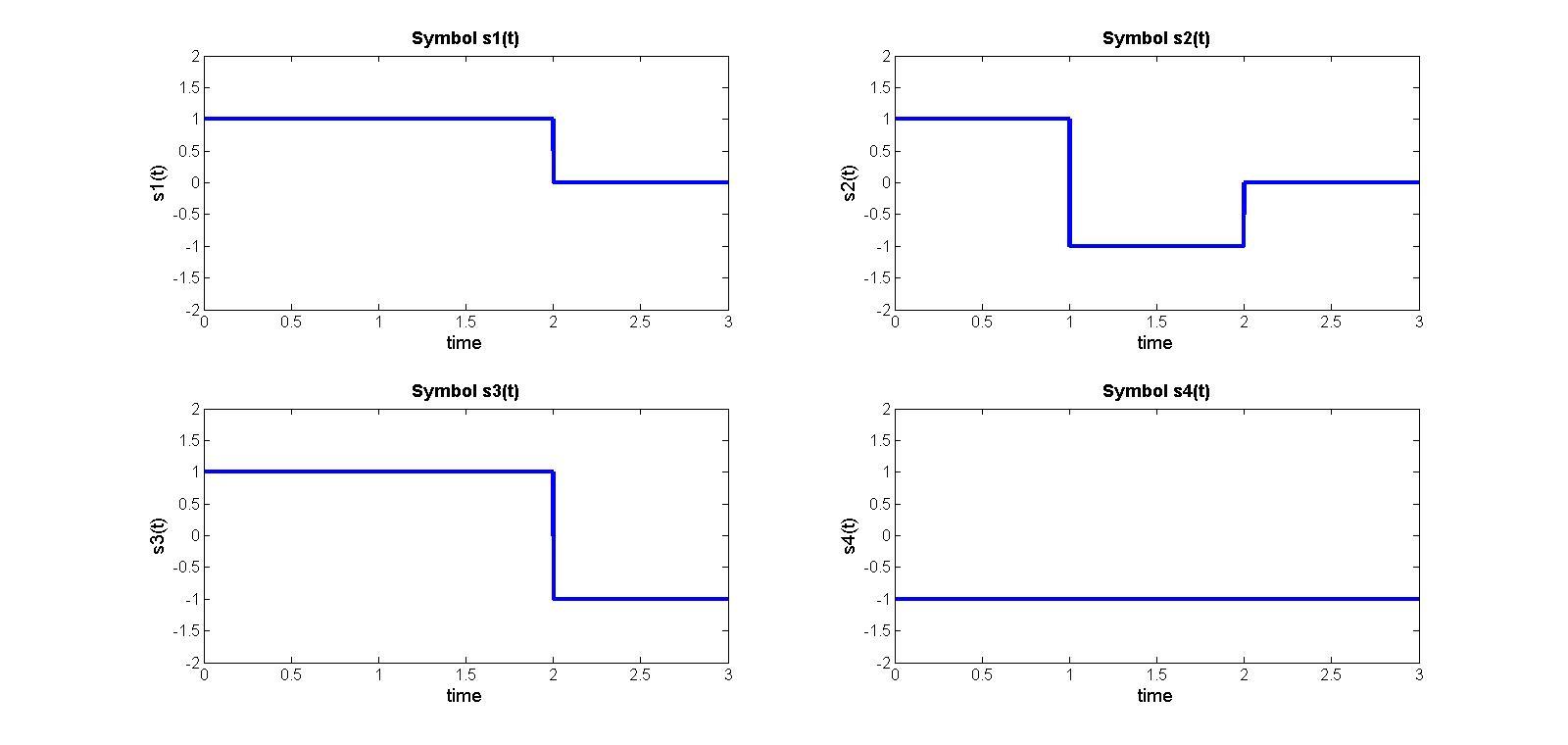


Figure : the 4 symbols, s1, s2, s3, s4

The simulation results are shown in question section

### Maximum-Likelihood Decoder Implementation

There was no error as long as we subtracted half the energy. When the energy was not subtracted the

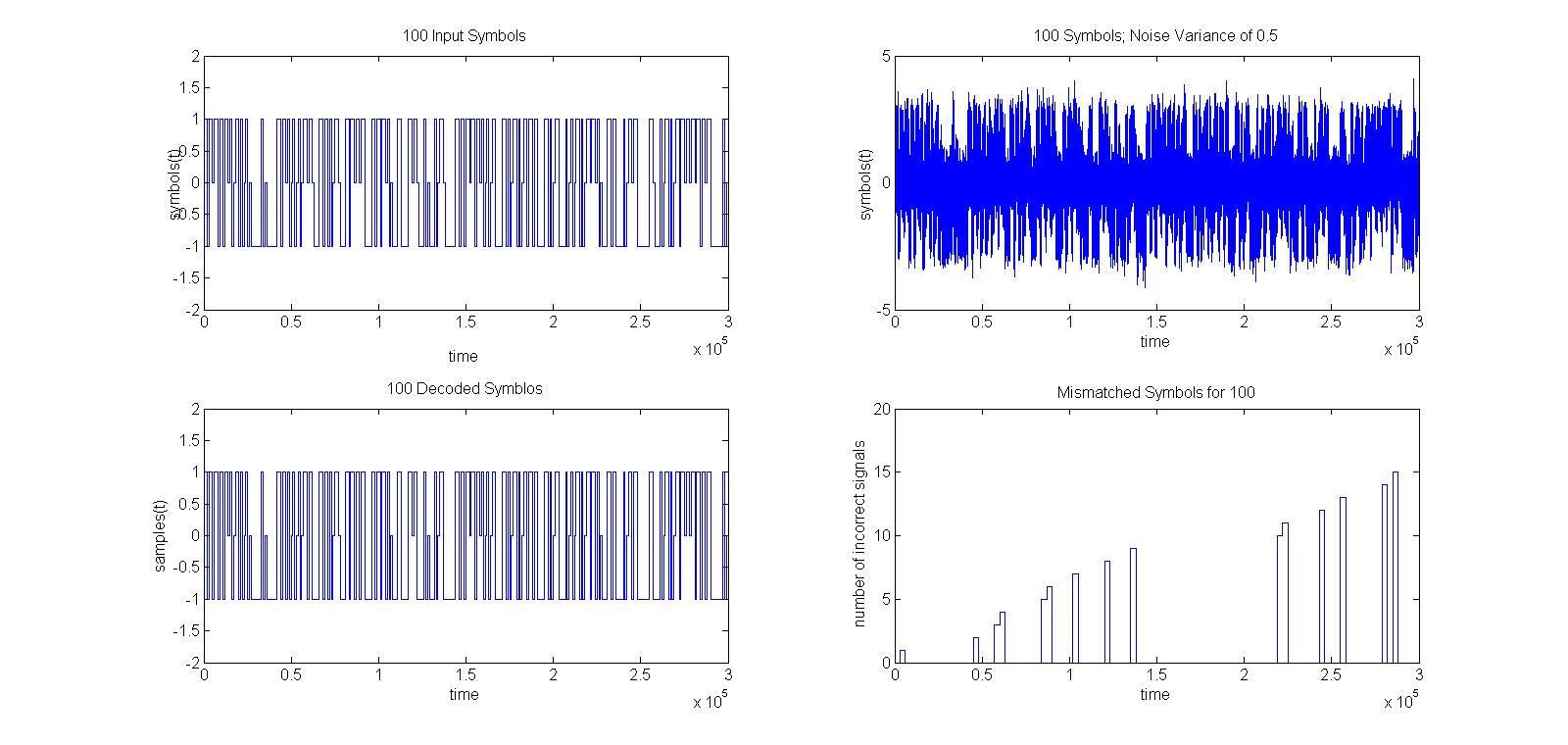


Figure : errors without the energy subtraction

The set of graphs that I have in word shows the input signal, the transmitted signal and the decoded signal as well as the graph which indicate at what time the errors occur as well as the number of errors over time. The first and last error in the 100 bit transmission are shown, both should have decoded to symbol 1 but instead were determined to be symbol 3. The error rate is approximately 12.5%. The only errors that we have observed while deciding these symbols has been symbol 1 decoding as symbol 3. The symbols are randomly distributed with each occurring 25% of the time. This would imply that symbol 1 is only properly decoded around half the time. This is a huge amount of error from one symbol.

### Correlator Realization of a Receiver in Simulink

The results are shown in the question section

## USRP Hardware Implementation

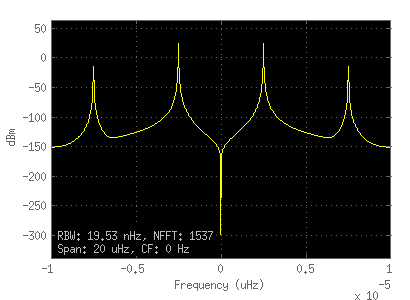


Figure 18 the output of the transmitter and a graph after the DPBSK demodulator in the receiver

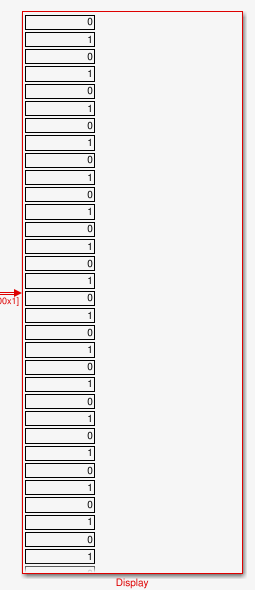


Figure : The output of the DBPSK tranceiver. The results for the DQPSK were identical.

## Frame Synchronization

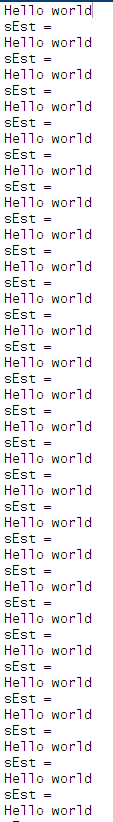


Figure the output of the frame synchronization output

# Conclusion

In this lab, we finished both software implementation and hardware implementation. In software implementation of the lab, we designed and implemented two different receivers, the matched filter and correlator-based receivers.

In hardware implementation section, two modulation schemes, DBPSK and DQPSK, are implemented using USRP hardware. In a DBPSK system, the input binary sequence is modulated using a DBPSK modulator differentially. One significant advantage of a DBPSK modulation scheme is that we do not need to worry about the carrier recovery as DBPSK could estimates and compensates for frequency and phase differences between a received signal’s carrier wave and the receiver’s local oscillator. In addition, we used Callback function to initiate the parameters such as the Sample time and Samples per frame. One thing needs to pay attention at the receiver end is the downsampling rate of the Raised Cosine Receive Filter. As the Mueller-Muller block also performed downsample for the purpose of calculation, the downsampling rate of the receiver filter is half of the upsampling of the transmitter. Otherwise, we can only receive half samples of the originals.

Then, we performed another experiment with DQPSK modulator scheme and we compared the performance of DQPSK with DBPSK. Even though we could receive correct signals from both systems, considering DQPSK has four phases while DBPSK has only two, so DQPSK has a higher efficiency.

Finally, a frame synchronization approach is designed. It is important for the receiver to know where each frame starts, otherwise the receiver may start to decode in the middle of the frame, which will result scrambled information. The frame synchronization is implemented by attaching Barker Code with the frame. We first test the system with frame synchronization using Barker code in the Simulink. Next, after we receive the correct message at the receiver, we began to transmit messages over the air via USRP N210. Finally, we received the message, i.e., ‘hello world’, from another USRP successfully.

# Appendix 1: Matlab code for the Software Implementation

## correlator.m

clear all;

close all;

clc;

number=100; % total transmission time (300s)/each signal duration (3s)

% randomly choose one of the four equiprobable signals

seedUsed = rng;

sampleRate = 1000;

% 1000 samples per second means 3000 samples per signal

time = linspace(0,3,3\*sampleRate);

nsamples = length(time);

s1 = ones(1,nsamples);

s1(2\*sampleRate:end) = 0;

s2 = ones(1,nsamples);

s2(1:sampleRate-1) = 1;

s2(sampleRate:(2\*sampleRate)-1) = -1;

s2(2\*sampleRate:end) = 0;

s3 = ones(1,nsamples);

s3(1:(2\*sampleRate)-1) = 1;

s3(2\*sampleRate:end) = -1;

s4 = -1\*ones(1,nsamples);

figure();

subplot(2,2,1)

plot(time,s1);

xlabel('time')

ylabel('s1(t)')

title('Symbol s1(t)')

subplot(2,2,2)

plot(time,s2);

xlabel('time')

ylabel('s2(t)')

title('Symbol s2(t)')

subplot(2,2,3)

plot(time,s3);

xlabel('time')

ylabel('s3(t)')

title('Symbol s3(t)')

subplot(2,2,4)

plot(time,s4);

xlabel('time')

ylabel('s4(t)')

title('Symbol s4(t)')

% zero mean white Gaussian noise of variance 0.5 added

outputTotal = zeros(1, nsamples \* number);

totalTime = linspace(0,3\*number,nsamples \* number);

inputTotal = zeros(1, nsamples \* number);

sumInput = zeros(1, number);

sumOutput = zeros(1, number);

variance = 0.5;

for indexNumber = 1:number

sdNoise = sqrt(variance);

noiseArray = sdNoise.\*randn(1,nsamples);

nextSignal = randi(4);

tempInput = 0;

switch nextSignal

case 1

tempInput = s1;

case 2

tempInput = s2;

case 3

tempInput = s3;

case 4

tempInput= s4;

end

tempOutput = tempInput + noiseArray;

inputTotal(1+((indexNumber-1)\*nsamples):indexNumber\*nsamples) = tempInput;

outputTotal(1+((indexNumber-1)\*nsamples):indexNumber\*nsamples) = tempOutput;

end

figure();

subplot(1,2,1)

plot(totalTime,inputTotal);

xlabel('time(s)')

ylabel('symbols(t)')

ylim([-5 5])

title([num2str(number),' symbols for input',]);

subplot(1,2,2)

plot(totalTime,outputTotal);

xlabel('time(s)')

ylabel('symbols(t)')

title([num2str(number),' symbols with AWGN with variance of ', num2str(variance) ]);

ylim([-5 5])

% Define the orthonormal functions {fm(t)}

fm1 = ones(1,nsamples);

fm1(1:(2\*sampleRate)-1) = 1;

fm1(2\*sampleRate:end) = -1;

fm2 = zeros(1,nsamples);

fm2(2\*sampleRate:end) = 1;

fm3 = zeros(1,nsamples);

fm3(sampleRate:(2\*sampleRate)-1) = -1;

figure();

subplot(2,2,1)

plot(time,fm1);

xlabel('time')

ylabel('fm1(t)')

title('Orthonormal function fm1(t)')

subplot(2,2,2)

plot(time,fm2);

xlabel('time')

ylabel('fm2(t)')

title('Orthonormal function fm2(t)')

subplot(2,2,3)

plot(time,fm3);

xlabel('time')

ylabel('fm3(t)')

title('Orthonormal function fm3(t)')

% Integration to get observation vector (ov)

sumOutputFM = zeros(4, number);

sumOutputSignalS = zeros(4, number);

for indexNumber = 1:number

currentReceivedSymbol = outputTotal(1+((indexNumber-1)\*nsamples):indexNumber\*nsamples);

multi1 = (currentReceivedSymbol .\* s1);

sumOutput(1,indexNumber) = sum(multi1);

multi2 = (currentReceivedSymbol .\* s2);

sumOutput(2,indexNumber) = sum(multi2);

multi3 = (currentReceivedSymbol .\* s3);

sumOutput(3,indexNumber) = sum(multi3);

multi4 = (currentReceivedSymbol .\* s4);

sumOutput(4,indexNumber) = sum(multi4);

end

% Plot

timeAxisSum = linspace(0,3\*number,number);

figure();

subplot(2,2,1)

stem(timeAxisSum,sumOutput(1,:));

xlabel('time')

ylabel('Received Signal \* s1(t)')

title('Integration S1(t) \* X')

subplot(2,2,2)

stem(timeAxisSum,sumOutput(2,:));

xlabel('time')

ylabel('Received Signal \* s2(t)')

title('Integration S2(t) \* X')

subplot(2,2,3)

stem(timeAxisSum,sumOutput(3,:));

xlabel('time')

ylabel('Received Signal \* s3(t)')

title('Integration S3(t) \* X')

subplot(2,2,4)

stem(timeAxisSum,sumOutput(4,:));

xlabel('time')

ylabel('Received Signal \* s4(t)')

title('Integration S4(t) \* X')

## decoder.m

% clear;

% clc;

% sv are signal vectors

sv =zeros(4,3);

sv(1,:)=[ 2/sqrt(3) 2/sqrt(6) 0];

sv(2,:)=[ 0 0 sqrt(2)];

sv(3,:)=[ sqrt(3) 0 0];

sv(4,:)=[-1/sqrt(3) -4/sqrt(6) -1];

%E(i) is the energy

E = [0 0 0 0];

for i=1:4

E(i) = sv(i,1)^2+sv(i,2)^2+sv(i,3)^2;

end

% Accumulator and subtraction

for j = 1:number

Xs1(j) = dot(outputTotal(1+((j-1)\*3000):j\*3000), s1);

Xs2(j) = dot(outputTotal(1+((j-1)\*3000):j\*3000), s2);

Xs3(j) = dot(outputTotal(1+((j-1)\*3000):j\*3000), s3);

Xs4(j) = dot(outputTotal(1+((j-1)\*3000):j\*3000), s4);

end

figure() % accumulator figure (without subtracting the energy)

subplot(2,2,1)

plot(Xs1);

xlabel('time')

ylabel('X\*s1(t)')

title('Accumulator for X\*s1(t)')

subplot(2,2,2)

plot(Xs2);

xlabel('time')

ylabel('X\*s2(t)')

title('Accumulator for X\*s2(t)')

subplot(2,2,3)

plot(Xs3);

xlabel('time')

ylabel('X\*s3(t)')

title('Accumulator for X\*s3(t)')

subplot(2,2,4)

plot(Xs4);

xlabel('time')

ylabel('X\*s4(t)')

title('Accumulator for X\*s4(t)')

Ys1 = Xs1-(E(1)/2);

Ys2 = Xs2-(E(2)/2);

Ys3 = Xs3-(E(3)/2);

Ys4 = Xs4-(E(4)/2);

% Select largest

decoded\_symbols\_with\_energy = X; % for size

for j= 1:number

if ((Xs1(j)> Xs2(j))&&(Xs1(j)> Xs3(j))&&(Xs1(j)> Xs4(j)))

decoded\_symbols\_with\_energy(1+((j-1)\*3000):j\*3000) = s1;

elseif ((Xs2(j)> Xs3(j))&&(Xs2(j)> Xs4(j)))

decoded\_symbols\_with\_energy(1+((j-1)\*3000):j\*3000) = s2;

elseif (Xs3(j)> Xs4(j))

decoded\_symbols\_with\_energy(1+((j-1)\*3000):j\*3000) = s3;

else

decoded\_symbols\_with\_energy(1+((j-1)\*3000):j\*3000) = s4;

end

end

decoded\_symbols\_without\_energy = X; % for size

for j= 1:number

if ((Ys1(j)> Ys2(j))&&(Ys1(j)> Ys3(j))&&(Ys1(j)> Ys4(j)))

decoded\_symbols\_without\_energy(1+((j-1)\*3000):j\*3000) = s1;

elseif ((Ys2(j)> Ys3(j))&&(Ys2(j)> Ys4(j)))

decoded\_symbols\_without\_energy(1+((j-1)\*3000):j\*3000) = s2;

elseif (Ys3(j)> Ys4(j))

decoded\_symbols\_without\_energy(1+((j-1)\*3000):j\*3000) = s3;

else

decoded\_symbols\_without\_energy(1+((j-1)\*3000):j\*3000) = s4;

end

end

%q is the difference between the input and the decoded symbols

q\_with = decoded\_symbols\_with\_energy-inputTotal;

errors\_with\_energy = 0;

for j= 1:number

if sum(q\_with(1+((j-1)\*3000):j\*3000))~=0

errors\_with\_energy = errors\_with\_energy+1;

q\_with(1+((j-1)\*3000):j\*3000) = errors\_with\_energy\*ones(1:3000,1);

end

end

percent\_errors\_with\_energy = 100\*errors\_with\_energy/number;

%q is the difference between the input and the decoded symbols

q\_without = decoded\_symbols\_without\_energy-inputTotal;

errors\_without\_energy = 0;

for j= 1:number

if sum(q\_without(1+((j-1)\*3000):j\*3000))~=0

errors\_without\_energy = errors\_without\_energy+1;

q\_without(1+((j-1)\*3000):j\*3000) = errors\_without\_energy\*ones(1:3000,1);

end

end

percent\_errors\_without\_energy = 100\*errors\_without\_energy/number;

error\_comparison = ['Without subtracting the symbol energies, there are ',num2str(percent\_errors\_with\_energy),'% errors in ', num2str(number),' symbols.';

'Removing the energy of the symbols, there are ', num2str(percent\_errors\_without\_energy),'% errors in ', num2str(number), ' symbols. ']

% Plot with energy

figure();

subplot(1,2,1)

subplot(2,2,1)

plot(inputTotal);

xlabel('time')

ylabel('symbols(t)')

title([num2str(number),' Input Symbols']);

subplot(2,2,2)

plot(X);

xlabel('time')

ylabel('symbols(t)')

title([num2str(number),' Symbols; Noise Variance of ', num2str(variance) ]);

subplot(2,2,3)

plot(decoded\_symbols\_without\_energy);

xlabel('time')

ylabel('samples(t)')

title([num2str(number), ' Decoded Symblos']);

subplot(2,2,4)

plot(q\_without);

xlabel('time')

ylabel('number of incorrect signals')

title(['Mismatched Symbols for ',num2str(number)]);

## Correlator Realization of a Receiver in Simulink

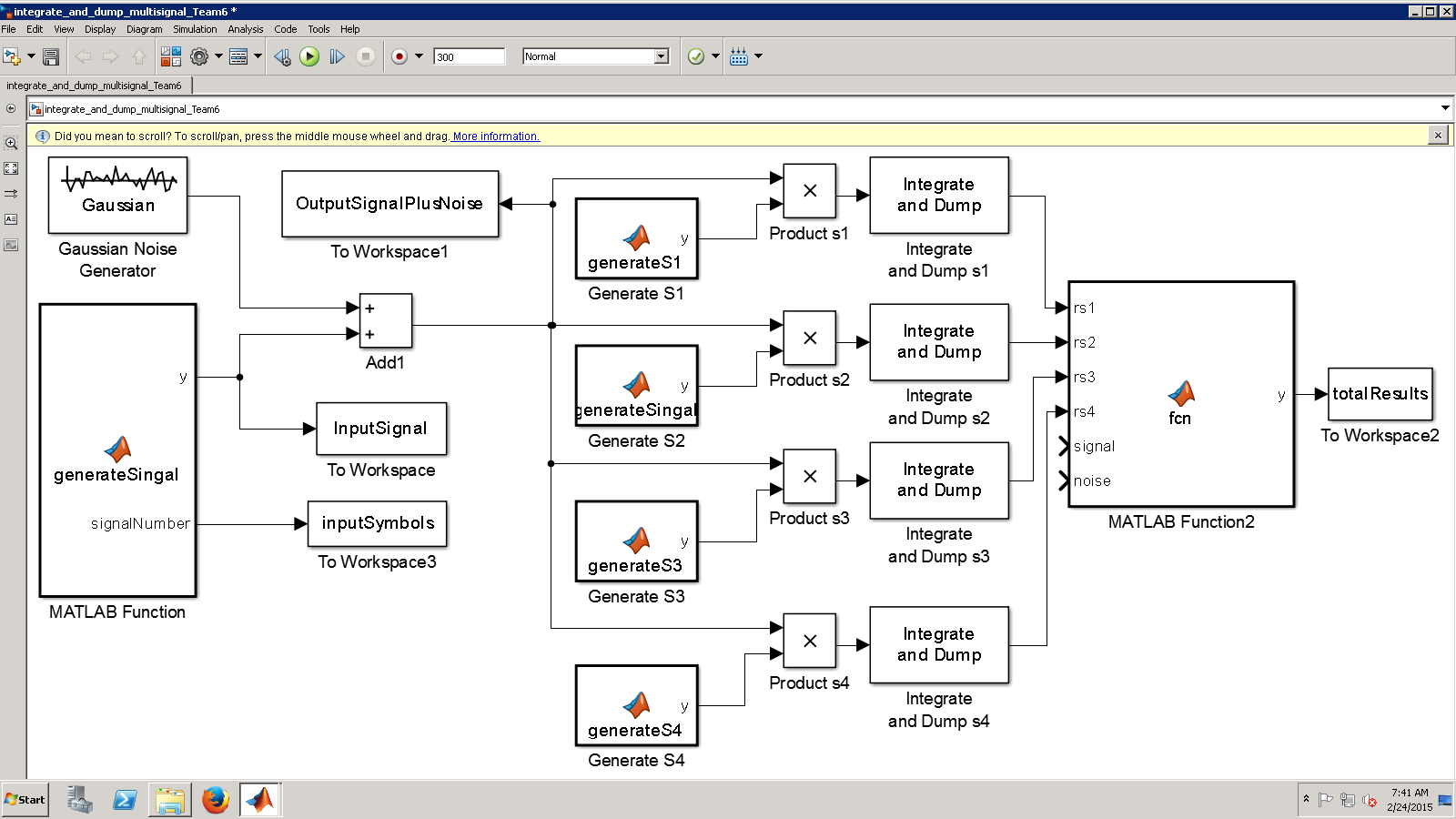
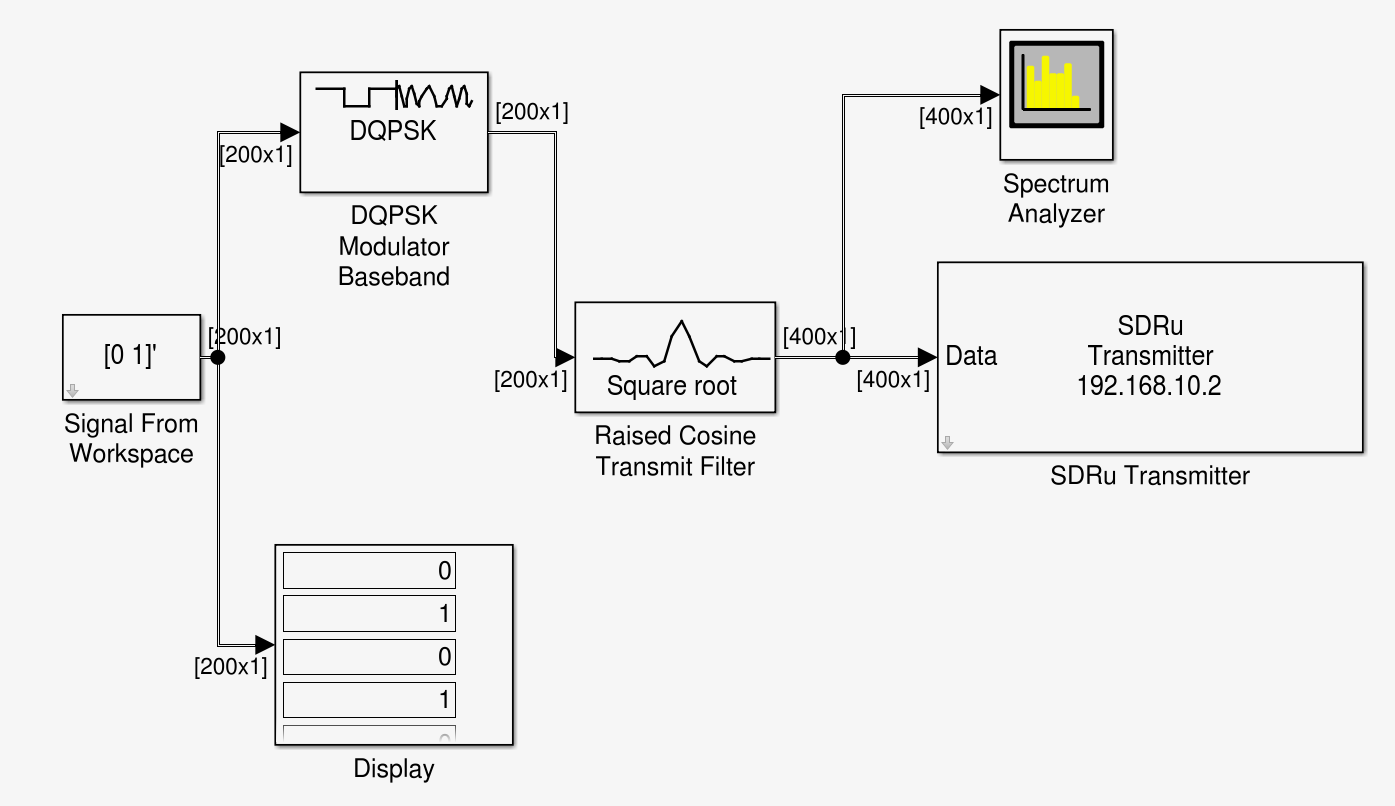
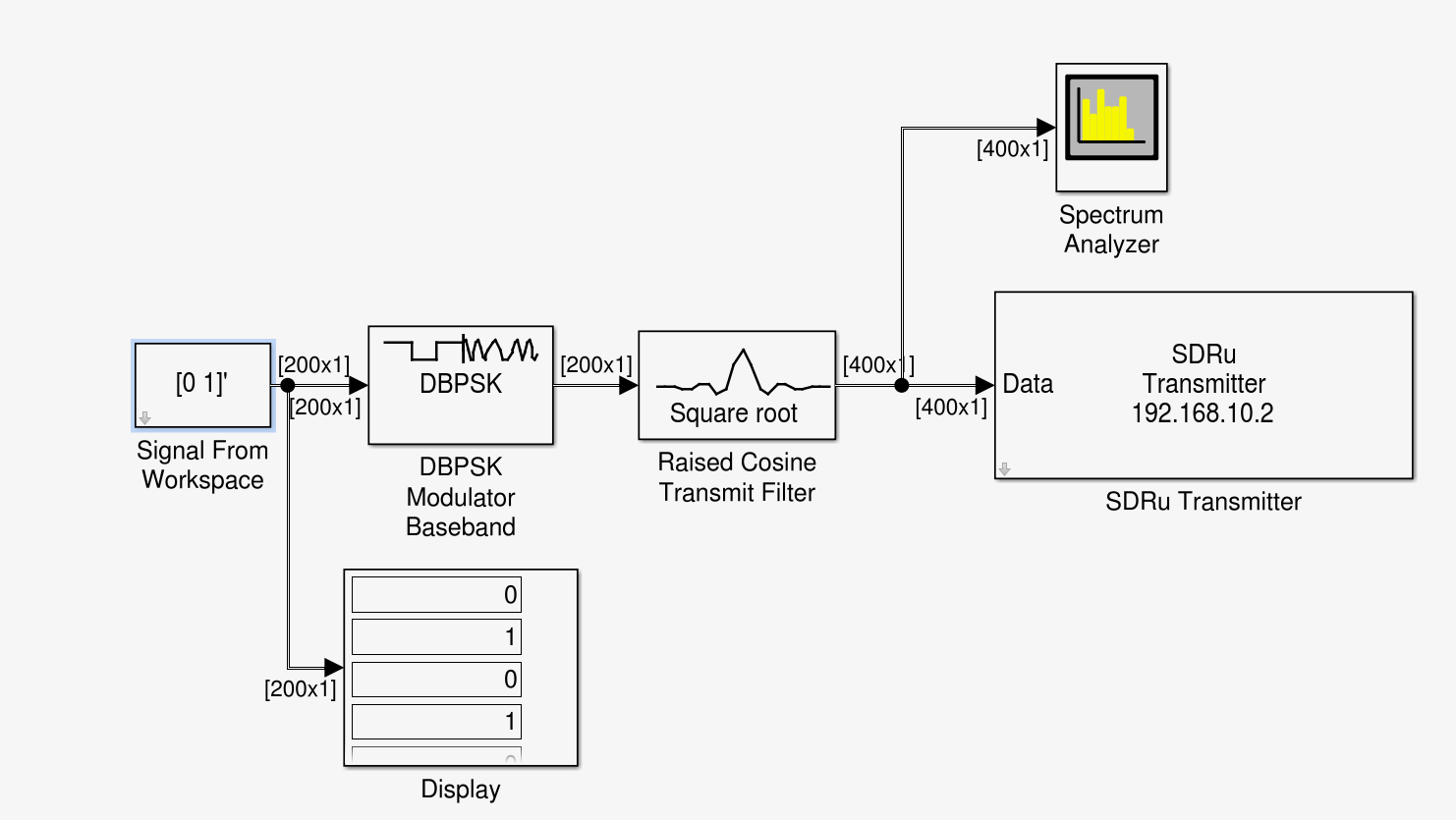


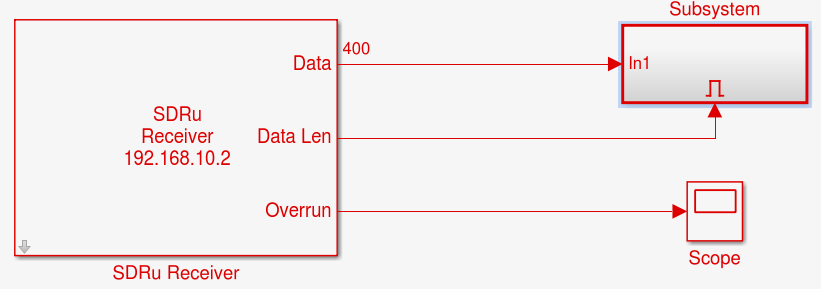
Figure 20 Correlator implementation in Simulink with decision block to find the best match of the transmitted symbol

Appendix 2: Simulink Models for the USRP Hardware Implementation

## DBPSK/DQPSK transmitter

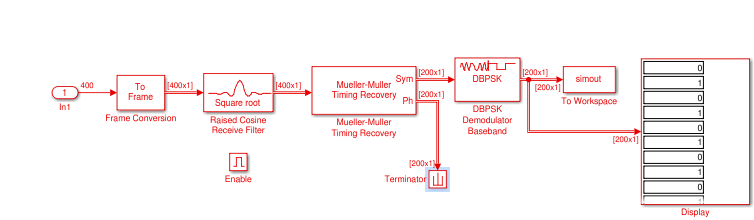


## DBPSK Receiver



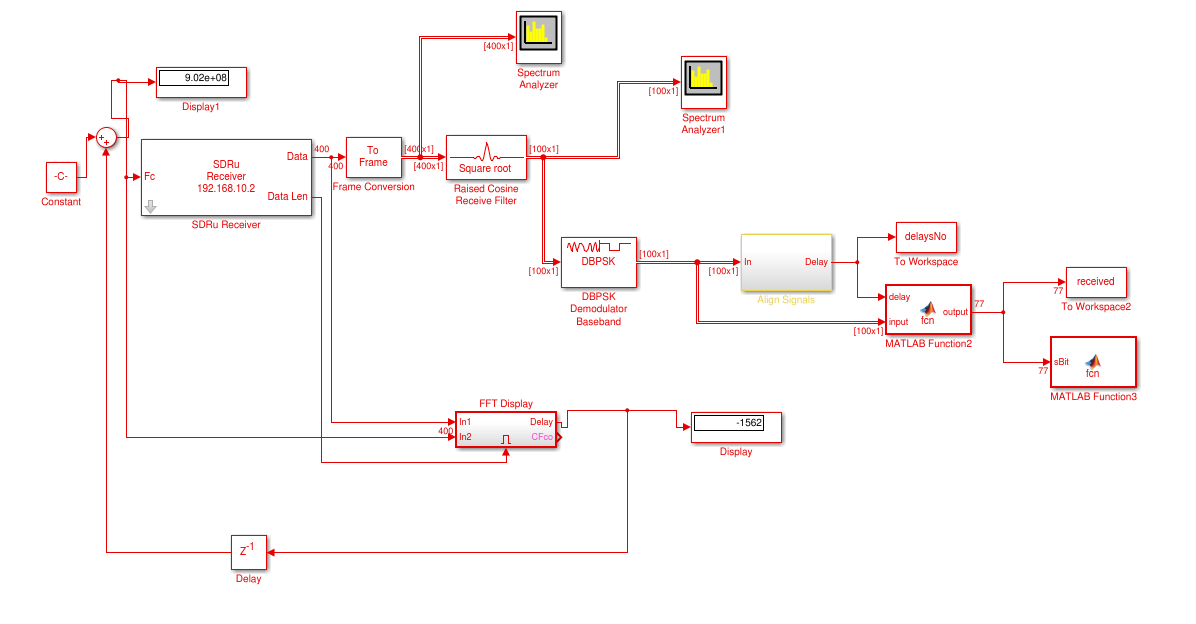
## 

DBPSK Receiver Subsystem



Appendix 3

## Frame Synchronization Receiver



Frequency Correction block

