Title: Protocol Compressor and Optimizer

Final Implementation and results:

In this project the main aim was to implement the huffman encoding and decoding algorithm. Various implementations have been done to achieve this.

Overall three programs have been written :

1. The huffman_tree_gen.c generated the huffman tree for a given input string present in the file input2.txt. This log files has been generated by the Linux syslog file. This program generated the required lookup-table and huffman.h header files required for encoding and decoding on the KL25Z and the PC. The output of this program is as follows:

```
$ ./huffman.exe
Huffman Lookup table below:
huffman_table_t huffman_table[] = {
{0, 0x00, 0},
{1, 0x00, 0},
{2, 0x00, 0},
{3, 0x00, 0},
{4, 0x00, 0},
{5, 0x00, 0},
{6, 0x00, 0},
{7, 0x00, 0},
{8, 0x00, 0},
{9, 0x00, 0},
{10, 0x4c, 7},
{11, 0x00, 0},
{12, 0x00, 0},
{13, 0x00, 0},
{14, 0x00, 0},
{15, 0x00, 0},
{16, 0x00, 0},
{17, 0x00, 0},
{18, 0x00, 0},
{19, 0x00, 0},
{20, 0x00, 0},
{21, 0x00, 0},
{22, 0x00, 0},
{23, 0x00, 0},
{24, 0x00, 0},
{25, 0x00, 0},
{26, 0x00, 0},
{27, 0x00, 0},
{28, 0x00, 0},
{29, 0x00, 0},
{30, 0x00, 0},
{31, 0x00, 0},
{32, 0x00, 3},
{33, 0xf46, 12},
{34, 0x134, 9},
{35, 0x2ba, 10},
{36, 0x00, 0},
{37, 0x00, 0},
{38, 0x00, 0},
{39, 0x5a, 8},
{40, 0x136, 9},
{41, 0x135, 9},
{42, 0x00, 0},
{43, 0xde4, 12},
{44, 0x6f3, 11},
{45, 0x10, 5},
{46, 0x12, 6},
{47, 0x0a, 6},
{48, 0x1c, 5},
{49, 0x16, 5},
{50, 0x23, 6},
{51, 0x18, 7},
{52, 0x0c, 5},
{53, 0x04, 5},
{54, 0x1f, 7},
{55, 0x34, 7},
{56, 0x6e, 7},
{57, 0x27, 7},
```

In the above figure, as it can be seen the look-up table has been generated and the corresponding header file will be generated from this look-up table.

A unit test function is also run, for a few random input strings to verify the proper encoding and decoding by the algorithm on the input data.

2. The KL25Z 'PES-Project' program built on MCUexpresso is using the previously used command processor with modifications done for the Huffman encoding, decoding. In the code file huffman.c a syswrite test function written which outputs multiple strings of variable length using printf. These outputs are received by the PC side program. The way this is achieved is that the syswrite functionality is re-written in the uart.c file. The buffer received is send to encoding function. The encoding function uses the look-up table to generate variable length characters codes and these are framed in form of bytes. For one input string to syswrite a '0xFF' end of character token is also added to differentiate the frames on the PC decompression side. Thus **the huffman encoding is tied to the sys_write functionality** of the command processor program. After this a unit test function is run again to test encoding and de-coding on the KL-25Z itself and after all of this, the normal command processor program is run.
3. The third program developed is for the PC side, which is receiving the incoming encoded bytes from the KL25Z and is de-framming the data using the 0xFF token. A de-framed input is given to the decode function. Multiple strings are received from the UART of the KL25Z which are decompressed.

   Finally, the decompressed strings are tested using automated tests, **by comparing the input strings originally compressed by the KL25Z with the decompressed strings on the PC**.

   This is shown below:

```
vishalraj@LAPTOP-DBOOPNST /cygdrive/d/PES/Project/Code/Huffman
$ ./serial_old.exe
Waiting for encoded inputs from target controller...
encoded string dump:(hex)

ca 3f 1d 7a 2e 3e 1d fa 4b a6 8d 3d d7 a0 00

Decoded string:
entering a random strin

Test 1 passed!

------------------------------------------------------------------------------------------
encoded string dump:(hex)

7e 69 e3 4f 75 e8 b8 b4 66 18 40 b7 7e b3 13 a6
9a 69 a6 84 2d b5 7a 2e ba e0 cb 00

Decoded string:
test string 12345 RaNDommmmm cAsinggg

Test 2 passed!

------------------------------------------------------------------------------------------
encoded string dump:(hex)

f4 78 ed 3c 2f da 53 e3 4f fd 24 1e de 3b 07 e9
21 bf f7 a1 b0 6e 04 3a bc c6 cb

Decoded string:
Just gonna stand there and watch me cry

Test 3 passed!

------------------------------------------------------------------------------------------
encoded string dump:(hex)

89 88 16 b0 e3 15 c2 29 84 1b 5f 4d fe 78 35 ca
77 6e c2 f2 cf f9 f9 40 91 f0 b8 59 80 e3 00

Decoded string:
Dec 11 04:05:45 vishal-Lenovo-ideapad-520S-14

Test 4 passed!

------------------------------------------------------------------------------------------
encoded string dump:(hex)

89 88 16 b0 e3 15 c4 ea 61 06 d7 d3 7f 9e 0d 72
9d db b0 bc b3 fe 7e 50 24 7c 2e 16 65 43 c9 87
5c ba 4b d3 a6 e2 45 b7 93 ce 94 b3 46 5f df 2b
47 66 12 aa 85 75 c1 cf 5b 4f 7e f2 56 22 13 a7
97 05 40 03 b3 5d 39 c8 c5 14 51 5d 72 e5 2f 4e
9b 85 2e b1 d1 46 f6 47 4f 9a 28 d6 02 00

Decoded string:
Dec 11 04:09:45 vishal-Lenovo-ideapad-520S-14IKB org.gnome.Shell.desktop[2145]: #0 0x7ffed15c99d0 I   resource:///org/gnome/gjs/modules/

Test 5 passed!

---------------------------------
   All PC decode tests passed!
---------------------------------

Average compression ratio achieved:62.669563

------------------------------------------------------------------------------------------
```

As it can be seen above the decoding test passed for all the inputs given by the KL25Z with the actual decoded strings displayed. A hex-dump of the encoded data is also displayed. Thus the compressed data on the KL25 is decompressed on the PC-side and tested for correctness. An example of a decode string can be seen below:

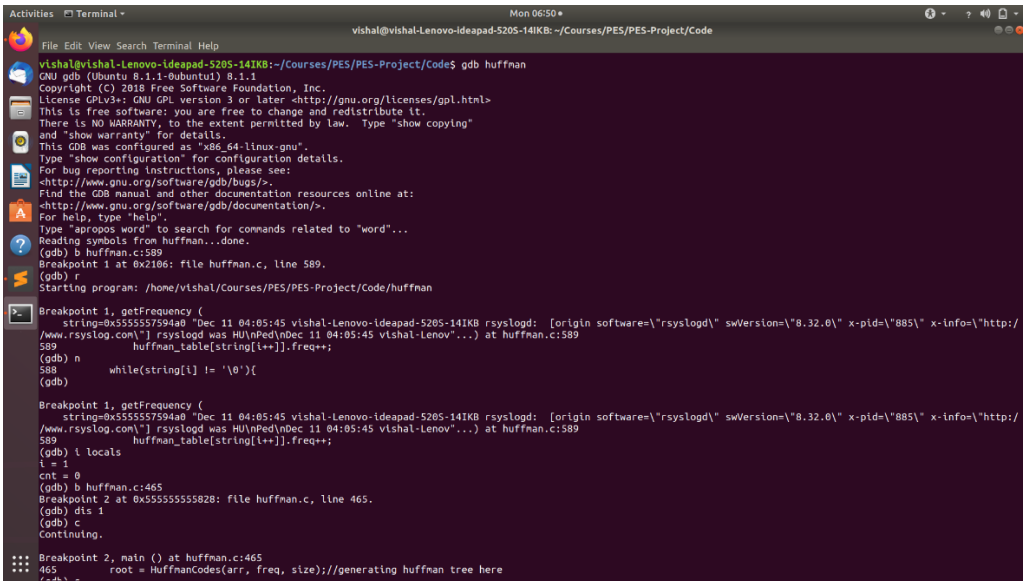
```
Decoded string:
Dec 11 04:09:45 vishal-Lenovo-ideapad-520S-14IKB org.gnome.Shell.desktop[2145]: #0 0x7ffed15c99d0 I   resource:///org/gnome/gjs/modules/

Test 5 passed!
```

RESULT: Finally, the average compression ratio achieved by the algorithm and the complete system working is found to **be 62.669%, i.e almost 40% reduction** of size is observed.

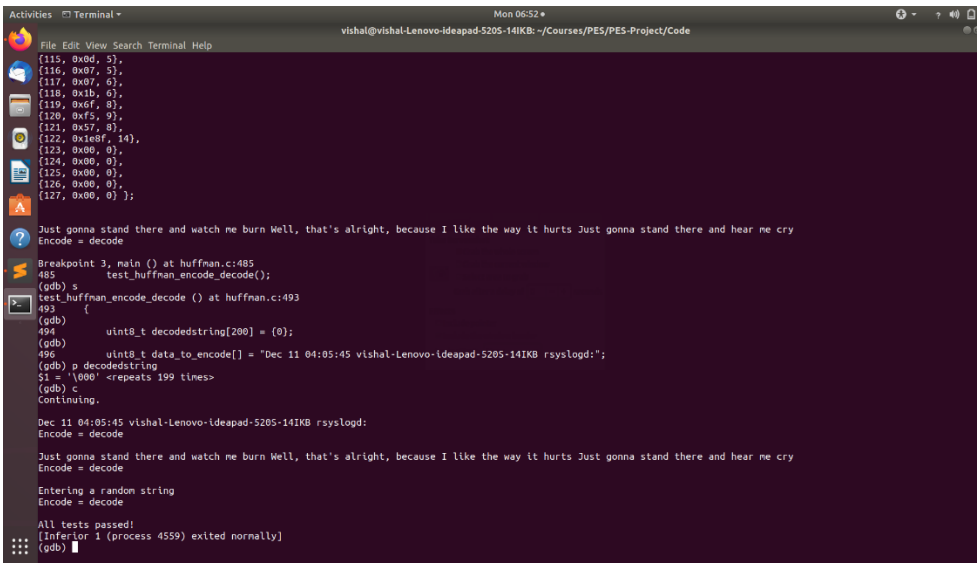
```
Average compression ratio achieved:62.669563
```

Other technologies used:

The GDB command line debugger was used at various stages of project development. It was used for debugging and seeing the results of the Huffman algorithm and also of the encoding and decoding steps. The use of GDB was found to be very beneficial as many issue were able to be resolved using the various features provided by the GDB.



Various function of the GDB were used like breakpoint execution, step execution, info of variables, memory contents of pointers, backtrace and enabling-disabling of breakpoints.



End of document