Channel Sharing & Design of a multi-channel capacitive game controller

Joshua Smith
Intel Research Seattle

Outline

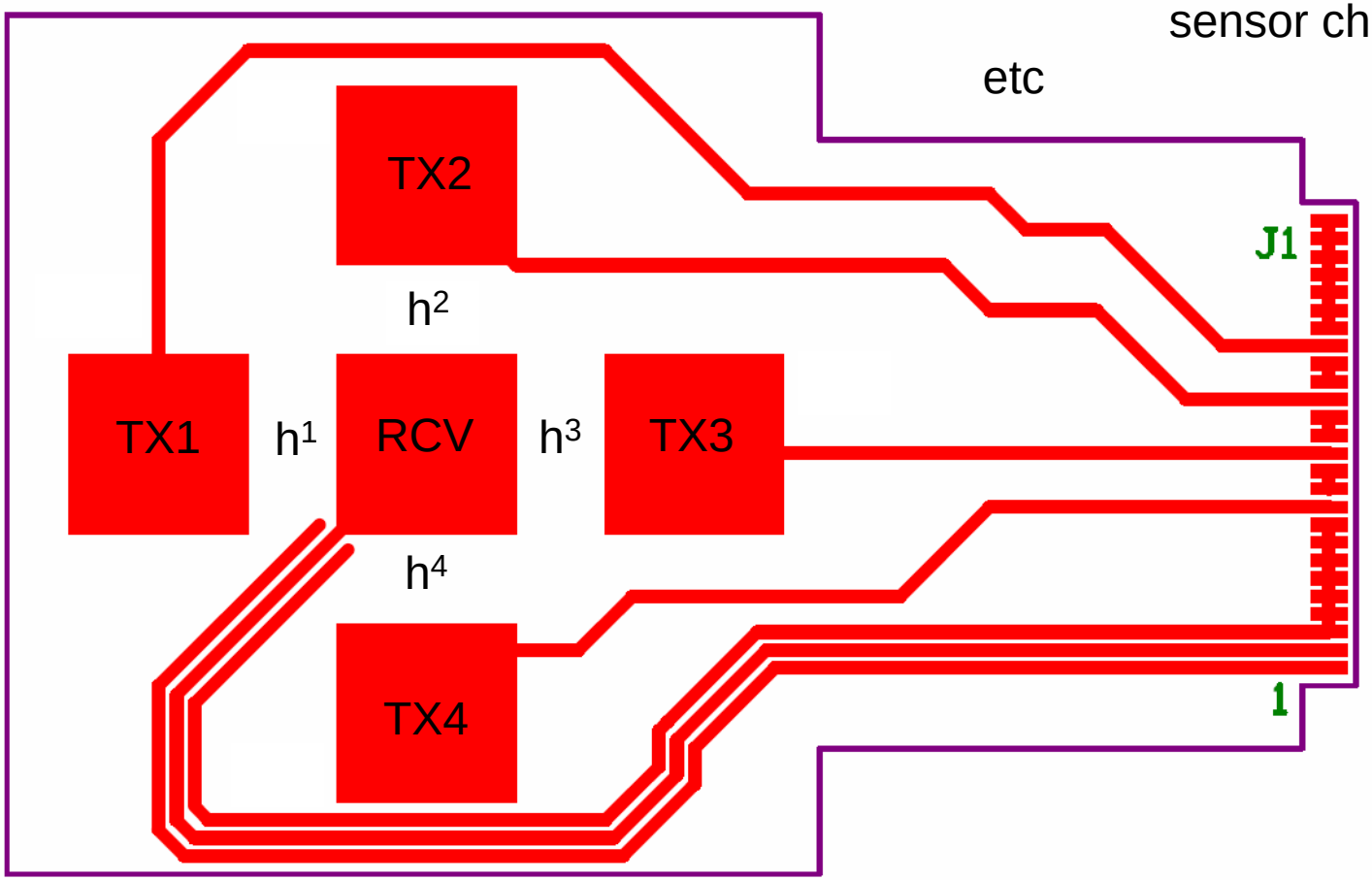
- Studied & built (lab 3) single e-field sensor channel
- Now we're going to make a multi-channel game controller
  - "Airstick" with 4 channels
  - Gives "2 ½ D" control: left/right, fwd/backward, some up/down
- Could simply make multiple copies of TX & RCV electrodes, and RCV electronics, but this is wasteful
- Channel sharing techniques will save resources
  - Just one set of analog electronics
- We'll view channel sharing from abstract vector point of view & in intuitive concrete fashion

Airstick electrodes

h^1: measurement of hand by sensor channel 1

h^2: measurement of hand by sensor channel 2

etc



Channel sharing methods

- TDMA (Time Division Multiple Access)
  - Talk/listen at different times for each channel
- FDMA (Frequency Division Multiple Access)
  - Talk/listen on different frequencies for each channel
- CDMA (Code Division Multiple Access)
  - Talk/listen on different “codes” for each channel
  - AKA Direct Sequence Spread Spectrum
  - Other spread spectrum methods:
    - frequency hopping
    - Chirp
  - Ultrawideband (UWB) does even more spreading

Lab 7

- Build a 4 channel CDMA electric field sensor
- Compare performance to TDMA EF sensor (given)

Review of Lab 3 material

```
acc = 0; // acc should be a 16 bit variable
For (i=0; i<NSAMPS; i++) {
            SET PORTB HIGH
            acc = acc + ADCVALUE
            SET PORTB LOW
            acc = acc - ADCVALUE
}
Return acc
```

Why is this implementing inner product correlation? Imagine unrolling the loop. We'll write ADC_1, ADC_2, ADC_3, … for the 1st, 2nd, 3rd, … ADCVALUE

$acc = ADC_1 - ADC_2 + ADC_3 - ADC_4 + ADC_5 - ADC_6 + \ldots$

$acc = +1*ADC_1 + -1*ADC_2 + +1*ADC_3 + -1*ADC_4 + \ldots$

$acc = C_1*ADC_1 + C_2*ADC_2 + C_3*ADC_3 + C_4*ADC_4 + \ldots$

where C_i is the ith sample of the carrier

acc = <C,ADC> Inner product of the carrier vector with the ADC sample vector

New convention: vectors are blue

Review

$acc =< C, ADC >$

Where C is the carrier vector and ADC is the vector of samples.
Let's write out ADC:

$$ADC = hC$$

where h (hand) is sensed value and hC means scalar $h\times$ vector C

$$
\begin{aligned}
&acc \\
&=< C, hC > \\
&= h < C, C > \\
&= h \\
&\text{if } < C, C >= 1
\end{aligned}
$$

Multi-access communication

Abstract view

Suppose we have two carriers, C^1 and C^2
And suppose they are orthogonal, so that $< C^1, C^2 >=0$
The received signal is

$$ADC = h^1 C^1 + h^2 C^2$$

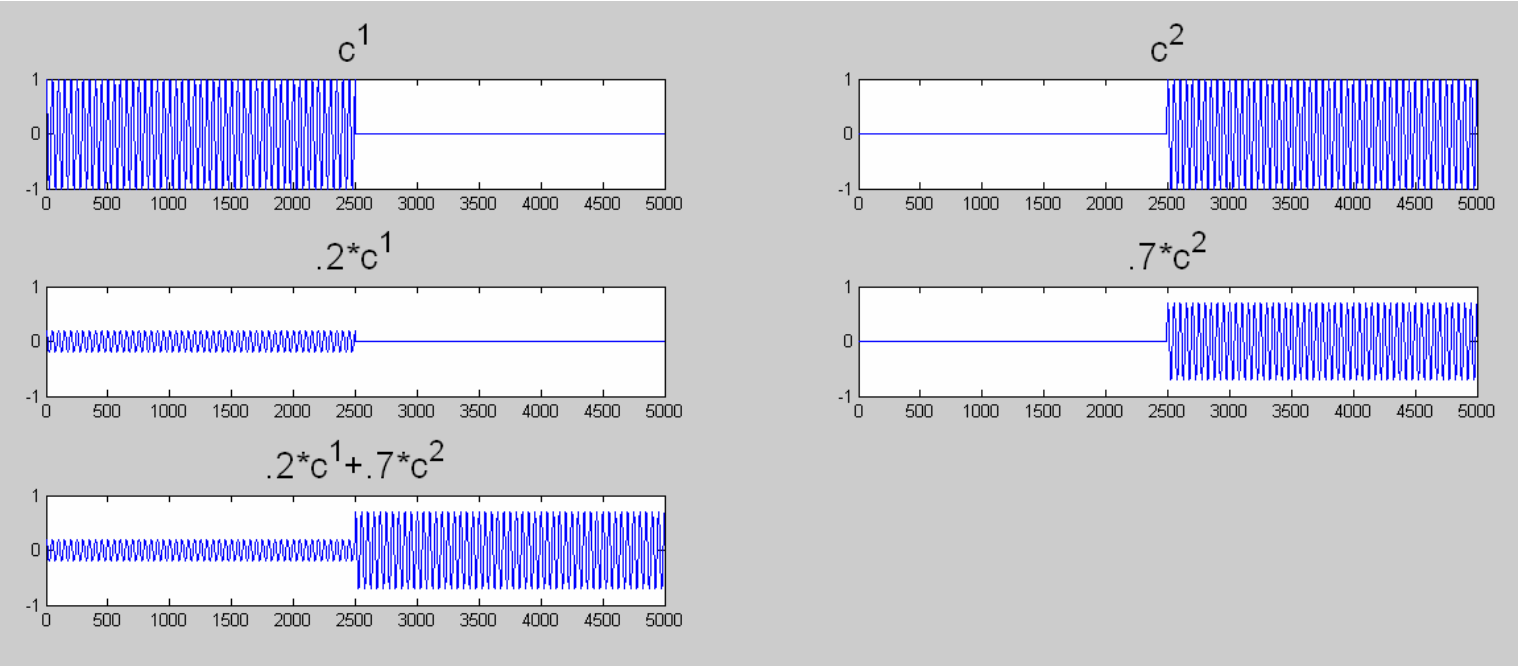
Let's demodulate with C^1:

$$\begin{aligned} & acc \\ =& < C^1, ADC > \\ =& < C^1, h^1 C^1 + h^2 C^2 > \\ =& < C^1, h^1 C^1 > + < C^1, h^2 C^2 > \\ =& \; h^1 < C^1, C^1 > + h^2 < C^1, C^2 > \\ =& \; h^1 \end{aligned}$$

$$\text{if } < C^1, C^1 >= 1 \text{ and } < C^1, C^2 >= 0$$

TDMA

Abstract view



Verify that $<C^1,C^2>=0$

Modulated carriers

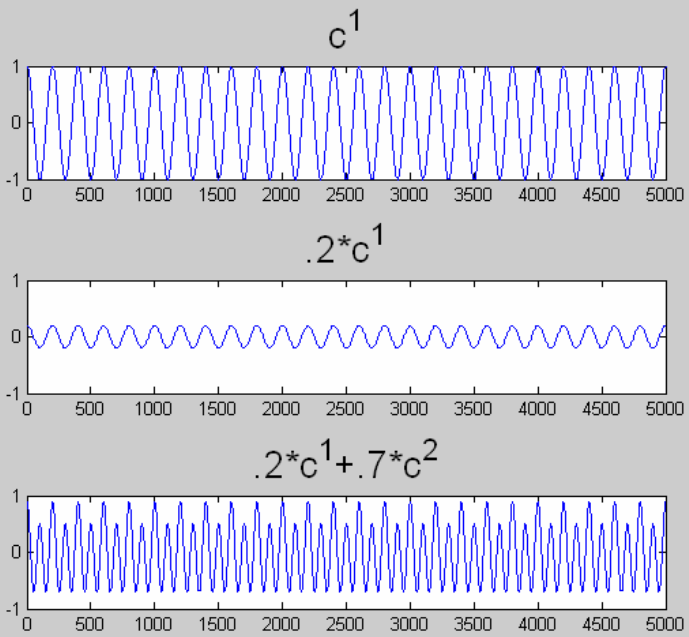
Sum of modulated carriers

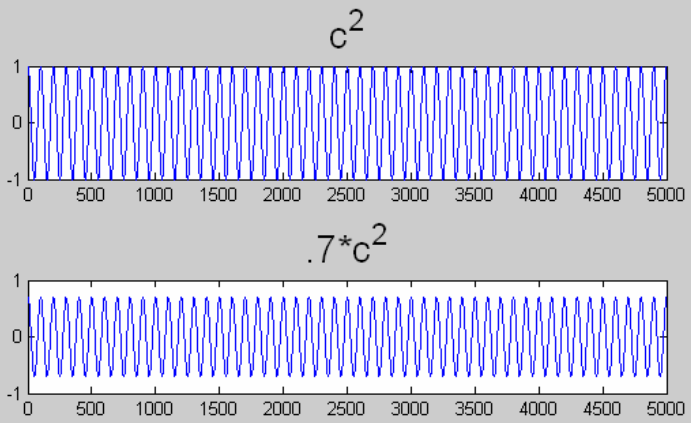
Horizontal axis: time
Vertical axis: amplitude (arbitrary units)

$<C^1, .2C^1 +.7C^2>=$
$<C^1, .2C^1> +<C^1,.7C^2>=$
$.2 <C^1, C^1> + 0$

FDMA

Abstract view



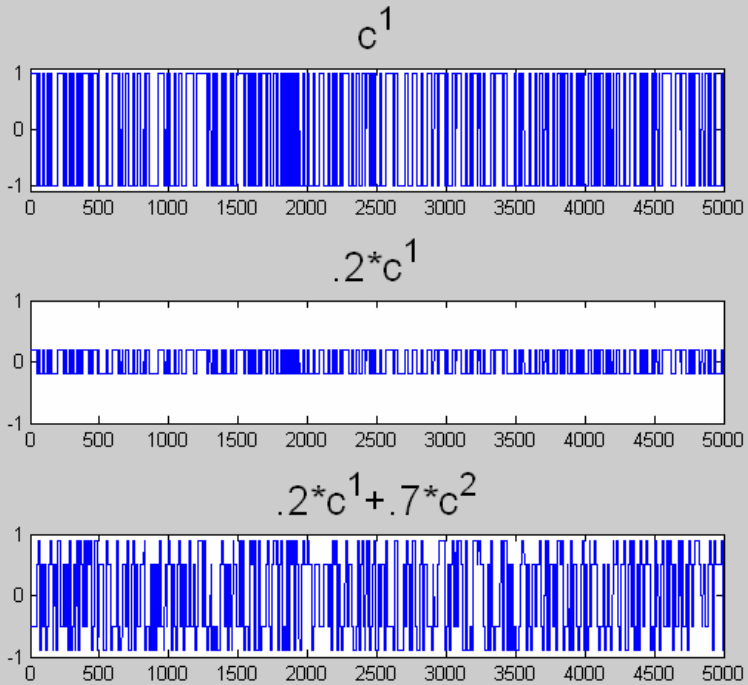


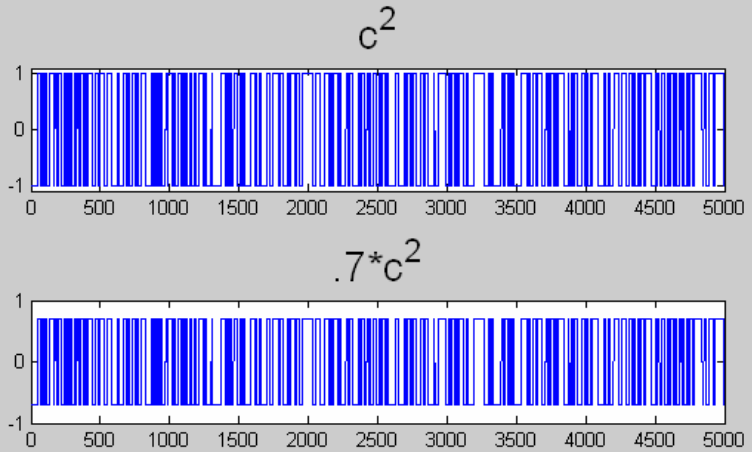

```
>> n1=sum(c1 .* c1)
n1 =  2.5000e+003

>> n2=sum(c2 .* c2)
n2 =  2.5000e+003

>> n12=sum(c1 .* c2)
n12 = -8.3900e-013

>> rcv = .2*c1 + .7*c2;
>> sum(c1/n1 .* rcv)
ans =    0.2000

>> sum(c2/n2 .* rcv)
ans =    0.7000
```

Horizontal axis: time
Vertical axis: amplitude (arbitrary units)

CDMA

S'pose we pick random carriers: `c1 = 2*(rand(1,500)>0.5)-1;`






```
>> n1=sum(c1 .* c1)
n1 =    5000

>> n2=sum(c2 .* c2)
n2 =    5000

>> n12=sum(c1 .* c2)
n12 =  -360

>> rcv = .2*c1 + .7*c2;
>> sum(c1/n1 .* rcv)
ans =  0.1496

>> sum(c2/n2 .* rcv)
ans =  0.6856
```

Horizontal axis: time
Vertical axis: amplitude (arbitrary units)

Note: Random carriers here consist of 500 rand values repeated 10 times each for better display

LFSRs (Linear Feedback Shift Registers)

The right way to generate pseudo-random carriers for CDMA

- A simple pseudo-random number generator
  - Pick a start state, iterate
- Maximum Length LFSR visits all states before repeating
  - Based on primitive polynomial…iterating LFSR equivalent to multiplying by generator for group
  - Can analytically compute auto-correlation
- Easy to compute in HW (not as nice in SW)
- Totally uniform auto-correlation

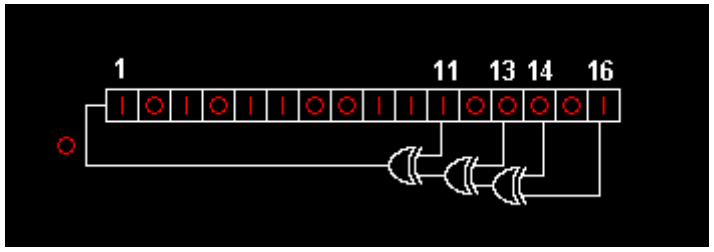


Image source: wikipedia

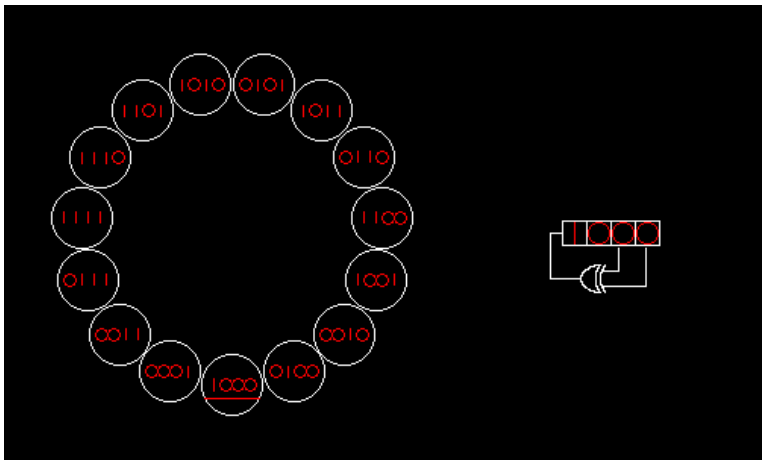


Image source: wikipedia

LFSR TX

8 bit LFSR with taps at 3,4,5,7 (counting from 0). Known to be maximal.

```
for (k=0;k<=3;k++) { // k indexes the 4 LFSRs
    low=0;
    if(lfsr[k]&8) // tap at bit 3
      low++; // each addition performs XOR on low bit of low
    if(lfsr[k]&16) // tap at bit 4
      low++;
    if(lfsr[k]&32) // tap at bit 5
      low++;
    if(lfsr[k]&128) // tap at bit 7
      low++;
    low&=1; // keep only the low bit
    lfsr[k]<<=1; // shift register up to make room for new bit
    lfsr[k]&=255; // we only want to use 8 bits (or make sure lfsr is 8 bit var)
    lfsr[k]|=low; // OR new bit in
}
OUTPUT_BIT(TX0,lfsr[0]&1); // Transmit according to LFSR states
OUTPUT_BIT(TX1,lfsr[1]&1);
OUTPUT_BIT(TX2,lfsr[2]&1);
OUTPUT_BIT(TX3,lfsr[3]&1);
```

LFSR demodulation

```
meas=READ_ADC(); // get sample…same sample will be processed in different ways
for(k=0;k<=3;k++) {
    if(lfsr[k]&1) // check LFSR state
      accum[k]+=meas;  // make sure accum is a 16 bit variable!
    else
      accum[k]-=meas;
}
```

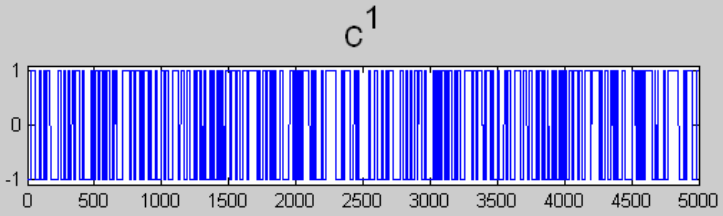
LFSR state sequence

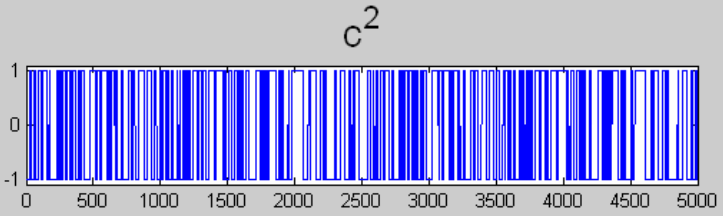
```
>> lfsr1(1:255)

ans =
2     4     8    17    35    71   142    28    56   113   226   196   137    18
37    75   151    46    92   184   112   224   192   129     3     6    12    25
50   100   201   146    36    73   147    38    77   155    55   110   220   185
114   228   200   144    32    65   130     5    10    21    43    86   173    91
182   109   218   181   107   214   172    89   178   101   203   150    44    88
176    97   195   135    15    31    62   125   251   246   237   219   183   111
222   189   122   245   235   215   174    93   186   116   232   209   162    68
136    16    33    67   134    13    27    54   108   216   177    99   199   143
30    60   121   243   231   206   156    57   115   230   204   152    49    98
197   139    22    45    90   180   105   210   164    72   145    34    69   138
20    41    82   165    74   149    42    84   169    83   167    78   157    59
119   238   221   187   118   236   217   179   103   207   158    61   123   247
239   223   191   126   253   250   244   233   211   166    76   153    51   102
205   154    53   106   212   168    81   163    70   140    24    48    96   193
131     7    14    29    58   117   234   213   170    85   171    87   175    95
190   124   249   242   229   202   148    40    80   161    66   132     9    19
39    79   159    63   127   255   254   252   248   240   225   194   133    11
23    47    94   188   120   241   227   198   141    26    52   104   208   160
64   128     1
```

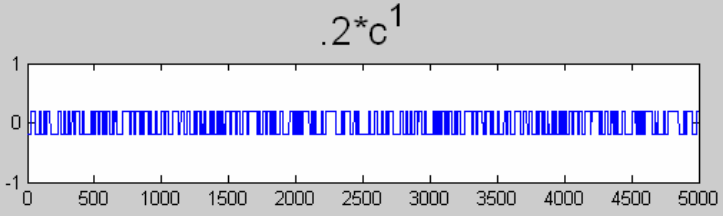
LFSR output

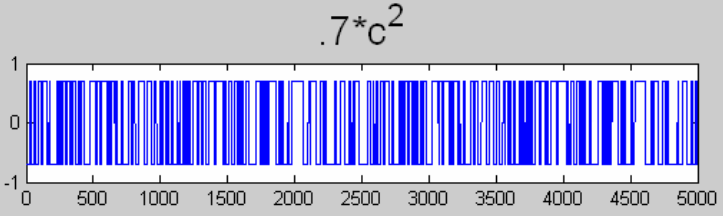
```
>> c1(1:255)  (EVEN LFSR STATE ➔ -1, ODD LFSR STATE➔ +1)

ans =
    -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    -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    -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    -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
```

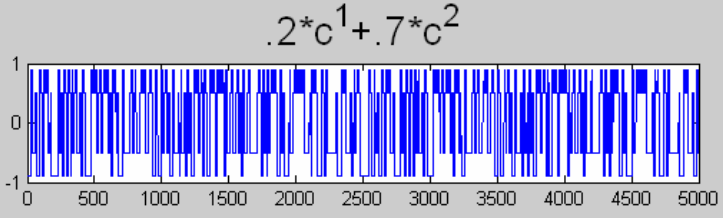
CDMA by LFSR












```
>> n1 = sum(c1.*c1)
n1 =    5000

>> n2 = sum(c2.*c2)
n2 =    5000

>> n12 = sum(c1.*c2)
n12 =    -60

>> rcv = .2 *c1 + .7*c2;
>> sum(c1/n1 .* rcv)
ans =    0.1916

>> sum(c2/n2 .* rcv)
ans =    0.6976
```

Note: CDMA carriers here consist of 500 pseudorandom values repeated 10 times each for better display

How to estimate X,Y,Z from sensor values

- X: (E-W)/(E+W)
- Y: (N-S)/(N+S)
- Z: N+E+W+S

Where N,E,W,S are the North, East, West, and South sensor values

- Threshold X,Y,Z…for some large values, behavior can get crazy

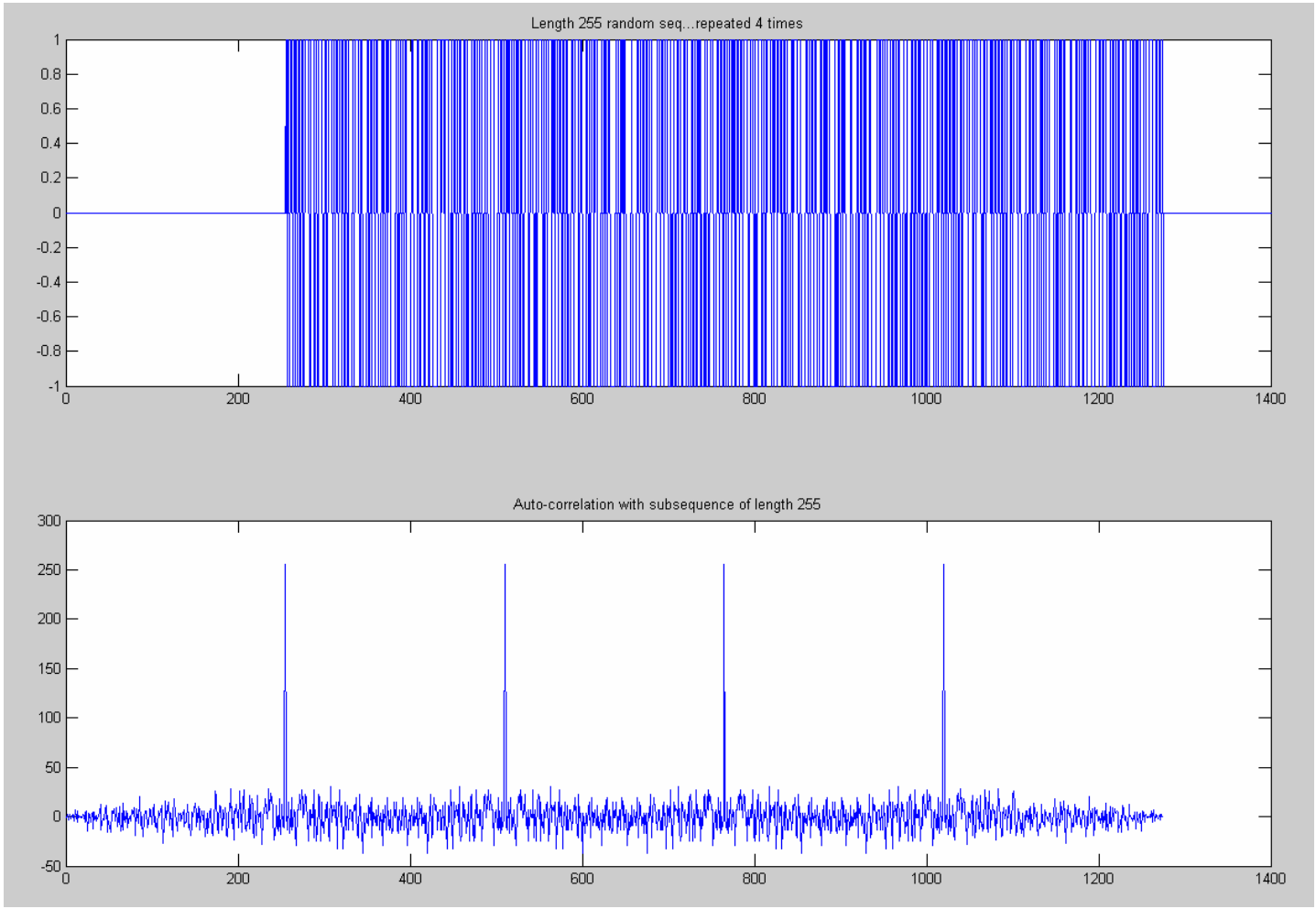
Evaluating sensor performance

- For sensors based on attenuation, performance given by
  - Contrast to noise ratio (not signal to noise ratio)
  - Contrast = max-min
  - CNR: (max-min) / std(constant signal value)

Break

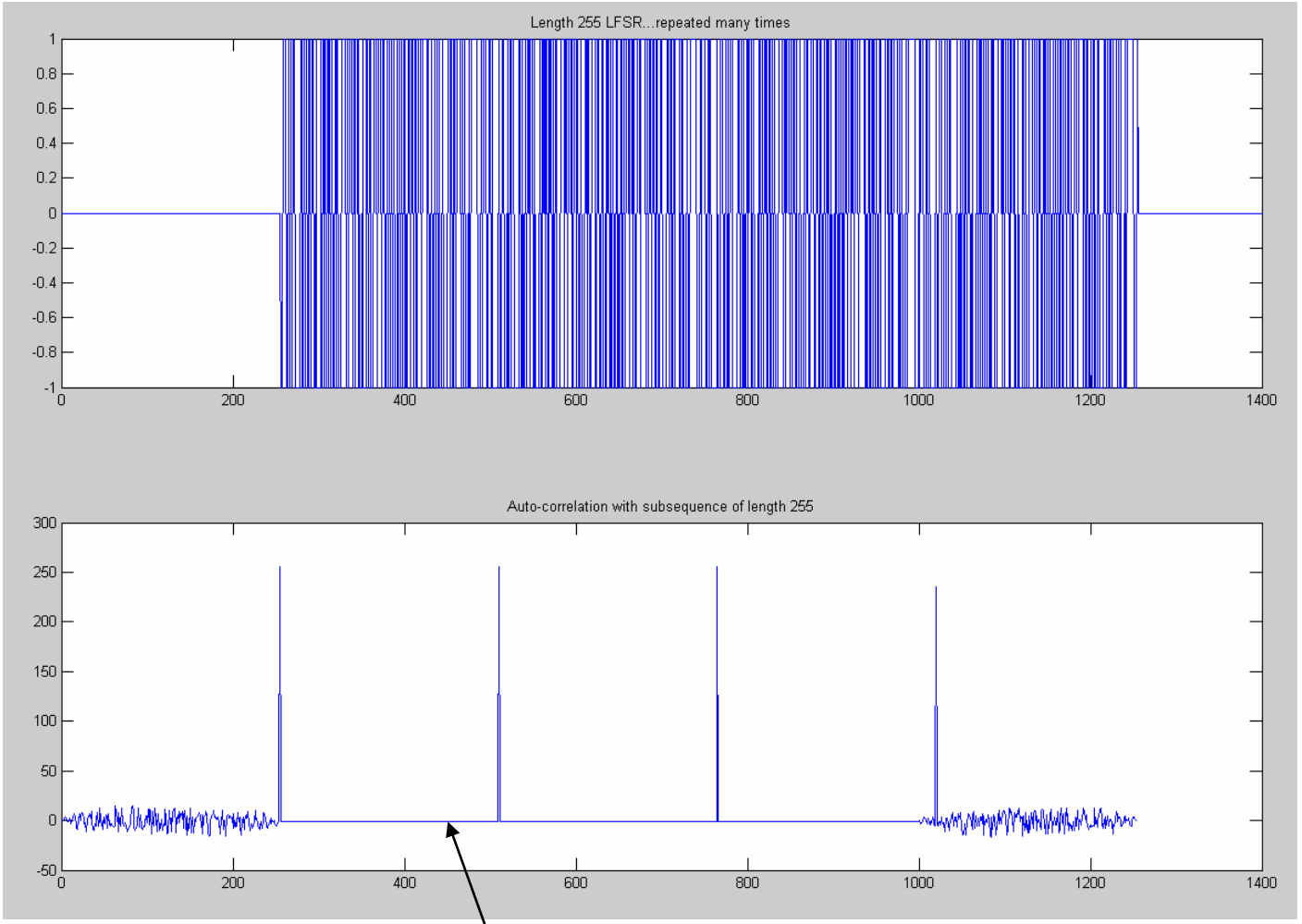
- That covers everything needed for the lab
- Remainder of these slides is extra background & further detail

Autocorrelation of pseudo-random (non-LFSR) sequence of length 255

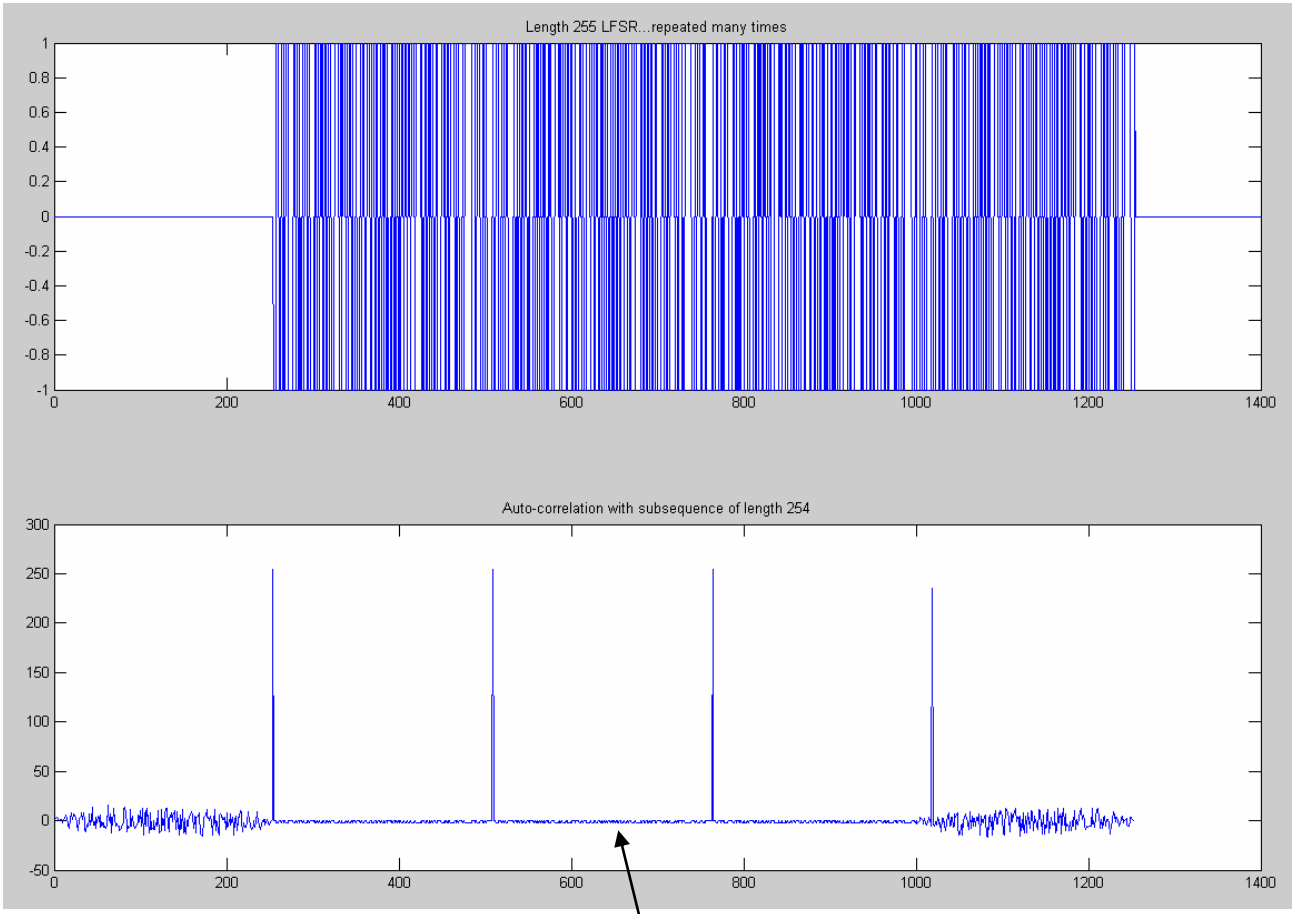


PR seq Generated w/ Matlab rand cmd

Autocorrelation (full length 255 seq)

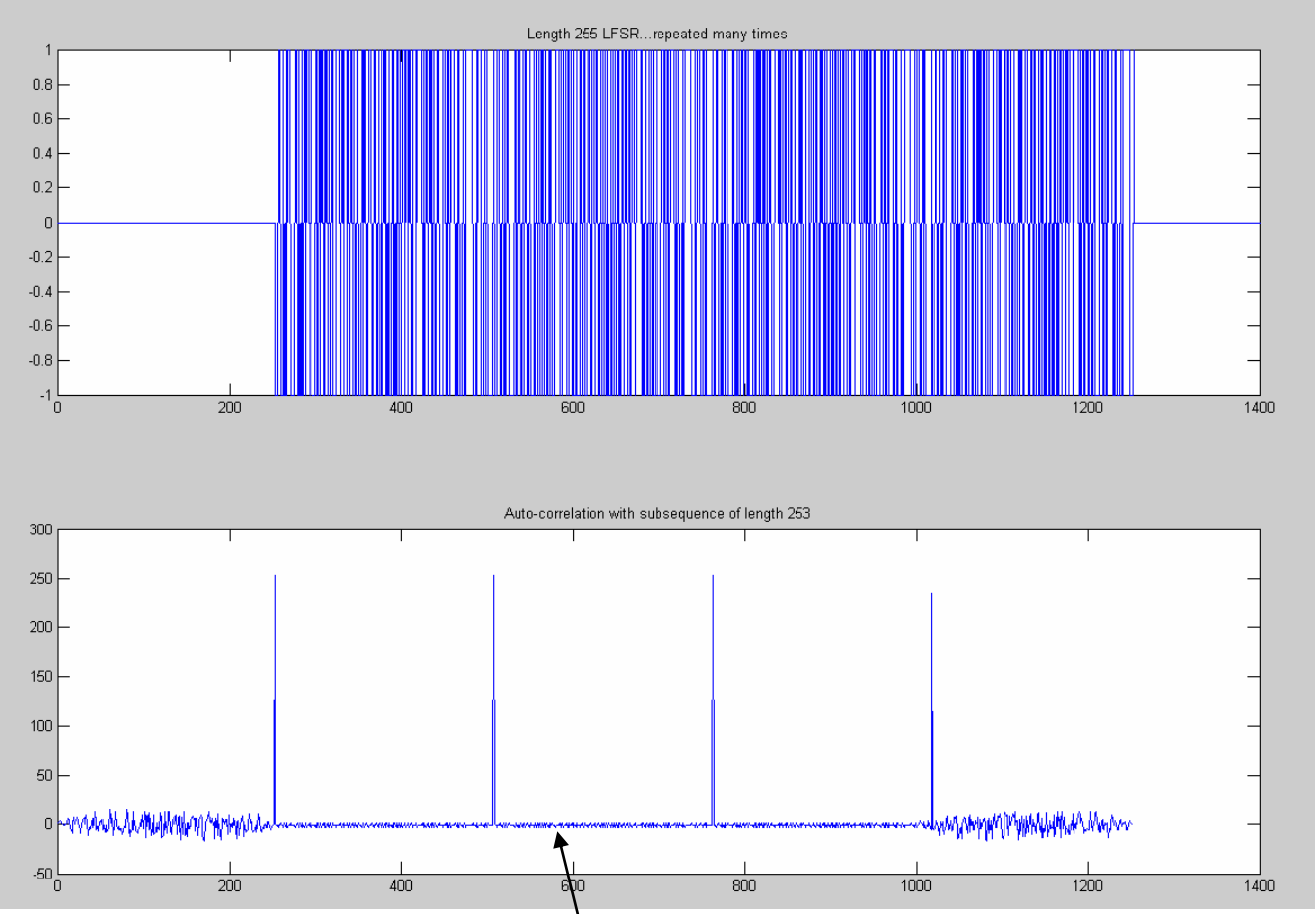


Autocorrelation (length 254 sub-seq)



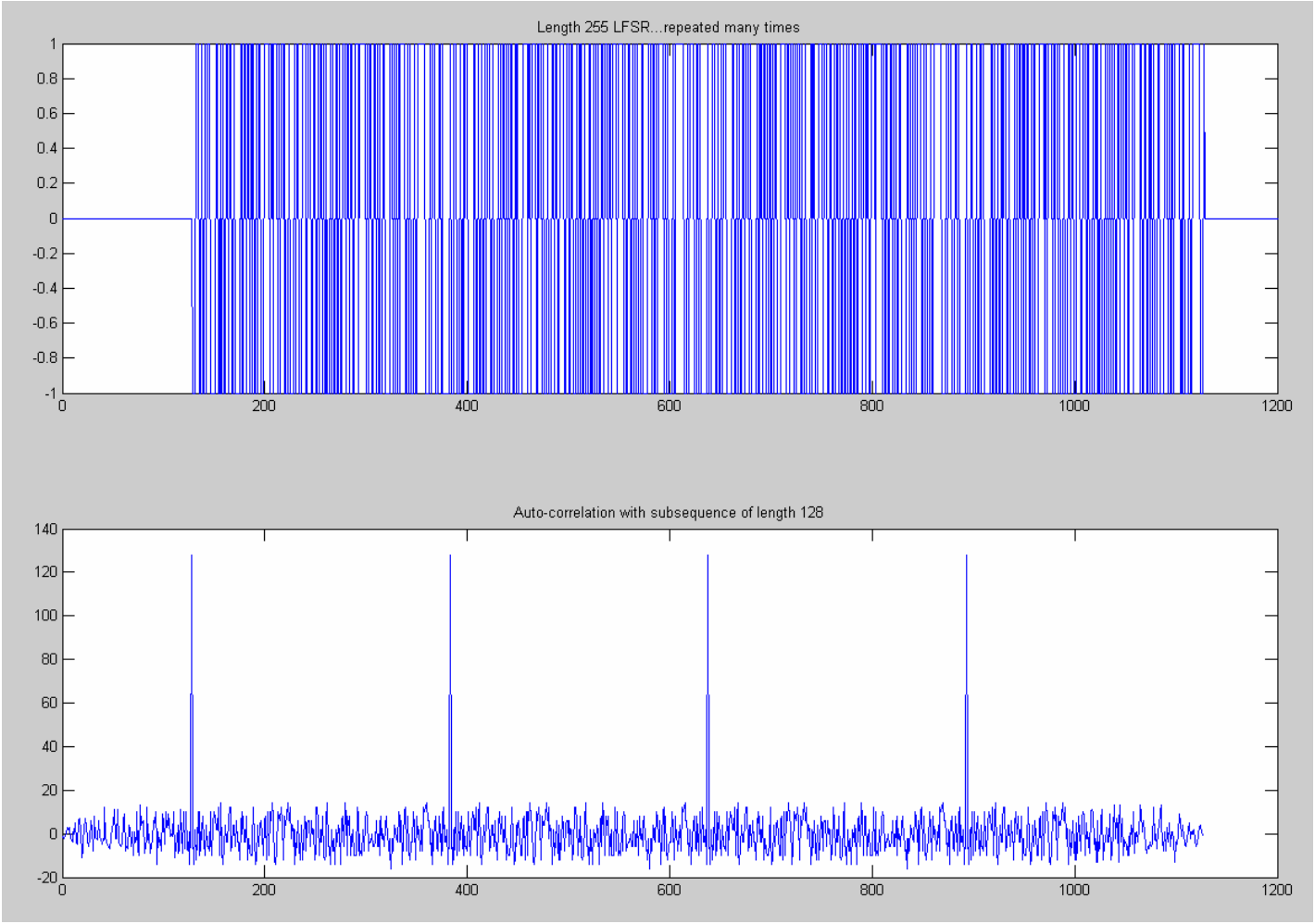
0 or -2

Autocorrelation (length 253 sub-seq)



1,-1, or -3

Autocorrelation (length 128 sub-seq)

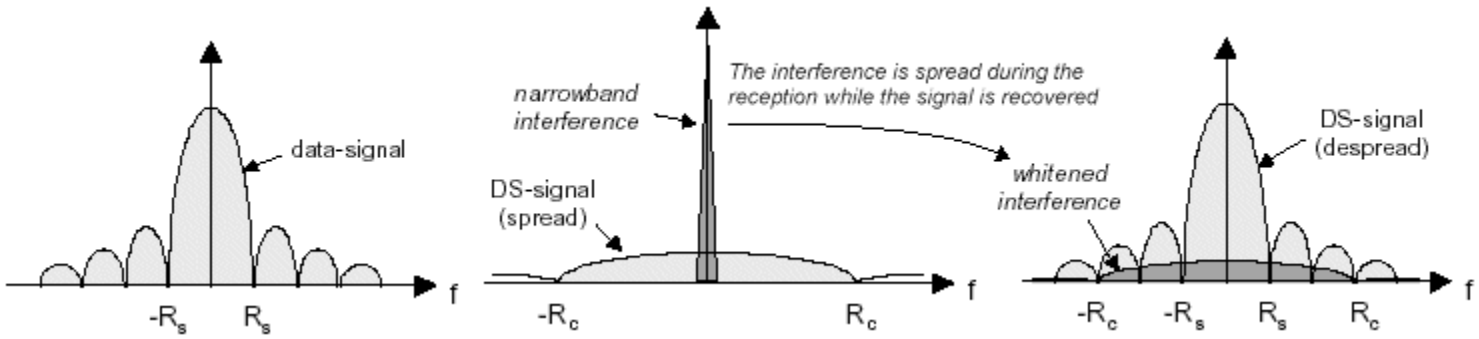


More on CDMA & LFSRs

- Other places where DSSS is used
  - 802.11b, GPS

- Terminology
  - Symbols: data
  - Chips: single carrier value
  - Varying number of chips per symbol varies data rate…when SNR is lower, increase number of chips per symbol to improve robustness and decrease data rate
  - Interference: one channel impacting another
  - Noise (from outside)

Visualizing DSSS



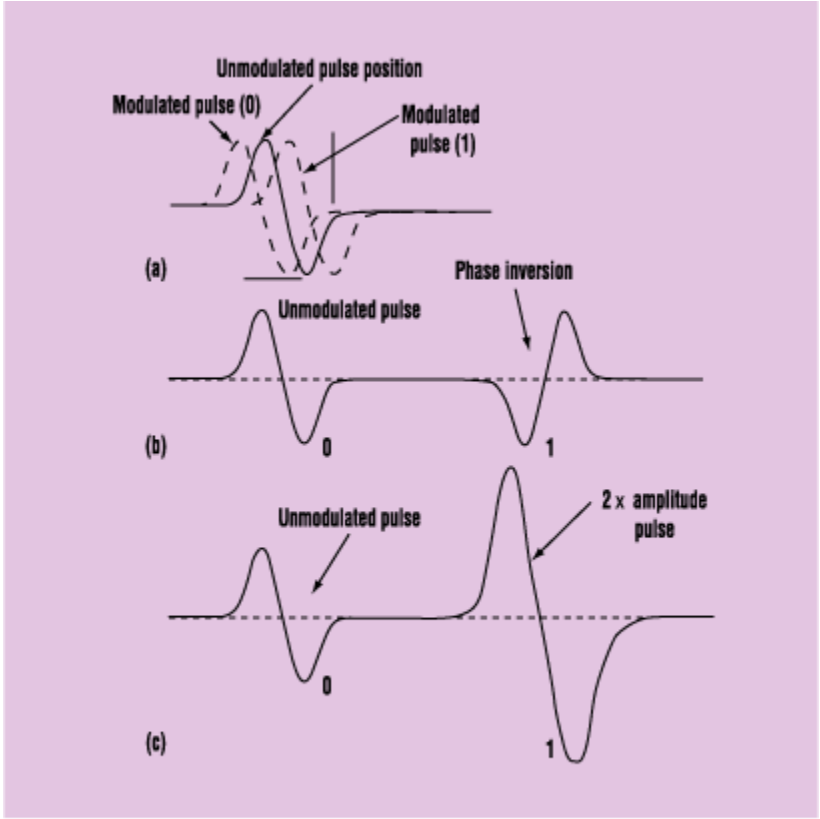
https://www.okob.net/texts/mydocuments/80211physlayer/images/dsss_interf.gif

Practical DSSS radios

- DSSS radio communication systems in practice use the pseudo-random code to modulate a sinusoidal carrier (say 2.4GHz)
- This spreads the energy somewhat around the original carrier, but doesn't distribute it uniformly over all bands, 0-2.4GHz
- Amount of spreading is determined by chip time (smallest time interval)

Ultrawideband

- Uses very short pulses
  - Pulse-position modulation
  - Binary phase shift
  - Pulse amplitdue modulation
- Effectively occupies all bandwidth up to 1/pulse time…much more than DSSS



3. In PPM, the pulses occur earlier (binary 0) or later (binary 1) than the pulse train without modulation (a). With biphase mod-ulation, a form of binary shift keying, the phase of the trans-mitted pulse is changed 180° to transmit a 1 (b). PAM changes the amplitude of the pulse to distinguish between a 0 and a 1.

http://electronicdesign.com/Files/29/1860/Figure_03.gif

Other

- Walsh-Hadamard Transform
  - A generalized Fourier Transform w/ binary values
  - Instead of using LFSR codes, can use Walsh functions
  - Has fast algorithm, so demodulating n channels can be done in time n log n instead of n^2
- Other uses of spread spectrum modulation
  - Digital watermarking (images, audio, video)