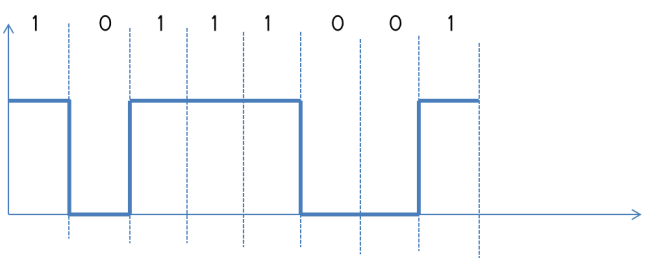
**Unipolar Line Coding With MatLAB Source Code For Encoding and Decoding**

**Basic**

Unipolar encoding is a line code. A positive voltage represents a binary 1, and zero volts indicates a binary 0. It is the simplest line code, directly encoding the bitstream.

Here is an example of Unipolar line coding for bits 10111001.



Conversion from bitstream to signal is called encoding.

Conversion from signal to bitstream is called decoding.

**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:

|  |  |
| --- | --- |
| 1 | bits = [1 0 1 1 1 0 0 1]; |

or,

|  |  |
| --- | --- |
| 1 | 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.

|  |  |
| --- | --- |
| 2 | 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.

|  |  |
| --- | --- |
| 3 | 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.

|  |  |
| --- | --- |
| 4 | 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.

|  |  |
| --- | --- |
| 5 | 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).

|  |  |
| --- | --- |
| 6 | 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.

|  |  |
| --- | --- |
| 7 | 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().

|  |  |
| --- | --- |
| 8 | x = zeros(1,length(t)); |

Now for each bit if that bit is one than a range of the amplitude domain is redefined with value 1. If not defined anything else. Here we will continue a loop from 1 to length of bitstream. Inside the loop we will check whether the bit is 1 or 0. If the bit is 0 then no change will be needed. But if the bit is 1 the we will fill range from x((i-1)\**n+1*) to x(i\*n). That is if i=1 then we will fill from 1 to n’th (sample points per bit) position of the array with 1. If i=2 then we will fill from (n+1) to 2\*n’th position. Here is the loop required.

|  |  |
| --- | --- |
| 9  10  11  12  13 | for i=1:length(bits)    if bits(i)==1      x((i-1)\*n+1: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.

|  |  |
| --- | --- |
| 14 | plot(t, x, 'Linewidth', 3); |

**Code For Encoding**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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\*n) = 1;    end  end  plot(t, x, 'Linewidth', 3); |

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.

|  |  |
| --- | --- |
| 15 | 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 than we will increase the value of counter. And will assign the value of x(i) to result(counter) array. In this way we will assign the second sample value of each bit to the result bitstream. And for other samples the if condition will be false. For example, At first the value of t(1)=0 and that is not greater than counter so nothing occured. 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 x(2). 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 held. The loop looks like below.

|  |  |
| --- | --- |
| 16  17  18  19  20  21 | for i = 1:length(t)    if t(i)>counter      counter = counter + 1;      result(counter) = x(i);    end  end |

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.

|  |  |
| --- | --- |
| 22  23 | disp('NRZ Unipoler Decoding:');  disp(result); |

**Full Code (Encoding and decoding)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | 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\*n) = 1;    endif  endfor  plot(t, x, 'Linewidth', 3);  counter = 0;  for i = 1:length(t)    if t(i)>counter      counter = counter + 1;      result(counter) = x(i);    endif  endfor  disp('NRZ Unipoler Decoding:');  disp(result); |

I think, your experience with this tutorial is quite good. You can notify me anything related to this tutorial via comment or contact form.

Thanks.