

Encoding Techniques with MATLAB

Create two MATLAB programs, one main and one subprogram.

Submit your report in a Microsoft Word document.

Create two MATLAB programs, one main and one subprogram. The codes for both programs are below.

line_encoding.m (Main Program)

%EE 640

%Line Encoding Demo

%Input Data bit strings

input_data_0 = '101011100';

input_data_1 = '00101010011100100';

input_data_2 = '0101010111000';

%Generates graph of line encoding of input data

generate_line_encoding_graph(input_data_0);

generate_line_encoding_graph(input_data_1);

generate_line_encoding_graph(input_data_2);

generate_line_encoding_graph.m (Subprogram)

Generates a plot of six different line encoding techniques for a given

%binary input

%input_data – The binary data in either an int array or char array

%Example usage:

```

%line_encoding_demo([1 0 1 0 1 1 1 0 0]);

%line_encoding_demo('101011100');

function generate_line_encoding_graph(input_data)

%Determine which format the input data is in and convert it if
%necessary.

if(ischar(input_data))

data = input_data - '0';

else

data = input_data;

end

%Verify if input data is valid

if(~(min(data) == 0 || min(data) == 1) || ~(max(data) == 0 || max(data) == 1))

fprintf('\n\nIncorrect input data\n');

fprintf('Usage:\n');

fprintf('\tline_encoding_demo()\n');

fprintf('Example Usage:\n');

fprintf('\tline_encoding_demo([1 0 1 0 1 1 1 0 0])\n');

fprintf('\tline_encoding_demo("101011100")\n');

return;

end

%Calculate the number of data points used for plotting purposes

```

```

num_steps_per_bit = 1000;      %The number of points per bit
step_size =
1/num_steps_per_bit;          %The size between points
x=0:step_size:length(data);    %The x vector for plotting
x=(x(1,1:length(x)-1));       %Cleanup the x vector
y=zeros(6,length(x));         %The output vector for each
                                encoding

```

%The labels used for each of the plots

```

labels= [{'Unipolar','NRZ'};{'Polar','NRZ'};{'NRZ','Inverted'};
{'Bipolar','Encoding'};{'Manchester','Encoding'};{'Differential','Manchester'}];

```

%The current direction (inflection of NRZ_inverted)

```
nrz_inverted_direction = -1;
```

%The current direction of bipolar

```
bipolar_direction = -1;
```

%The current direction of differential manchester

```
differential_manchester_direction = -1;
```

```

%Generates the output data for plotting of the six line encoding
%techniques for each of the bits in the input_data

```

```
for i = 1:length(data)
```

```
    %Unipolar_NRZ
```

```
    %+V for 1 and 0 for 0
```

```
    index = (i-1)*num_steps_per_bit+1;
```

```
    y(1,index:(index+num_steps_per_bit-1)) = data(i);
```

```
    %Polar_NRZ
```

%+V for 1 and -V for 0

if(data(i) == 1)

y(2,index:(index+num_steps_per_bit-1)) = 1;

else

y(2,index:(index+num_steps_per_bit-1)) = -1;

end

%NRZ_Inverted

%Signal changes between +V and -V for every bit with a value of 1

if(data(i) == 1)

nrz_inverted_direction = -nrz_inverted_direction;

end

y(3,index:(index+num_steps_per_bit-1)) = nrz_inverted_direction;

%Bipolar Encoding

%0 for 0 and for 1 the signal alternates between +V and -V for

%every bit with a value of 1

if(data(i) == 1)

bipolar_direction = -bipolar_direction;

end

index = (i-1)*num_steps_per_bit+1;

y(4,index:(index+num_steps_per_bit-1)) = bipolar_direction*data(i);

%Manchester Encoding

%Signal transitions in the middle of the bit duration

%-V to +V for 1

%+V to -V for 0

if(data(i) == 1)

manchester_direction = -1;

else

manchester_direction = 1;

end

y(5,index:(index+num_steps_per_bit/2-1)) = manchester_direction;

y(5,(index+num_steps_per_bit/2):(index+num_steps_per_bit-1)) = -
manchester_direction;

%Differential Manchester

%Signal transitions in the middle of the bit duration

%The signal remains the same at the bit transition point for 1

%The signal switches sign at the bit transition point if the bit is 0

differential_manchester_direction = -differential_manchester_direction;

if(data(i) == 0)

differential_manchester_direction = -differential_manchester_direction;

end

y(6,index:(index+num_steps_per_bit/2-1)) = differential_manchester_direction;

y(6,(index+num_steps_per_bit/2):(index+num_steps_per_bit-1)) = -
differential_manchester_direction;

end

%plots the six line encoding techniques

figure;

for i = 1:6

%Switch to the subplot for a given line encoding techniques

subplot(6,1,i);

%Plot the x axis and adjust its line width

graph = line([0 length(data)],[0 0]);

set(graph,'LineWidth',2);

hold on;

%Plot the signal representing the line encoding

graph = plot(x,y(i,:), 'k');

grid on;

%Set the correct properties of the plot

axis([0 length(data) -1.5 1.5]);

set(gca,'ytick',[-1 0 1]);

set(gca,'xtick',0:1:length(data));

set(graph,'LineWidth',2);

ylabel(labels(i,:));

set(get(gca,'YLabel'),'Rotation',0)

set(get(gca,'YLabel'),'Units','Normalized','Position',[-0.1, 0.25, 0]);

%Add the bits to the top of the plot

for j=0:(length(data)-1)

text(j+0.5,1.25,num2str(data(j+1)));

end

end

end

Directions

1. Run the main program for the input data “01110010110” and analyze the figures.
2. What happens if the signals are reversed?
3. In a table, compare NRZ, NRZI, Manchester, and Manchester Differential techniques in terms of efficiency and clock recovery.
4. Write a report with the output figures and a table that compares the four techniques. Draw a conclusion. Submit this report.