Lab Report No 10

Digital Wireless communication

Submitted By: Muhammad Ali

Registration No: 19pwcse1801

Submitted to : Sir Ihsan-ul-haq

Section: A
Date: 25/2/2022

Department of Computer Systems Engineering
University of Engineering and Technology Peshawar

Course Outline: -

Simulate a Basic Digital Communications Link: -

Implement the essential components of a single carrier digital communications system, including additive white Gaussian noise.

Pulse Shaping Filters

Incorporate transmit and receive filters into your simulation.

Multipath Channels

Model a multipath channel and evaluate the impact on link quality.

OFDM

Implement orthogonal frequency-division multiplexing (OFDM), a multicarrier modulation scheme.

n this activity, you'll simulate a simple back-to-back communications link that uses *quadrature amplitude modulation*, or QAM. You'll generate a source signal of random bits, modulate the source bits using QAM, demodulate the QAM symbols back into bits, and compare the demodulated bits to the source bits to see if they match.

16-QAM is a common single carrier modulation scheme. It maps 4 input bits to one of 16 complex numbers, called *symbols*. For each complex-valued symbol, the real and imaginary parts represent the in-phase and quadrature components, respectively, of a waveform.

The "16" in 16-QAM is the *modulation order*. It's useful to store the modulation order in a variable, so it can be used throughout your simulation.

Let's look at the constellation of the modulated QAM signal. To do this, use the scatterplot function. This function plots the imaginary part (quadrature) vs. the real part (in-phase) of each complex-valued QAM symbol.

```
scatterplot(y)
```

What you see in the scatter plot is the ideal 16-QAM constellation — 16 complex-valued symbols arranged in a square, with each symbol equally spaced from its neighbors. The greater the separation between points, the better the error rate performance. Later, when you add noise to the simulation, you'll see the received observations depart from this ideal configuration.

For now, the simulation assumes the modulated signal is unchanged when it's received by the receiver.

In the receiver, demodulation turns the 16-QAM symbols back into bits.

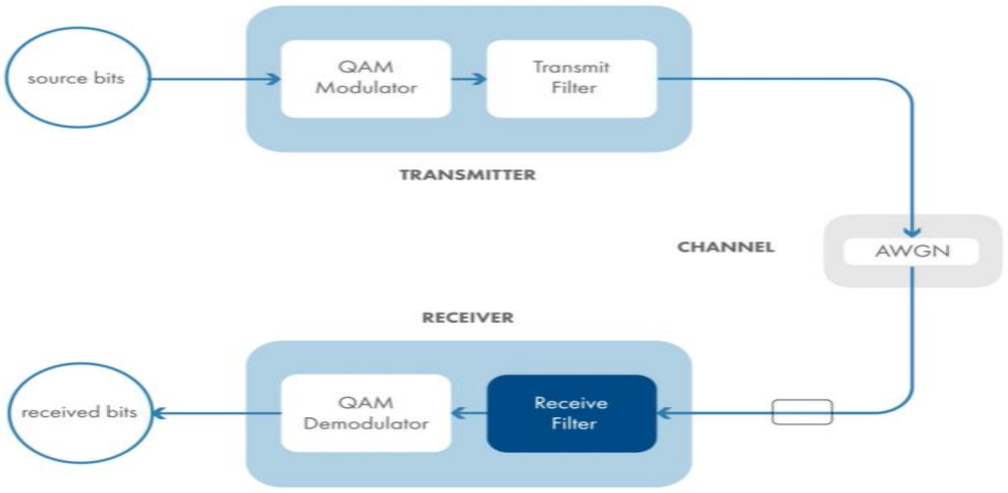
You can use the qamdemod function to demodulate the received signal. It has almost the same syntax as qammod. The only difference is that qamdemod specifies the "OutputType".

```
z = qamdemod(y,modOrder,"OutputType","bit")
```

The output z is a column vector of 1s and 0s – the received bits.

```
numBits = 20000

srcBits = randi([0,1],numBits,1)
```



Task 2

```
modOrder = 16
```

Task 3

```
modOut = qammod(srcBits,modOrder,"InputType","bit")
```

Task 4

```
scatterplot(modOut)
```

Task 5

```
chanOut = modOut;
```

Task 6

```
demodOut = qamdemod(chanOut,modOrder,"OutputType","bit")
```

Since the 16-QAM signal was not subject to any channel effects, the received bits should match the source bits exactly.

To see if two vectors are identical, use the isequal function.

```
check = isequal(x1,x2)
```

The output check is a logical variable with value 1 if the two vectors are identical and 0 if they are not.

Task 7

```
check = isequal(srcBits,demodOut)
```

Adding Noise: -

A common corruption, or noise source, in the received signal is due to electron motion from the receiver's electronics. This noise is modeled as part of the channel.

The noise is called *additive white Gaussian noise*, or AWGN, because it's typically *added* to the signal, has a flat (or *white*) spectral density, and has a *Gaussian* probability density function.

In this activity, you'll add noise to your 16-QAM link simulation and note its effects.

The awgn function applies *additive white Gaussian noise*, or AWGN, to its input signal. The amount of noise added is specified as a signal-to-noise ratio, assuming the input signal has an average power of 1.

Fortunately, you can tell the qammod function to output a signal with an average power of 1. Simply use a second property name-value pair to set "UnitAveragePower" to true.

```
sigOut = qammod(bits,modOrder,...
    "InputType","bit",...
    "UnitAveragePower",true)
```

Note: Have you ever had trouble reading a long line of code? You can use ... to split a long command into multiple lines.

Next, model the noisy channel by applying AWGN to the modulated signal.

You can use the awgn function to apply AWGN to a signal. The noise power added is specified as a signal-to-noise ratio, SNR, measured in decibels (dB).

```
SNR = 7  % dB
sigOut = awgn(sig,SNR)
```

To see the impact of the noise, you can view the constellation of the channel output using the scatterplot function, as before.

```
scatterplot(sig)
```

Now you can see that the noise has perturbed the symbols from their ideal values.

The qamdemod function uses a "nearest neighbor" decision rule to map received observations back to the nearest ideal symbol constellation point. If an observation has been perturbed too much, it will get mapped to the wrong point. This results in symbol errors, in turn yielding bit errors.

Since you set the "UnitAveragePower" property when you called qammod, you must also set it in qamdemod.

```
sigOut = qamdemod(sig,modOrder,...
    "OutputType","bit",...
    "UnitAveragePower",true)
```

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code sets up the simulation.

Task 1

```
modOut = qammod(srcBits,modOrder,"InputType","bit","UnitAveragePower",true);
```

Task 2

```
SNR = 15

chanOut = awgn(modOut,SNR)
```

Task 3

```
scatterplot(chanOut)
```

Task 4

```
demodOut = qamdemod(chanOut,modOrder,"OutputType","bit","UnitAveragePower",true);

check = isequal(srcBits,demodOut)
```

Calculating the Bit Error Rate : -

Noise can introduce errors in the received bits. Comparing each bit sent to the corresponding bit received tells you if an error was made. The fraction of bits that contain bit errors is a key way to measure the performance of a communications system.

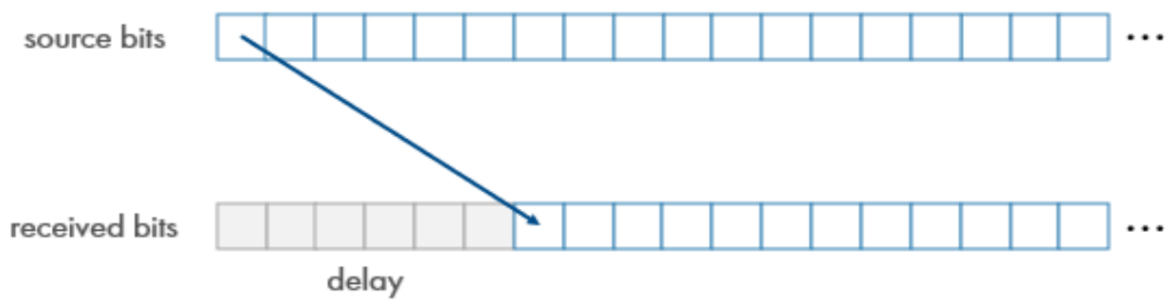
To count bit errors, you start by performing a bit-by-bit comparison of the source bits and the received bits.

The logical operator ~= ("not equal") compares two arrays element-by-element.

```
isErr = x~=z
```

The output isErr is a logical array. It has the value 1 where the elements of x and z are **not** equal and 0 where they are.

The bit error rate, or BER, is calculated by dividing the number of bit errors by the total number of bits.



Task 1

```
isBitError = srcBits~=demodOut
```

Task 2

```
numBitErrors = nnz(isBitError)
```

Task 3

```
BER = numBitErrors/numBits
```

Transmit and Receive Filters: -

In this activity, you'll add square-root raised cosine filters to the transmitter and receiver. The transmit filter takes the 16-QAM signal as input, and its output is sent over the channel. The receive filter takes the channel output as its input, and its output is passed to the QAM demodulator.

Because systems typically use matched filters, it's common to create the transmit and receive filters at the same time.

You can create a square-root raised cosine transmit filter using the following syntax.

```
    txFilt = comm.RaisedCosineTransmitFilter
```

A similar syntax creates the matched receive filter.

```
    rxFilt = comm.RaisedCosineReceiveFilter
```

The filters have default properties already set. If you leave off the semicolon, you can see some of the properties in the output panel. For example, the property OutputSamplesPerSymbol has the default value 8, which means the filter upsamples the signal so it has 8 samples for each QAM symbol.

Notice that txFiltOut is 8 times longer than modOut, which is to be expected based on the filter property OutputSamplesPerSymbol.

Next, you want to apply AWGN to the filtered signal. However, the transmit filter changed the signal power. To scale the noise power appropriately, you can pass a third input to the awgn function.

```
    sigOut = awgn(sig,SNR,"measured")
```

When you use the "measured" option, the function calculates the input signal's power and scales the noise power based on the signal-to-noise ratio.

Now apply the receive filter to the signal. Like with the transmit filter, you can pass the signal as an input to the stored filter.

```
    sigOut = rxFilt(sig)
```

Task 1

```
txFilt = comm.RaisedCosineTransmitFilter

rxFilt = comm.RaisedCosineReceiveFilter
```

Task 2

```
txFiltOut = txFilt(modOut);
```

Task 3

```
SNR = 7;

chanOut = awgn(txFiltOut,SNR,"measured");
```

Task 4

Demodulate back into bits. This code only executes after rxFiltOut has been created.

```
if exist("rxFiltOut","var")  % code runs after you complete Task 4

   scatterplot(rxFiltOut)

   demodOut = qamdemod(rxFiltOut,modOrder,"OutputType","bit","UnitAveragePower",true);

end
```

Transmit and receive filters incur delays on the signal, which must be accounted for when the two bit sequences are compared.

For each filter, the delay is half the filter length, in symbols.

You can extract the filter length by accessing the FilterSpanInSymbols property. Then divide by 2 to get the delay.

```
txFilt.FilterSpanInSymbols/2
rxFilt.FilterSpanInSymbols/2
```

The total delay is the sum of the transmit and receive filter delays.

However, to compare bits, you need to determine the delay in bits, not symbols.

For 16-QAM, four bits are mapped to one symbol. This conversion factor has been stored in the variable bitsPerSymbol, defined in the script.

You can multiply by bitsPerSymbol to convert the delay from symbols to bits.

```
delayInSymbols * bitsPerSymbol
```

First, get the aligned bits from the source bit sequence by extracting all but the last delayInBits bits.

```
srcBits(1:(end-delayInBits))
```

Next, get the aligned bits from the received bit sequence by extracting all but the first delayInBits bits.

```
demodOut((delayInBits+1):end)
```

Now you can compare the aligned bits and count the number of bit errors.

Remember you can use ~= to perform a bit-by-bit comparison to find nonmatching bits, and then use nnz to count.

```
    nnz(bitSeq1 ~= bitSeq2)
```

the aligned bit stream is shorter than the original.

```
    numAlignedBits = length(srcAligned)
```

To calculate the bit error rate, you need to divide the number of bit errors by the number of aligned bits.

```
numBits = 20000;

modOrder = 16;  % for 16-QAM

bitsPerSymbol = log2(modOrder)  % modOrder = 2^bitsPerSymbol

txFilt = comm.RaisedCosineTransmitFilter;

rxFilt = comm.RaisedCosineReceiveFilter;



srcBits = randi([0,1],numBits,1);
modOut = qammod(srcBits,modOrder,"InputType","bit","UnitAveragePower",true);

txFiltOut = txFilt(modOut);



SNR = 7;  % dB
chanOut = awgn(txFiltOut,SNR,"measured");



rxFiltOut = rxFilt(chanOut);
demodOut = qamdemod(rxFiltOut,modOrder,"OutputType","bit","UnitAveragePower",true);
```

Task 1

```
delayInSymbols = rxFilt.FilterSpanInSymbols/2 + txFilt.FilterSpanInSymbols/2
```

Task 2

```
delayInBits = delayInSymbols * bitsPerSymbol
```

Task 3

```
srcAligned = srcBits(1:(end-delayInBits))
```

Task 4

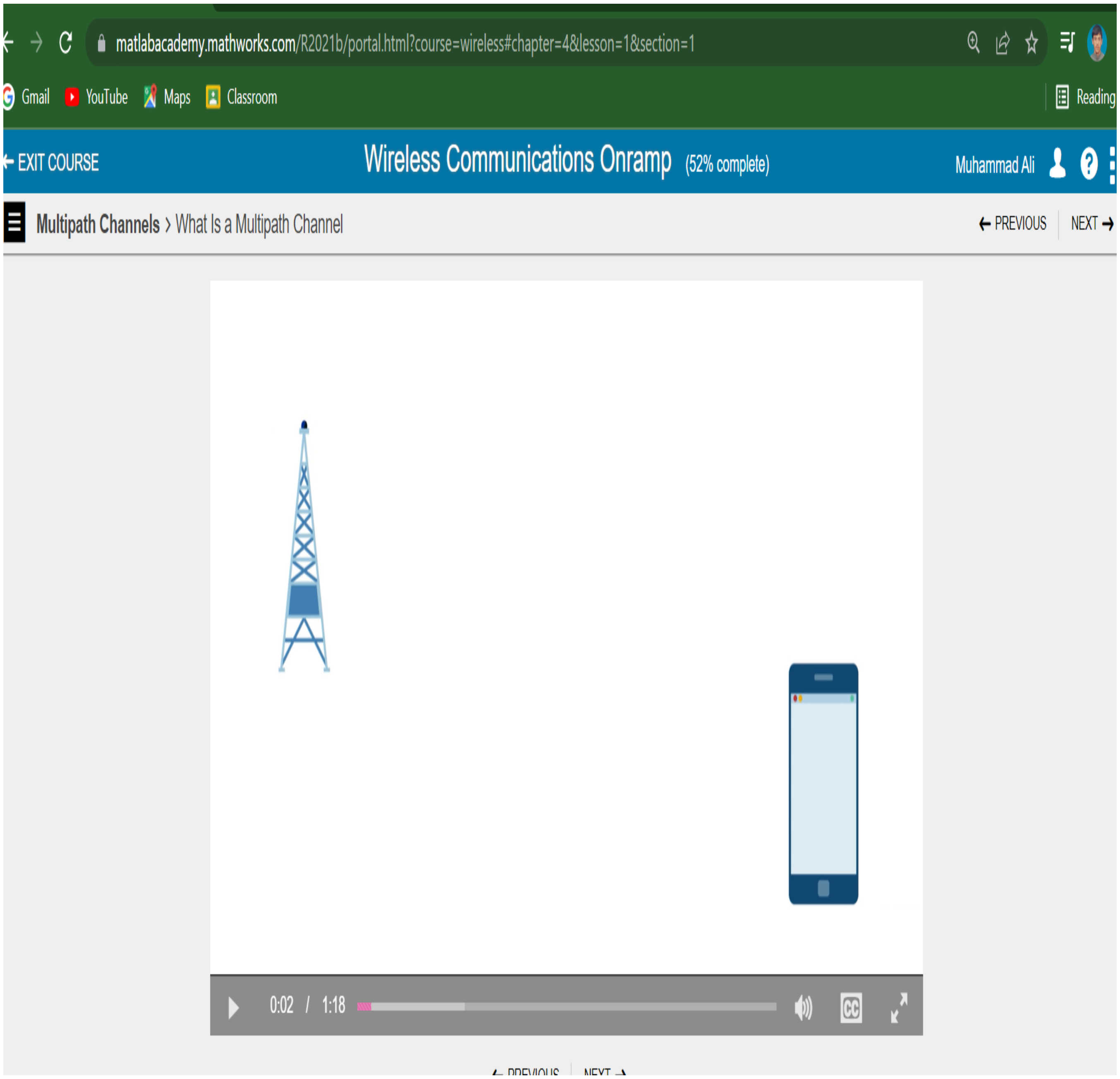
```
demodAligned = demodOut((delayInBits+1):end)
```

Task 5

```
numBitErrors = nnz(srcAligned ~= demodAligned)
```

Task 6

```
 numAlignedBits = length(srcAligned);
BER = numBitErrors/numAlignedBits
```



matlabacademy.mathworks.com/R2021b/portal.html?course=wireless#chapter=4&lesson=1&section=1

Wireless Communications Onramp (52% complete)

Multipath Channels › What Is a Multipath Channel