**Home assignment**

To communicate with DP-DSP we define a unique protocol over Serial communication. The protocol states that a data packet should be formatted as followed:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Delimiter | Length | Control | Opcode | Payload | Checksum | Delimiter |
| 1 byte | 1 byte | 1 byte | 1 byte | Up to 25 bytes | 1 byte | 1 byte |

**Notes:**

Delimiter is 0X7E. Length is the length of control+opcode+payload. Control is the value 0 always.

Opcode is hexadecimal number represented as int.

Payload is a list of up to 25 hexadecimal numbers represented as ints.

Checksum is the sum of the packet **without** delimiters.

1. Write a function that encapsulates the input in the packet format given above.
2. Apply byte stuffing: To distinguish between message delimiters and 0x7E data, the protocol uses “byte stuffing” mechanism. Hence, whenever the value 0x7E (or 0x7D) is appearing in the message (excluding delimiters), an escape byte (equals to 0x7D) is sent, followed by the original data octet with bit 5 inverted. For example, if the data is 0x24,0x7E, 0x35, the actual sent data will be 0x24, 0x7D, 0x5E, 0x35.  
   Whenever an escape byte is appearing an escape byte (equals to 0x7D) is sent, followed by an escape byte with bit 5 inverted.  
   \* bit 5 is where the ‘X’ is: 00X0 0000

Example:

A call to:

encapsulate\_frame(0x12, [1, 7, 0, 8, 0])

should return: (either or byte-array whatever works for you)

[126, 7, 0, 18, 1, 7, 0, 8, 0, 41, 126]

['0x7e', '0x7', '0x0', '0x12', '0x1', '0x7', '0x0', '0x8', '0x0', '0x29', '0x7e']

Text

Description automatically generated

1. Here are examples of a Read channel values Request & Response.

#### Request Packet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Size | Byte # | Value | Description |
| Opcode | 1 Byte | 4 | 0x3B | Channels readout |
| Payload | 1 Byte | 5 | Variable | Index 0-4:  0x00 – index 1  0x01 – index 2  0x02 – index 3  0x03 – index 4 |
| Payload | 1 byte | 6 | Variable | Channels mask  The response packet will include only the value of the channels whose corresponding bit in the mask was set to ‘1’. |

#### Response Packet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Size | Byte # | Value | Description |
| Opcode | 1 Byte | 4 | 0x3B | Channels readout |
| payload | 1 byte | 5 | variable | Index |
| Payload | 3 bits | 6[7:5] | Variable | channel 0-7 |
| Payload | 21 bits | 6[4:0],7-8 | Variable | Value.  The returned value is the measured **voltage** in mV. |
| : | | | | |
| Payload | 3 bits | 3n+6[7:5] | Variable | channel 0-7 |
| Payload | 21 bits | 3n+6[4:0],3n+7 – 3n+8 | Variable | Value.  The returned value is the measured **voltage** in mV. |

Assume the request asks for index 1, channels ~~1,2,3,6,7~~ 0,1,2,5,6(0b01100111 = 0x67)

Then an example response is:  
[126, 18, 0, 59, 0, 0, 0, 2, 32, 0, 1, 64, 0, 0, 160, 0, 0, 192, 0, 16, 32, 126]

**Create a function that parses the response and returns the channels mV values.** Use the example above to test your function. The return format is for you to decide, it should be clear which value matches each channel.

Hint: the result values should be around these values for the example above [0, 1000, 2000, 5000, 6000]