

IMPLEMENTATION OF 8/10 DECODER OF ETHERNET PHY USING PYTHON AND VERIFICATION USING MATLAB.

APRIL 10, 2022

FOLDER :

[https://drive.google.com/drive/folders/1otBIr0hru0VCBP826z43tjeTDj4B9lA9?
usp=sharing](https://drive.google.com/drive/folders/1otBIr0hru0VCBP826z43tjeTDj4B9lA9?usp=sharing)

1. ***NAME:*** Nilay Arya
ID NO: 2019AAPS1230H
2. ***NAME:*** Surya Chandra G
ID NO: 2019AAPS1221H
3. ***NAME:*** Pavan Prathapa S
ID NO: 2019AAPS1225H

8B/10B ENCODER-DECODER

ACKNOWLEDGEMENT

We would like to thank Dr. Subhendhu Sahoo Kumar for giving this project on 8/10 decoder. It was a great opportunity to learn about encoders and decoders and to get some hands-on experience in python and MATLAB. We would also like to thank our friends who helped us with the project. It was a collaborative effort, and we learnt a lot about teamwork and delegation. Overall, it was a great learning experience.

CONTENTS

- INTRODUCTION
- FEATURES
- APPLICATION
- BLOCK DIAGRAMS
- 8B/10B CODE MAPPING
- DISPARITY
- RUNNING DISPARITY
- 8B/10B ENCODING-DECODING TABLES
- CODE IN PYTHON FOR DECODER
- DECODER VERIFICATION USING MATLAB

INTRODUCTION

- 8b/10b encoding was proposed by Albert X. Widmer and Peter A. Franaszek of IBM Corporation in 1983
- 8B/10B Encoder encodes data for transmission where 8-Bit data converted to 10-Bit data.
- Single bit error detection is possible
- High-speed Serial data transmission protocols utilize 8B/10B Encoding scheme.

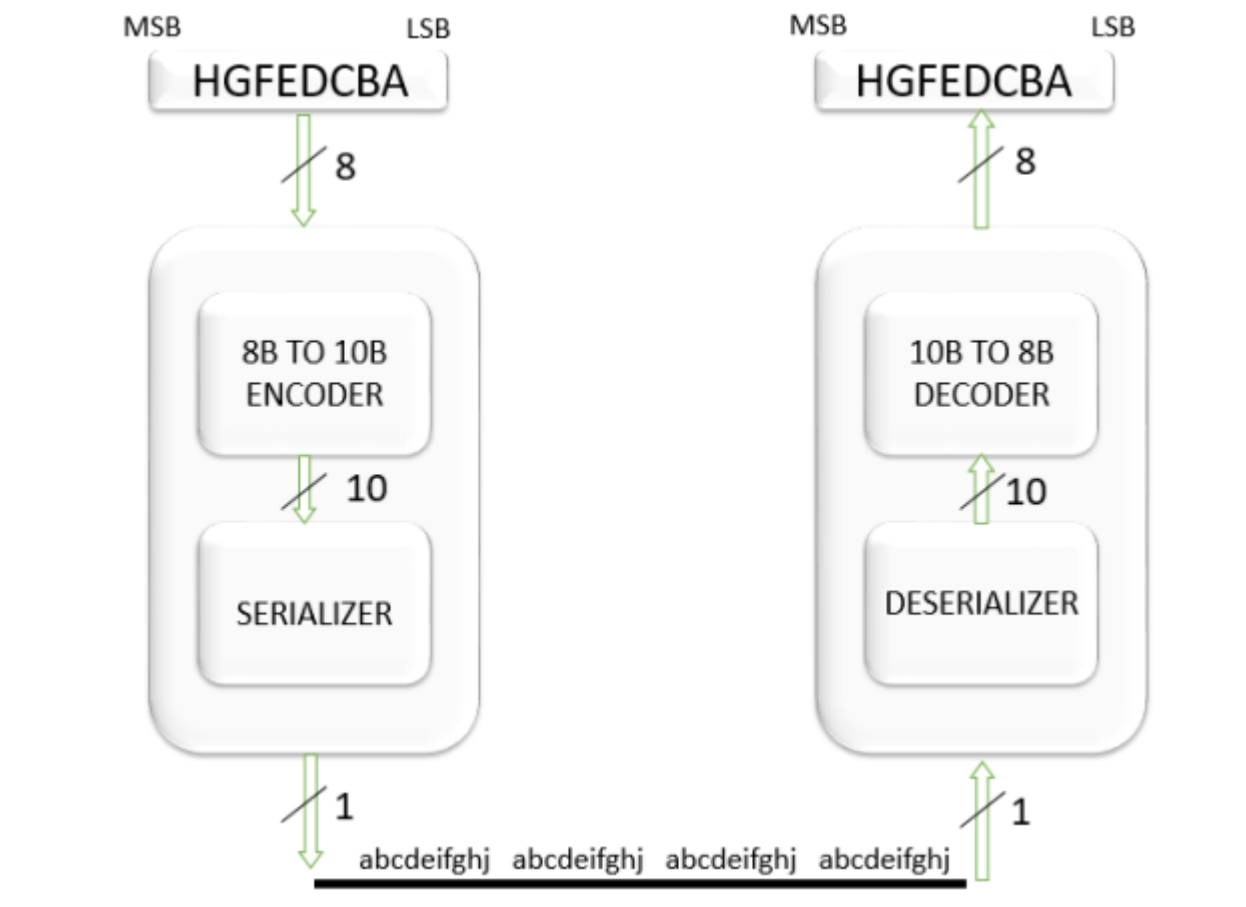
FEATURES

- Encoding of 8-bit bytes into 10-bit symbols
- Decoding of 10-Bit symbols into 8-bit Bytes

APPLICATIONS

- **Gigabit Ethernet**
- **PCI-Express 1.x and 2.x**
- **JESD 204B**
- **USB 3.0**
- **Display Port**
- **Serial ATA (SATA)**
- **XAUI**
- **SAS**

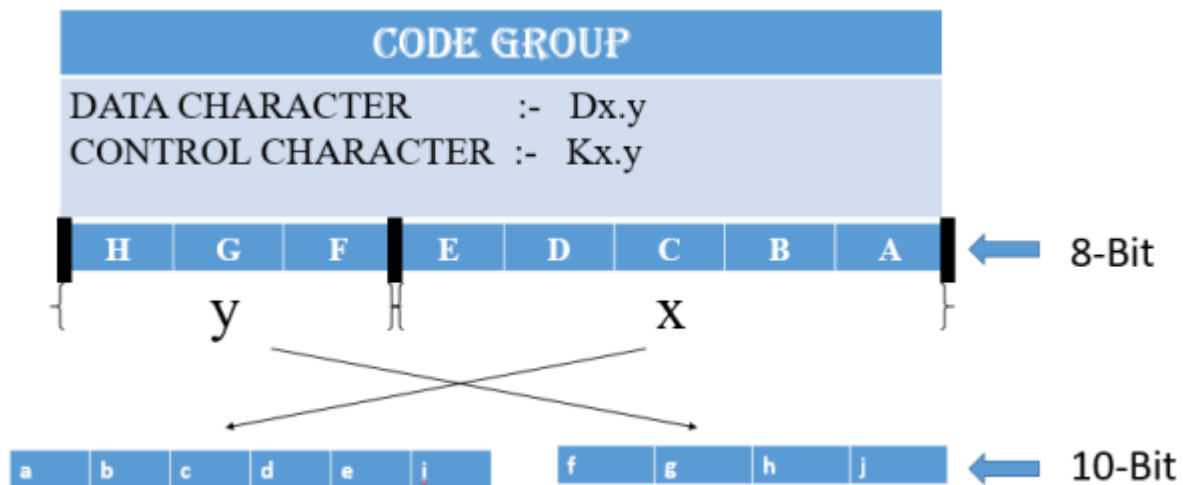
BLOCK-DIAGRAM



8B TO 10B CODE MAPPING

- Encodes 8-Bit Bytes to 10-Bit Symbols
- Code groups includes 256 data characters and 12 control characters
- Data character named as Dx.y :- 8-bit width
- Control character named as Kx.y :- 8-bit width
- This coding scheme breaks data in to two blocks
 - 3 MSB Bits
 - 5 LSB bits
- 3 Bit encoded to 4 Bits
- 5 Bit encoded to 6 Bits
- 4-Bit and 6-Bit blocks combined and encoded into 10-Bit symbol

For example



DISPARITY

- The disparity is required to be employed a balanced number of 0s and number of 1s
- The disparity of Block = number of 1s - the number of 0s
- The disparity of Block = 0 à "Disparity Neutral"
- The disparity of 4-Bit block and a 6-Bit block is Neutral then Disparity of 10-Bit block is neutral
- Disparity Neutral means perfect DC-balanced code stream.

For Example.....

DATA(0) or CONTROL(1)	8 BIT INPUT DATA	$D_{x.y} / K_{x.y}$
0	8'b_000_01110	D14.0
0	8'b_010_01100	D12.2
0	8'b_100_01001	D9.4
1	8'b_101_11100	K28.5
1	8'b_111_11011	K27.7
1	8'b_111_11110	K30.7

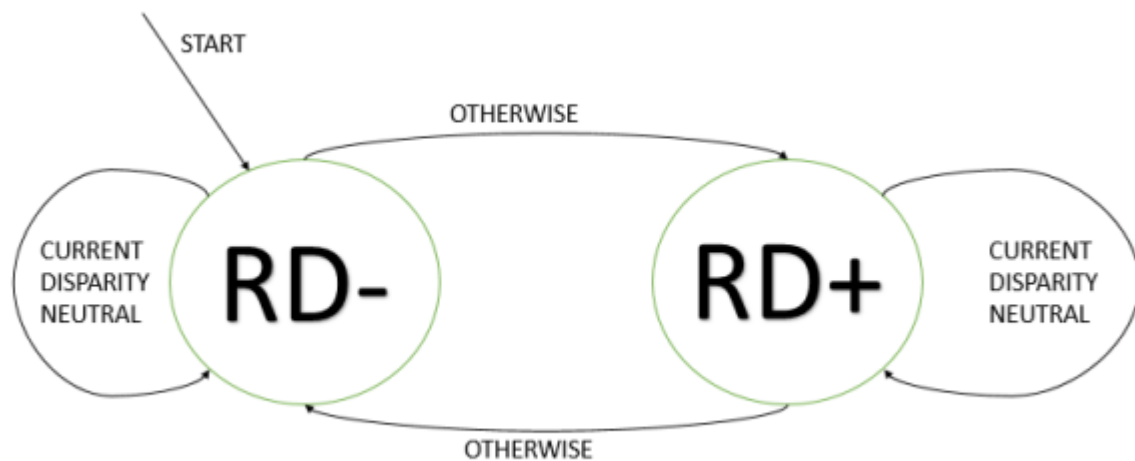
RUNNING DISPARITY

- Value of disparity of 10-Bit encoded data is +2 or -2 or 0.
- +1 or -1 is invalid running disparity of 10-Bit encoded data.
- The serial data stream is no longer DC-balanced if the disparity is +2 or -2
- Running disparity is introduced to transmit data stream DC-balanced
- In order to maintain a DC-balance data stream, each code group will be converted to one of the two possible values
 - RD+
 - RD-
- The RD- disparity will be +2 or 0
- The RD+ disparity will be -2 or 0

For example

3BIT to 4 BIT ENCODING		5 BIT to 6 BIT ENCODING	
TOTAL POSSIBLE COMBINATIONS OF 4 BIT	16	TOTAL POSSIBLE COMBINATIONS OF 6 BIT	64
TOTAL NUMBER OF COMBINATIONS WHICH CONTAINS NEUTRAL DISPARITY	6	TOTAL NUMBER OF COMBINATIONS WHICH CONTAINS NEUTRAL DISPARITY	20
TOTAL POSSIBLE COMBINATION OF 3 BIT DATA	8	TOTAL POSSIBLE COMBINATION OF 5 BIT DATA	32
So here 6 possible 4 bit combinations available for encoding 8 possible combinations of 3 bit values with neutral disparity which is not possible so disparity is not always 0.		So here 20 possible 6 bit combinations available for encoding 32 possible combinations of 5 bit values with neutral disparity which is not possible so disparity is not always 0.	

- Transmitter assumes negative disparity(RD-) at startup.
- The encoder will pick one of the two values based on the calculation of current Running Disparity
- Receiver side or decoder side input 10-Bit data is not valid if a 10-Bit value is not there in the encoding-decoding table.
- Running disparity state diagram is given below



- Every 8-Bit input data can be encoded to 10-Bit data by two possibilities first is positive disparity and second is a negative disparity.

For Example.....

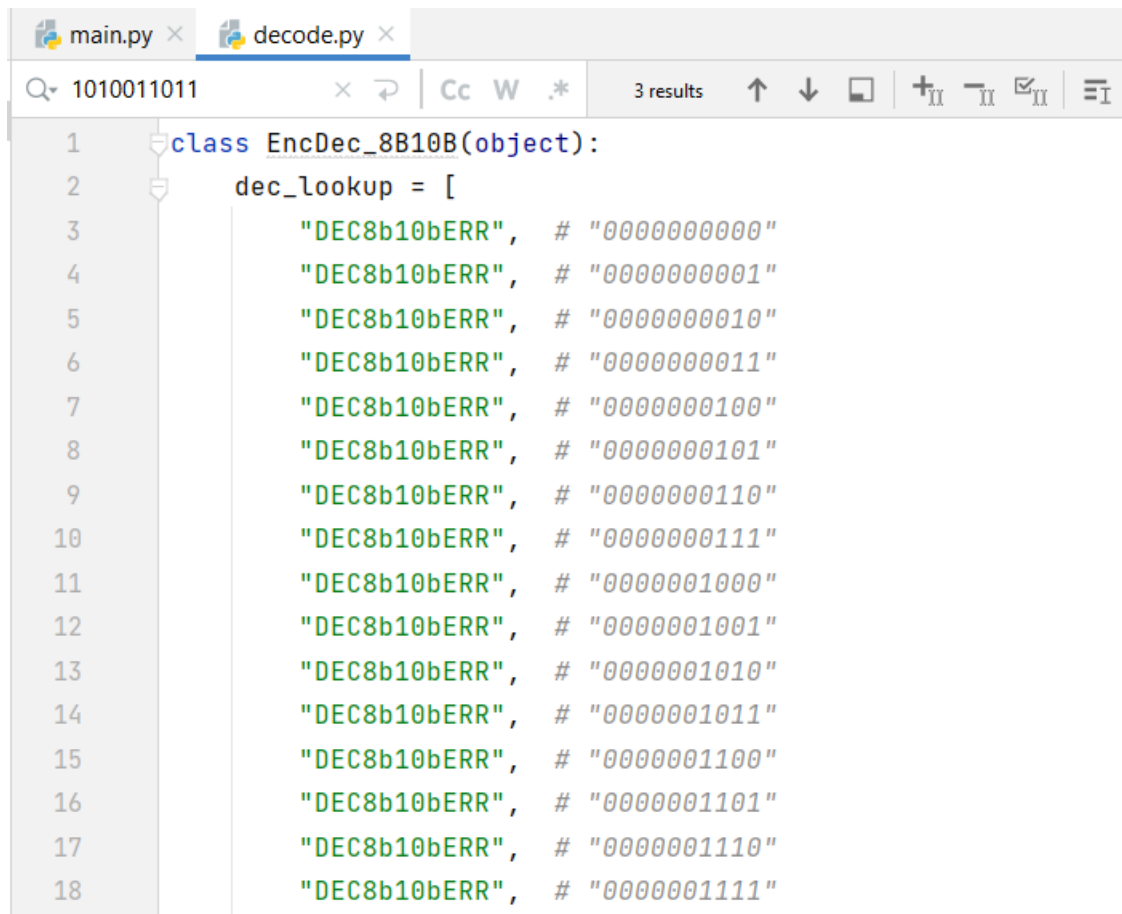
DATA(0) or CONTROL(1)	8 BIT INPUT DATA	D _{x,y} / K _{x,y}	10-BIT OUTPUT DATA (RD-)	10-BIT OUTPUT DATA (RD+)	DISPARITY (RD-)	DISPARITY (RD+)
0	8'b_000_01110	D14.0	10'b_011100_1011	10'b_011100_0100	2	-2
0	8'b_010_01100	D12.2	10'b_001101_0101	10'b_001101_0101	0	0
0	8'b_100_01001	D9.4	10'b_100101_1101	10'b_100101_0010	2	-2
1	8'b_101_11100	K28.5	10'b_001111_1010	10'b_110000_0101	2	-2
1	8'b_111_11011	K27.7	10'b_110110_1000	10'b_001001_0111	0	0
1	8'b_111_11110	K30.7	10'b_011110_1000	10'b_100001_0111	0	0

8B/10B ENCODING-DECODING TABLES

- Download Encoding-Decoding tables from following link.....

LINK :- [ENCODER AND DECODER TABLE](#)

PYTHON SIMULATION AND CODES:



The screenshot shows a Python IDE with two tabs: 'main.py' and 'decode.py'. A search bar at the top contains the text '1010011011'. Below the search bar, the results are displayed in a list. The first result is a class definition 'class EncDec_8B10B(object):' in 'decode.py'. The second result is a list 'dec_lookup = [' in 'decode.py'. The third result is a list of 16 elements, each being a string 'DEC8b10bERR' followed by a comment indicating a binary value. The list is as follows:

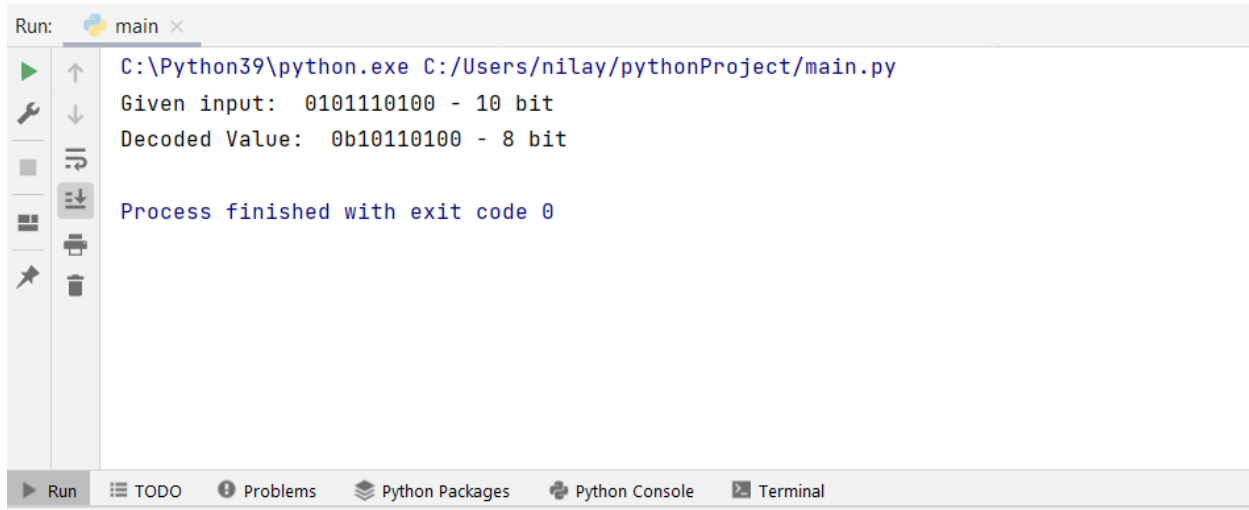
```
1 class EncDec_8B10B(object):
2     dec_lookup = [
3         "DEC8b10bERR", # "000000000000"
4         "DEC8b10bERR", # "000000000001"
5         "DEC8b10bERR", # "000000000010"
6         "DEC8b10bERR", # "000000000011"
7         "DEC8b10bERR", # "000000001000"
8         "DEC8b10bERR", # "000000001001"
9         "DEC8b10bERR", # "000000001010"
10        "DEC8b10bERR", # "000000001011"
11        "DEC8b10bERR", # "000000010000"
12        "DEC8b10bERR", # "000000010001"
13        "DEC8b10bERR", # "000000010010"
14        "DEC8b10bERR", # "000000010011"
15        "DEC8b10bERR", # "000000011000"
16        "DEC8b10bERR", # "000000011001"
17        "DEC8b10bERR", # "000000011010"
18        "DEC8b10bERR", # "000000011011"
```



```
main.py x decode.py x
Q 1010011011 3 results
1 class EncDec_8B10B(object):
2     dec_lookup = [...]
1028     enc_lookup = [...]
2054     @staticmethod
2055     def dec_8b10b(data_in, verbose=False):
2056         assert data_in <= 0x3FF, "Data in must be maximum 10 bits"
2057         decoded = EncDec_8B10B.dec_lookup[(data_in)]
2058         if decoded == "DEC8b10bERR":
2059             raise Exception(
2060                 "Input to 8B10B Decoder is not a 8B10B Encoded Word")
2061         decoded = int(decoded, 2)
2062         ctrl = (decoded >> 8) & 0x1
2063         decoded = decoded & 0xFF
2064         if verbose:
2065             print("Decoded: {:02X} - Control: {:01b}".format(decoded, ctrl))
2066
2067         return ctrl, decoded
2068
```

```
main.py x decode.py x
Q encode 0 results
1 import decode as d
2 running_disp = 0
3 code_to_decode = 0b0101110100
4 p, decoded = d.EncDec_8B10B.dec_8b10b(code_to_decode, running_disp)
5 #decoded_value = bin(decoded)
6 print("Given input: ", "0101110100 - 10 bit")
7 print("Decoded Value: ", bin(decoded), "- 8 bit")
```

RESULTS:



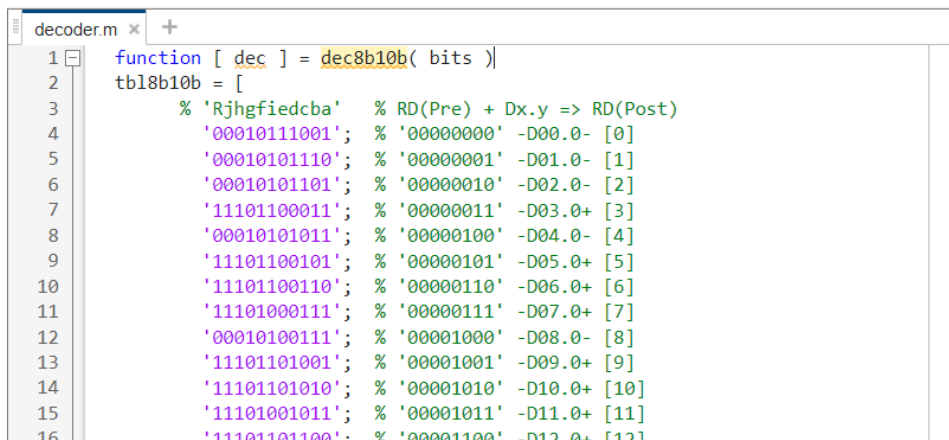
```
Run: main x
C:\Python39\python.exe C:/Users/nilay/pythonProject/main.py
Given input: 0101110100 - 10 bit
Decoded Value: 0b10110100 - 8 bit
Process finished with exit code 0
```

MATLAB SIMULATION CODE:

ASSUMPTIONS FOR INPUT:

The inputs given to the program are regular inputs as instructed by the assignment norms. The input is an 8 bit binary number and the output is a 10 bit binary encoded output.

Program is functioning without any errors and exceptions. Regular assumptions were made for coding of the encoder. Output is a RD- output.



```
decoder.m x +
1 function [ dec ] = dec8b10b( bits )
2 tbl8b10b = [
3     % 'Rjhgfiedcba' % RD(Pre) + Dx.y => RD(Post)
4     '00010111001'; % '00000000' -D00.0- [0]
5     '00010101110'; % '00000001' -D01.0- [1]
6     '00010101101'; % '00000010' -D02.0- [2]
7     '11101100011'; % '00000011' -D03.0+ [3]
8     '00010101011'; % '00000100' -D04.0- [4]
9     '11101100101'; % '00000101' -D05.0+ [5]
10    '11101100110'; % '00000110' -D06.0+ [6]
11    '11101000111'; % '00000111' -D07.0+ [7]
12    '00010100111'; % '00001000' -D08.0- [8]
13    '11101101001'; % '00001001' -D09.0+ [9]
14    '11101101010'; % '00001010' -D10.0+ [10]
15    '11101001011'; % '00001011' -D11.0+ [11]
16    '11101101100'; % '00001100' -D12.0+ [12]
```

```
Command Window

>> 0b10110100

ans =

    uint8

    180

>>
```

```
decoder.m x +
183      '10101101110'; % '10100001' -D01.5+ [161]
184      '10101101101'; % '10100010' -D02.5+ [162]
185      '00101100011'; % '10100011' -D03.5- [163]
186      '10101101011'; % '10100100' -D04.5+ [164]
187      '00101100101'; % '10100101' -D05.5- [165]
188      '00101100110'; % '10100110' -D06.5- [166]
189      '00101000111'; % '10100111' -D07.5- [167]
190      '10101100111'; % '10101000' -D08.5+ [168]
191      '00101101001'; % '10101001' -D09.5- [169]
192      '00101101010'; % '10101010' -D10.5- [170]
193      '00101001011'; % '10101011' -D11.5- [171]
194      '00101101100'; % '10101100' -D12.5- [172]
195      '00101001101'; % '10101101' -D13.5- [173]
196      '00101001110'; % '10101110' -D14.5- [174]
197      '1010111010'; % '10101111' -D15.5+ [175]
198      '10101110110'; % '10110000' -D16.5+ [176]
199      '00101110001'; % '10110001' -D17.5- [177]
200      '00101110010'; % '10110010' -D18.5- [178]
201      '00101010011'; % '10110011' -D19.5- [179]
202      '00101110100'; % '10110100' -D20.5- [180]
203      '00101010101'; % '10110101' -D21.5- [181]
204      '00101010110'; % '10110110' -D22.5- [182]
205      '10101010111'; % '10110111' -D23.5+ [183]
206      '10101110011'; % '10111000' -D24.5+ [184]
207      '00101011001'; % '10111001' -D25.5- [185]
208      '00101011010'; % '10111010' -D26.5- [186]
209      '10101011011'; % '10111011' -D27.5+ [187]
210      '00101011100'; % '10111100' -D28.5- [188]
```



Command Window

CONCLUSION

We can conclude that the above code for the 8b/10b decoder in python is working accurately and well as we have verified it using MATLAB. Hence the objective of the project has been met.

REFERENCES

- Xu, Q., & Liu, H. (2012). 8b/10B encoder design. *Proceedings of the 2nd International Conference on Computer Application and System Modeling*. <https://doi.org/10.2991/iccasm.2012.25>
- Wikimedia Foundation. (2022, March 3). *8b/10b encoding*. Wikipedia. Retrieved April 10, 2022, from https://en.wikipedia.org/wiki/8b/10b_encoding

THANK YOU