Recap

- Inter-Symbol Interference (ISI)
  - Handle band limit ISI
  - Handle multipath ISI
    - Viterbi
      - problems: Its complexity grows exponentially with $D$ (the number of multipaths taps relative to the symbol time)
      - $Q$: how to reduce $D$?

Benefit of Symbol Rate on ISI

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

OFDM: Basic Idea

- Uses multiple carriers modulation (MCM)
  - each carrier (called a subcarrier) uses a low symbol rate
    - for $N$ parallel subcarriers, the symbol time can be $N$ times longer
  - spread symbols across multiple subcarriers
    - also gains frequency diversity
**Multiple Carrier Modulation (MCM): Problem**

- Despite wave shaping, there can be leak from one subcarrier to another subcarrier
- Conventional design: guard bands to avoid interference among subcarriers
- Guard band wastes spectrum

**Objective: Avoid subcarrier interference**

- Interference of subcarrier \(i\) on subcarrier \(j\)

Assume no pulse wave shaping, matched filter

\[
\int_0^T \sin(2\pi f_i t + \phi_i) \sin(2\pi f_j t + \phi_j) dt = \frac{1}{2} \int_0^T \cos(2\pi(f_i - f_j) t + \phi_i - \phi_j) + \cos(2\pi(f_i + f_j) t + \phi_i + \phi_j) dt
\]

Condition for the interference to be always 0?

**OFDM Key Idea: Orthogonal Subcarriers**

- Each subcarrier frequency is chosen so that an integral number of cycles in a symbol period, i.e.,
  - subcarrier freq = \(k \frac{1}{T}\)

They do not need to have the same phase, so long integral number of cycles in symbol time \(T\)!

**OFDM Modulation**

OFDM allows overlapping subcarriers frequencies

**Orthogonal Frequency Division Multiplexing**

- OFDM allows overlapping subcarriers frequencies

http://www1.linksys.com/products/images/ofdm.gif
OFDM Implementation

- Take N symbols and place one symbol on each subcarrier (freq.)

\[
\begin{align*}
\text{Freq}_0 & : d_n e^{j 2\pi n f_s c T} \\
\text{Freq}_1 & : d_n e^{j 2\pi (n-1) f_s c T} \\
\vdots & \\
\text{Freq}_{N-1} & : d_n e^{j 2\pi (N-1) f_s c T}
\end{align*}
\]

- Q: complexity of the implementation strategy?

OFDM: Implementation Issue

- Hardware implementation can be expensive if we use one oscillator for each subcarrier
- Software implementation requires N multiplications per time output \( \Rightarrow N^2 \) multi. per N outputs

\[
\begin{align*}
\text{Freq}_0 & : d_n e^{j 2\pi n f_s c T} \\
\text{Freq}_1 & : d_n e^{j 2\pi (n-1) f_s c T} \\
\vdots & \\
\text{Freq}_{N-1} & : d_n e^{j 2\pi (N-1) f_s c T}
\end{align*}
\]

OFDM: Key Idea 2

- Consider data as coefficients in the frequency domain, use inverse Fourier transform to generate time-domain sequence

\[
\begin{align*}
X_k = \sum_{n=0}^{N-1} x_n e^{-j 2\pi nk/N} \\
x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j 2\pi nk/N} \\
\text{out}_t = \sum_{n=0}^{N-1} d_n e^{j 2\pi n (f_s c T)/N} \\
\text{Assume N outputs per symbol time } T, f_s = 1/T
\end{align*}
\]

- Consider data as coefficients in the frequency domain, use inverse Fourier transform to generate time-domain sequence

OFDM Implementation: FFT

- Parallel data streams are used as inputs to an IFFT
- IFFT does multiplexing and modulation in one step!

Guard Interval: Removing ISI

- Orthogonal subcarriers remove inter-carrier interference
- Slow symbol rate reduces inter-symbol interference, but may still have ISI

- Basic idea of GI: skip the first part “damaged” signal

More details: Chap. 13.1.4 Gast
High-throughput interfaces are needed to connect these functional blocks in their PHY components. These PHY processing blocks directly operate on the digital signal in the radio front-end and PHY as well as between some functional blocks. Data are streamed through these blocks sequentially, but with different data types and sizes. As illustrated in Figure 1, functional blocks are pipelined with one another. Data needs for implementing a software radio system on a general PC platform:

- The interfaces between the radio front-end and PHY as well as between some functional blocks.
- High system throughput.

Given the above discussion, we summarize the requirements for a software radio system on a general purpose OS.

### 2.3 Software Radio Requirements

- The interfaces between the radio front-end and PHY as well as between some functional blocks. Data are streamed through these blocks sequentially, but with different data types and sizes. As illustrated in Figure 1, functional blocks are pipelined with one another. Data needs for implementing a software radio system on a general PC platform:

- **Subcarrier frequency spacing 312.5kHz**
  - 1/312.5kHz = 3.2us
  - 44 samples FFT
  - 16 samples Guard Interval

- **Multicarrier System**: OFDM
  - Subcarrier frequency spacing 312.5kHz
  - 16 samples Guard Interval

- **802.11a**
  - 2.4 GHz band
  - 40 MHz bandwidth
  - 64QAM modulation

- **Other Multipath Techniques**
  - There are other techniques to handle multipath such as Rake Receiver
  - See backup slides for some details

- **Summary of PHY**

- **Wireless PHY**

- **OFDM Guard Interval**

- **OFDM in 802.11a**

- **OFDM Implementation**
Overview

- Mobile/Wireless software development framework for mobile wireless applications is a quite large topic
- We have already seen Gnuradio as an example framework
- We will cover more examples
  - TinyOS, J2ME, Android, IOS
- Approach for designing/evaluating each software development framework:
  - Focus on the key concepts introduced by each framework

Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio

GNURadio: Design Objective

- A software development toolkit that provides signal processing blocks to implement software-defined radio systems.

GNURadio Hardware Arch

Hardware Frontend

- RF Frontend
- ADC/DAC and Digital Frontend (USRP)

Host Computer

GNURadio Software

http://mobiledevices.kom.aau.dk/fileadmin/mobiledevices/teaching/software_testing/Gnu_radio_lecture.pdf
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNU Radio
    - Hardware setting
    - Software concepts

Basic Software Concepts

- gr_basic_block (name, in/out signature, msg queue)
  - gr_block (Leaf block; key functions forecast/generic_work)
- gr_hier_block2 (container block; key functions: connect/disconnect/lock/unlock)
  - gr_top_block (flow graph: start/stop/wait)

Software/Execution Model

- Software model
  - Python
    - Application management (e.g., GUI)
    - Flow graph construction
    - Non-streaming code (e.g., MAC layer)
  - C++
    - Signal processing blocks
    - Certain routines also coded in assembly
- Execution model
  - Python thread for each top_block

Discussion: benefits/issues of the hybrid software structure?

Summary: GNU Radio

- Interesting/key software design techniques you learned from GNU Radio?

Outline

- Admin and recap
- Mobile/wireless development framework
  - GNU Radio
    - Hardware setting
    - Software concepts
  - TinyOS
Design Goal

- A free and open source component based operating system and platform targeting wireless sensor networks (WSNs)
- Example app
  - Environment monitoring, e.g.,
    - measure temperature, lighting values/events
    - periodically transmit measurements/events to a base station
    - forward data for other nodes that are out of range of the base station

http://www.tinyos.net/tinyos-1.x/doc/tutorial/

Hardware

- Assembled from off-the-shelf components
- 4Mhz, 8bit MCU (ATMEL)
  - 512 bytes RAM, 8KB ROM
- Devices
  - serial Port
  - temperature sensor & light sensor
  - 900Mhz Radio (RF monolithics)
    - 10-100 ft. range
  - LED outputs

Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
    - Hardware setting
    - Software concepts
  - TinyOS
    - Hardware setting
    - Software concepts

TinyOS: Software Concept

- TinyOS: Generate customized OS + application for each given scenario
  - support one application at a time but flexible reprogramming
A TinyOS consists of one or more components linked together.
- software components motivated by hardware component

Each component specifies that
- it provides some interfaces
  - allows other components to control it
- also uses some interfaces
  - control other components

An interface declares
- a set of functions called commands that provider must implement
- another set of functions called events that the interface user must implement

A uses interfaces I1 and I2

B provides I1

C provides I3

C provides I2
Multipath Diversity: Rake Receiver

- Instead of considering delay spread as an issue, use multipath signals to recover the original signal
- Used in IS-95 CDMA, 3G CDMA, and 802.11
- Invented by Price and Green in 1958
  

- Use several "sub-receivers" each delayed slightly to tune in to the individual multipath components
- Each component is decoded independently, but at a later stage combined
  - this could very well result in higher SNR in a multipath environment than in a "clean" environment

Rake Receiver Blocks

- Correlator
- Finger 1, Finger 2, Finger 3
- Combiner

Rake Receiver: Matched Filter

- Impulse response measurement
- Tracks and monitors peaks with a measurement rate depending on speeds of mobile station and on propagation environment
- Allocate fingers: largest peaks to RAKE fingers

Rake Receiver: Combiner

- The weighting coefficients are based on the power or the SNR from each correlator output
- If the power or SNR is small out of a particular finger, it will be assigned a smaller weight:
  \[
  \alpha_m = \frac{Z_m^2}{\sum_{j=1}^{M} Z_j^2}
  \]

Comparison [PAH95]