Department of
Computer Science and Engineering

Title: Implementing Encoding and Decoding Scheme Using Manchester

Data Communication Lab

CSE 308

Green University of Bangladesh

1 Objective(s)

- To be familiar with Manchester Line coding technique.

2 Problem analysis

In telecommunication and data storage, Manchester code is a line code in which the encoding of each data bit is either low then high, or high then low, for equal time. Normally when the bit is 1 it is start from high and then goes to low, When the bit is 0 it is start from low and then goes to 1. Although the opposite coding is also Manchester coding.
Here is an example of Manchester line coding for bits 10111001 shown in Figure 1.

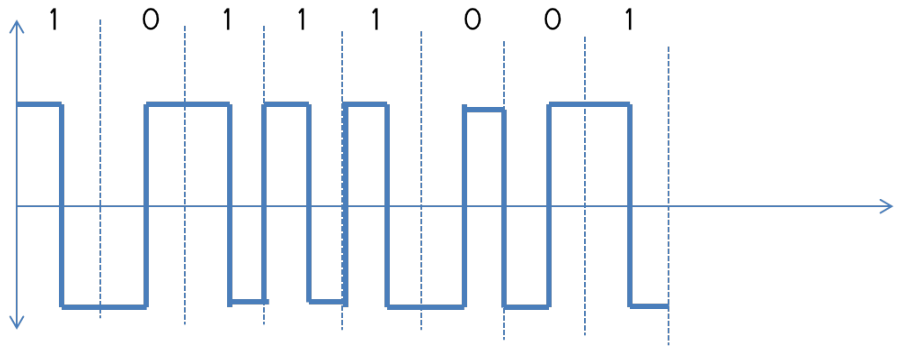

Figure 1: An example of Manchester line coding for bits 10111001

- **Conversion from bitstream to signal is called encoding.**
- **Conversion from signal to bitstream is called decoding.**

3 Coding With MatLab

For encoding we need the bitstream first. In MatLAB that can be defined as array or we can take input from user. Both ways are given below:

```
bits = [1 0 1 1 1 0 0 1];
```

or,

```
bits = input('prompt');
```

With bitstream we need bit rate that means how much bit(s) will appear in each second. This can be also defined as a variable or can be taken as input. Normally bit rate is considered as 1 bit per second. So we will define it to 1.

```
bitrate = 1;
```

We need a sampling for each bit. So that MatLAB needs to know how much sample point will be appeared for each bit. Low sampling can loss data or cannot define bits accurately. Very high sampling can cause slow your program. Here we will define it as 1000.

```
n = 1000;
```

MatLAB needs to know the total time needed for the bitstream. Sometimes total time T may be given or said to take as input. If not then we need to find it by dividing length of bitstream with bitrate.

```
T = length(bits)/bitrate;
```

We have a sampling for each bit we have declared previously. Now we need to find total sample point needed for whole bitstream. We can find it by multiplying sample point of each bit with the number of bits as below.

```
N = n*length(bits);
```

Now, we need to found time for each sample point. For this we divide the whole time (T) with number of total sample points (N).

```
dt = T/N;
```

We are near to the main process. Now we will define the time domain. That is the all values of X-axis. For this we need to declare an array from time 0 to time T by partitioning with dt. That is an array like this.

```
t = 0:dt:T;
```

For better performance we need to set all values of the result (amplitue domain) to 0. We can do this task with a build in function zeros().

```
x = zeros(1,length(t));
```

Now for each bit if that bit is 1 then a range of amplitude domain will be filled as first half of the time unit is 1 and the second half of the time unit is -1. If the bit is 0 then a range of amplitude domain will be filled as the first half of the time unit is -1 and the second half of the time unit is 1. That is in each bit signal cross zero at midpoint but not return to zero.

```
for i=1:length(bits)
  if bits(i)==1
    x((i-1)*n+1:(i-1)*n+n/2) = 1;
    x((i-1)*n+n/2:i*n) = -1;
  else
    x((i-1)*n+1:(i-1)*n+n/2) = -1;
    x((i-1)*n+n/2:i*n) = 1;
  end
end
```

Everything required for encoding is done. We will now plot the result (amplitude domain) on respect to time domain. Here is the plot. Here 'Linewidth', 3 means the signal line will be bold three times than original signal.

```
plot(t, x, 'Linewidth', 3);
```

4 Implementation in MatLAB

4.1 Code For Encoding

```
bits = input('prompt');
bitrate = 1;
n = 1000;
T = length(bits)/bitrate;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1,length(t));
for i=1:length(bits)
  if bits(i)==1
    x((i-1)*n+1:(i-1)*n+n/2) = 1;
    x((i-1)*n+n/2:i*n) = -1;
  else
    x((i-1)*n+1:(i-1)*n+n/2) = -1;
    x((i-1)*n+n/2:i*n) = 1;
  end
end
plot(t, x, 'Linewidth', 3);
```

4.2 Decoding

For decoding we need a counter that will count how much bit has been decoded. We will define the initial value of the counter to 0.

```
counter = 0;
```

Now we need a loop again to decode the signal. The loop will run from 1 to length of t. That is same times of total sample point. In the loop we will test whether t(i) is greater than the counter. If it is true then we will increase the value of counter. Then if x(i) is positive then we will assign x(i) to result(counter) and if x(i) is negative we will assign 0 to result(counter). In this way we will assign or test the second sample value of each bit to the result bitstream. And for other samples the if condition (external if) will be false. For example, At first the value of t(1)=0 and that is not greater than counter so nothing occurred. Then t(2)=0.001 and that is greater than 0. So counter increased to 1 and the bit is set to the value of 1 or 0 (depending on voltage level). Next values of t(i) will be skipped (if condition will be false) as long as it is not 1.001. For each bit same things will be occurred. The loop looks like below.

```
for i = 1:length(t)
  if t(i)>counter
    counter = counter + 1;
    if x(i)>0
      result(counter) = x(i);
    else result(counter) = 0;
    end
  end
end
```

4.3 Display

After this decoding is also done. We need to display it. We can display that by just writing the variable without semi-colon. Or this can shown with disp() function.

```
disp('Manchester Decoding:');
disp(result);
```

4.4 Full Code (Encoding and decoding)

```
bits = input('prompt');
bitrate = 1;
n = 1000;
T = length(bits)/bitrate;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1,length(t));
for i=1:length(bits)
  if bits(i)==1
    x((i-1)*n+1:(i-1)*n+n/2) = 1;
    x((i-1)*n+n/2:i*n) = -1;
  else
    x((i-1)*n+1:(i-1)*n+n/2) = -1;
    x((i-1)*n+n/2:i*n) = 1;
  end
end
plot(t, x, 'Linewidth', 3);
counter = 0;
for i = 1:length(t)
  if t(i)>counter
    counter = counter + 1;
```

```
    if x(i)>0
      result(counter) = x(i);
    else result(counter) = 0;
    end
  end
end
disp('Manchester Decoding:');
disp(result);
```

5 Sample Input/Output (Compilation, Debugging & Testing)

input: 1 0 1 1 1 0 0 1
Output:

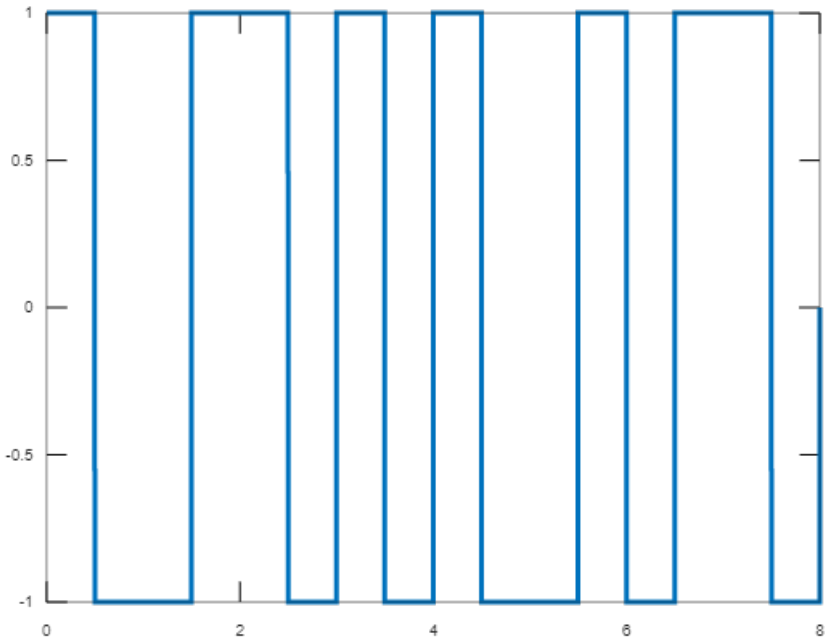

Figure 2: Output

6 Discussion & Conclusion

Based on the focused objective(s) to understand about the algorithms, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Implementing Encoding and Decoding Scheme Using Differential Manchester

7.1 Problem analysis

Differential Manchester is somewhat a combination of the RZ and NRZ-I line coding schemes. There is always a transition at the middle of the bit but the bit values are determined at the beginning of the bit. If the bit is

0 then there is a transition, if the bit is 1 then there is no transition. Although the transition can be from 1 to 0 and 0 to 1 it is usually used from 1 to -1 and -1 to 1.

Here is an example of Differential Manchester line coding for bits 10111001 shown in Figure 3.

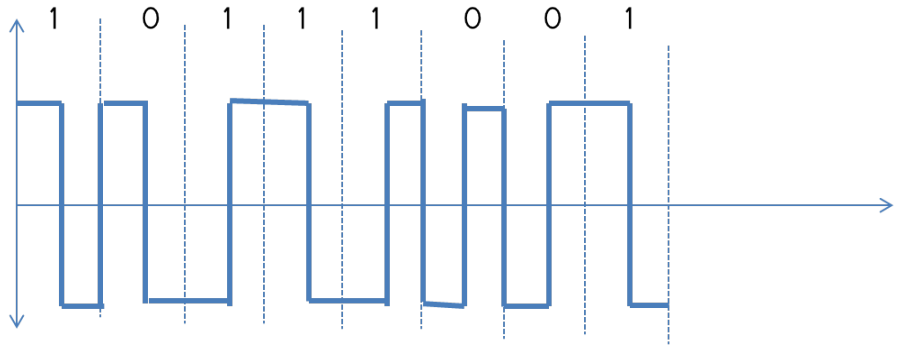

Figure 3: An example of Differential Manchester line coding for bits 10111001

8 Lab Exercise (Submit as a report)

- Implementing Encoding and Decoding Scheme Using Differential Manchester.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.