SOFTWARE DEFINED RADIO
USR SDR WORKSHOP, SEPTEMBER 2017
PROF. MARCELO SEGURA

SESSION 5: QPSK TX/RX IMPLEMENTATION
In the case M-PSK modulation, due to rotational symmetry, the synchronization could lock to any of the M phases.

To solve this situation, differential encoding can be used or Data Aided rotation.
Differential encoding maps the data to the phase shift instead to the absolute phase.

For BPSK, $M=2$, is very simple implementation using an XNOR.

<table>
<thead>
<tr>
<th>$b_i[n]$</th>
<th>$b_{e}[n-1]$</th>
<th>$b_{e}[n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
If we have a phase error of $\pi$, the encoded and decoded data is as follow:

**ENCODER**

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_r[n]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b_\epsilon[n-1]$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b_\epsilon[n]$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**DECODER**

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_\epsilon[n]$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b_\epsilon[n-1]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b_r[n]$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Differential Encoding: QPSK

- $M=4$
### DIFFERENTIAL ENCODING: QPSK

**Example**

\[ b_t = \{0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1\} \]

<table>
<thead>
<tr>
<th>ENCODER</th>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_t[n] )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b_t[n+1] )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( b_e[n-2] )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b_e[n-1] )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( b_e[n] )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( b_e[n+1] )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**DECODER**

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_e[n-2] )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( b_e[n-1] )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b_e[n] )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( b_e[n+1] )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b_r[n] )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( b_r[n+1] )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Phase error \( \pi \)
- From now we just work on the **PHY** layer, then we need to move to the Data Link Layer.

**COMM PROTOCOL**

- **Main Operations**
  - Software generating and accessing data
  - Data manipulation (encoding, compression etc)
  - Controls connections between nodes (handshake)
  - Segmentation of data, introduce ports
  - Packetise segments and consider addressing
  - Framing data and frame synch
  - DSP operations including modulation and Tx Rx

**Diagram:**

- **NODE 1**
  - 1. Physical Layer
  - 2. Data Link Layer
  - 3. Network Layer
  - 4. Transport Layer
  - 5. Session Layer
  - 6. Presentation Layer
  - 7. Application Layer

- **NODE 2**
  - 1. Physical Layer
  - 2. Data Link Layer
  - 3. Network Layer
  - 4. Transport Layer
  - 5. Session Layer
  - 6. Presentation Layer
  - 7. Application Layer

- Data flows from Node 1 to Node 2 via RF signals.
The basic task is to detect the start of a signal information. If we are transmitting text, encoded as ASCII, we need to detect the sequence start to decode properly.
The Header contains the information useful for synchronization.

Advantages: in cases of error, just re-transmit the lost part.

Frame structure with header and payload:

- Frame Length
- Frame information
- Part of the data stream is contained in the payload of each frame

Original message:

Message split into frames:
• Usually the Header contain a Preamble, that is a sequence with good autocorrelation properties.

• The preamble sequence is very important for start detection.

• **BARKER sequence** is one with have this characteristics.

![Autocorrelation Function of Barker Code with Length N=13](image-url)
The receiver implements a correlator, FIR with inverted coefficients, to detect the frame start, usually called Start Frame Delimiter (SFD).
• Depends on the type of communication systems, will be the load that header will introduce.
• In order to **get better correlation peaks**, the sequence can be repeated, this also reduce efficiency.

• Ex Barker N=26, Frame=200, Payload=174, efficiencies 174/200=87%
A threshold should be selected to capture the payload when the correlator output cross that level.

**Stage #1 - Receive the data bit stream**

**Stage #2 - Find correlation peak**

**Stage #3 - Reconstruct the frame**

**Stage #4 - Extract the payload**

*peak found at the end of the header’s preamble*
QPSK TX

- **Tx Model**

  ![Diagram](image)

  - **Bit Generation block:**
    - Data Frame = 200 bits
    - Header = 2 Baker 13 sequences (26 bits)
    - DataLenght = 200 - 26 = 174 bits
    - MessageLength = 7 bits (Ascii) x 15 (Hello World xxx) = 105 bits
    - Random bits to improve Scramble = 69 (Bernoulli Gen)
    - Scramble: use to reduce the number of consecutives ‘1’ or ‘0’, helps the synchronization.
QPSK TX

- Bit generator
AGC: The AGC is placed before the Raised Cosine Receive Filter so that the signal amplitude can be measured with an oversampling factor of four.
RRC: in this implementation decimate by 2
QPSK RX: COARSE FREQUENCY

- FFT implementation

\[ y(i) = u(i) \left( \cos \left( 2\pi \sum_{n=0}^{i-1} f(n)\Delta t + \varphi(i) \right) + j \sin \left( 2\pi \sum_{n=0}^{i-1} f(n)\Delta t + \varphi(i) \right) \right) \]
QPSK RX: FINE FREQUENCY

Function e = PED(In)

% Implementation of Maximum Likelihood Phase Error Detector listed in
% Chapter 7.2.2 of "Digital Communications - A Discrete-Time Approach" by
% Michael Rice. This employs a decision directed method.

% Input:
% In - Input samples, oversampled by two
% Output:
% e - Phase error, dc
% codegen

e = sign(real(In)).*imag(In) - sign(imag(In)).*real(In);
QPSK RX: TIME RECOVERY

\[ e(k) = x((k - 1/2)T_s + \hat{\tau}) \left[ \hat{a}(k - 1) - \hat{a}(k) \right] \]

```matlab
if strobe==1 && delayStrobe==strobe
    e = real(delay1) * (sign(real(delay2)) - sign(real(In))) + ...
    imag(delay1) * (sign(imag(delay2)) - sign(imag(In)));
else
    e = 0;
end
```

```matlab
if delayStrobe==strobe
    % Shift contents in delay register
    delay2 = delay1;
    delay1 = In;
elseif strobe==1
    % Two consecutive high strobos
    delay2 = 0; % Stuff missing sample
    delay1 = In;
end
```
QPSK RX: TIME RECOVERY

- Interpolation Filter

\[
y(k) = c_2 \cdot x(m_k - 2) + c_1 \cdot x(m_k - 1) + c_0 \cdot x(m_k) + c_1 \cdot x(m_k + 1)
\]

\[
c_2 = \alpha \mu_k^2 - \alpha \mu_k
\]
\[
c_1 = -\alpha \mu_k^2 + (\alpha + 1) \mu_k
\]
\[
c_0 = -\alpha \mu_k^2 + (\alpha - 1) \mu_k + 1
\]
\[
c_1 = \alpha \mu_k^2 - \alpha \mu_k
\]
\[
\alpha = -0.5 \text{ for Farrow Structure}
\]

```
persistent delay1 delay2 delay3; % Input delayed by 1, 2 and 3 samples
if isempty(delay1)
    [delay1, delay2, delay3] = deal(complex(0,0));
end
K = -0.5;

Out = delay2 + mu.*((K.*(In+delay2+delay3)+(1-K).*delay1)+
                        + K.*(delay1+delay2-In-delay3).*mu.^2);

% Update delay buffers
delay3 = delay2;
delay2 = delay1;
delay1 = In;
```
QPSK RX: TIME RECOVERY

- SYMBOL RECOVERY
QPSK RX: DATA DECODING
QPSK RX: DATA DECODING

- FRAME SYNC
QPSK RX: DATA DECODING

• Frame Sync

From Barker detector

function Data = aligndata(Delay, RxData)
% Extract data from the buffered RxData, using the
% start
% Input:
% Delay - Index of frame start
% RxData - Buffered symbol-spaced data
% Output:
% Data - Data with the Barker code at frame start
%#codegen
Data = RxData(Delay+1:Delay+length(RxData)/2);
QPSK RX: DATA DECODING

- Phase ambiguity correction: Data aided

```matlab
if In<pi/4 && In>=-pi/4
    Out = 0;
elseif In>=pi/4 && In<pi*3/4
    Out = pi/2;
elseif In>=-pi*3/4 && In<-pi*3/4
    Out = -pi;
else
    Out = -pi/2;
end
```
QPSK RX: DATA DECODING

- Header remove & Descramble
- Send data to workspace for ASCII conversion