Title: Protocol Compressor and Optimizer

Project functionality

With an increasing number of electronics devices and with the growth of various IOt devices around us, it is inevitable that the data these devices exchange has increased and is likely to peak even more in the future. Also, on the other hand, new protocols for information exchange are being developed which in most cases use the similar serial line communication on the physical layer. Hence it is necessary to devise a method to reduce the overhead on the physical layer and make it more efficient for information transfer. One way to achieve this is to compress the final data packet or frame that is being transmitted from one device to another so as to increase the transmission speed and also reduce the size/overhead by reducing the no of bits being transmitted, thus making the communication efficient. Therefore, this project will rely upon using a popular lossless compression algorithm called Huffman coding for reducing the effective bit rate of transmission of protocol frames from one device to another.

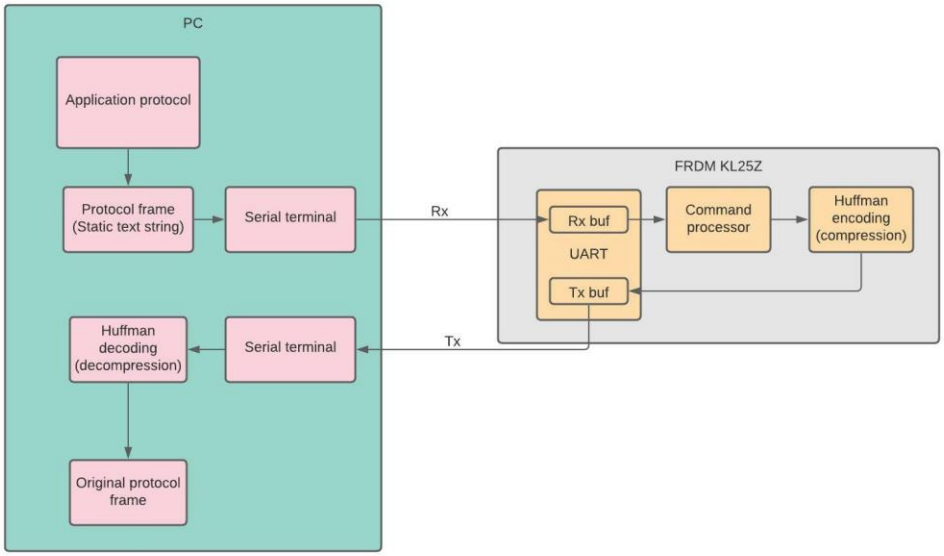

Figure 1: Block diagram of project software architecture

To demonstrate the functionality of the project, a predefined protocol frame in form of a text string will be sent through the PC terminal over the serial line to the KL25Z. On receiving the frame through UART, our embedded device will fetch the actual frame from the command processor and process the string further by performing compression over it using Huffman encoding. Once the string is processed and the compressed data is framed, it is again transmitted over UART from the embedded device to the PC. At this point, the PC will have another program running on a terminal which is used to read

incoming bytes from the embedded device and on receiving the encoded frame, it will send an acknowledgement to the embedded device of successful reception. After this, the program on the PC will continue with the next step to perform decompression or 'Huffman decoding' on the received frame. This will result in formations and retrieval of the original datagram which was transmitted.

To verify the correct operation of the system, a simple test program will also be run after retrieval of the compressed data, which will compare the original transmitted protocol frame string and decompressed protocol frame.

This program will be compatible with any high-level messaging protocol and to verify its functionality, data frames or packets of different protocols in the form of string can be used to test the system.

Technologies to be used

- The main KL25Z peripheral to be used for this project will be the UART0 module. The onboard USB to UART interface will be used. The configuration will be set such that the highest data transfer rate is set as default which can also be modified by changing simple macro definitions in the source code.
- Along with the UART, the earlier developed command processor can be used with all the existing underlying components like cbfifo, interrupts , sysread, syswrite, or the the inbuilt UART functionality provided by the SDK can be used with a new, much simpler command processor for interaction with the user.
- For incremental development the compression algorithm would first be written and tested on the gcc compiler on the PC. For new methods of unit testing, the 'cmocka' or 'ucunit' test framework would be used to test the individual modules of the project being developed.
- For the new method of debugging of the program the GNU GDB debugger will be used using the command line option.
- Data compression using Huffman coding will be performed on incoming bytes from the PC terminal and decompression will be performed on the data coming from KL25Z on the PC.

New learnings

- The new aspects learnt will firstly be related to the compression algorithm itself, one which will require a good understanding of algorithm encoding i.e creation of variable length prefix codes and decoding algorithm to bring the data back to its original form and tuning of the algorithm according to application.
- The second learning will be understanding of the various protocols whose message frames will be used as an input to the compression algorithm. The main options of

protocols whose data frames can be used are Modbus RTU, Modbus TCP, CAN bus, TCP/IP datagram, or a JSON packet format used in network-related applications.

- The third new learning will be that of a test framework like cmocka for automated unit testing of the software modules developed. The use of the framework is API based, hence this will require good review of the tool documentation.
- The other new thing which will be learnt will be the use of GNU GDB debugger, at various stages of program development to debug and analyze the algorithm and various other software modules through the command line.
- The required documentation will be - KL25Z reference manual, specific application protocol document/reference, Huffman coding algorithm documents, cmocka and ucunit tools documents and tutorials and the GNU GDB manual.

Testing strategy

As previously outlined, for testing the various modules of the project, an automated test framework(either cmocka or ucunit) will be used. Along with this method of testing, a normal method of automated testing using test functions, similar to the technique used in course assignments, will be used for basic validation of a few modules.

The approach for testing would be to first develop the Huffman encoding and decoding algorithm on the PC and perform unit tests on that using a test framework. After achieving correct results, the compression part to be used on the embedded device would be tested on the KL25Z using the second method of testing mentioned above(using test functions). A final test during program execution would also be done on the PC side, after it has decompressed the encoded data coming from the KL25Z in order to compare the original string and the one regenerated to verify the integrity of the overall system.

References:

1. ‘PES Final Project’ document by Howdy Pierce.
2. KL25 Sub-Family Reference Manual Rev.3.
3. https://en.wikipedia.org/wiki/Huffman_coding.
4. https://cmocka.org/.
5. http://www.ucunit.org/.
6. https://www.gnu.org/software/gdb/documentation/.
7. https://www.modbustools.com/modbus.html.
8. https://www.beningo.com/getting-started-with-json-for-embedded-developers/.

End of document