ECEN 5813 - Huffman Coding for Debug Messages

Final Project Report

Date	04/30/2022
Prepared By (Name)	Ayush Gupta

Overview

Implementation of Huffman Coding algorithm on FRDM KL25Z development board to compress the messages sent over Serial Port.

Hardware Components

- FRDM KL25Z Development board
- A PC with ubuntu operating systems

Software Components

- MCU Expresso IDE version - 1.2.0
- GCC compiler
- Teraterm/Putty to test the UART Transmitter and Receiver

Software Modules

Following are the software modules required to realize the proposed solution:

- **UART0 module**: Initialize the UART0 module on KL25Z to be able to sent out data on the serial port.
- **Huffman Coding Algorithm module**: This module will have all the Huffman coding logic and to be called before transmitting the string on to the serial port.
- **Circular Queue Module**: This module will have the implementation of Circular Queue which will act as a buffer, required to store the transmitting data.
- **Test Huffman Module**: This module will have the automated test cases to verify the Huffman Coding Algorithm.
- **Test Circular Queue Module**: This module will have the automated test cases to verify the working of Circular Queue.

Technologies Involved

- Huffman Coding (Data Compression Technique)
- Circular Buffers (to store the messages to be sent over the serial port)

Project Planning

The Project is divided into various submodules and then completed one step at a time. Here is the modular approach that has been followed during the execution of this project.

Generation of Huffman Map Table

A python script is used to make the Huffman table for encoding and huffman table for decoding using a corpus. This process is very flexible to do. All the user has to add sentences in the corpus and run the python script. It is generate the huffman tree for both encode and decode end. User then has to replace the previous huffman tables with the latest huffman table.

Instructions

There are 3 directories in this repository which a user should be aware of.

1. huffman_coding: contains the KL25z software.

   `To run the KL25z code, open the project in MCUExpresso, select huffman_coding from Project Exploler > right click and select debug as PEMicro Probes.`

2. huffman code generator: this directory has the corpus file and the python script to create encodeHuffmanCode.txt and decodeHuffmanCode.txt which will all be available in this repository after creation.

   To run the python script, open terminal and go to this directory (ubuntu system), and run the following command. Make sure you have corpus.txt file in the same directory.

   ```
   python3 huffman.py
   ```

3. serial receiver: An Ubuntu based C scripts which communicates with KL25Z using serial port.

   To run the software, open terminal in ubuntu and go to this directory. Run the following commands to execute the program.

   ```
   gcc read_serial.h read_serial.c main.c -o readSerial
   ```

   ```
   ./readSerial
   ```

   Note: Make sure you have readSerial executable running on the ubuntu before starting with KL25Z. The program starts when KL25Z sends a dummy string to the PC.

Functionalities

1. Following commands are present in the system.

   - Author: returns the name of the author.
   - dump : returns the hexdump of data from requested memory.
   - info: return a string in the format "Version 1.0 built on at <YYYY-MM-DD_HH:mm:ss> Commit <commit_id>"
   - compare: Huffman compression stats

2. Huffman code generator to generate HUFFMAN encode and decode tables.

Configuring the Serial Port

- KL25z: The macro BAUD_RATE present in uart.h can easily be configured to work with any Baud Rate.
- Ubuntu: The macro SERIAL_PORT_KL25Z to set the serial port (in my case: /dev/ttyS8) and BAUD_RATE (in my case B115200) to configure the serial port.

Currently, the serial port is working on **115200 Baud Rate, no parity and 1 stop bit mode**.

Software Flow Control

System Overview: There is a program running on KL25z which will receive the commands from Serial Port (UART0) and process it. If the command is valid it will send its response back which will be encoded using the Huffman Algorithm. Here is a high-level system overview.

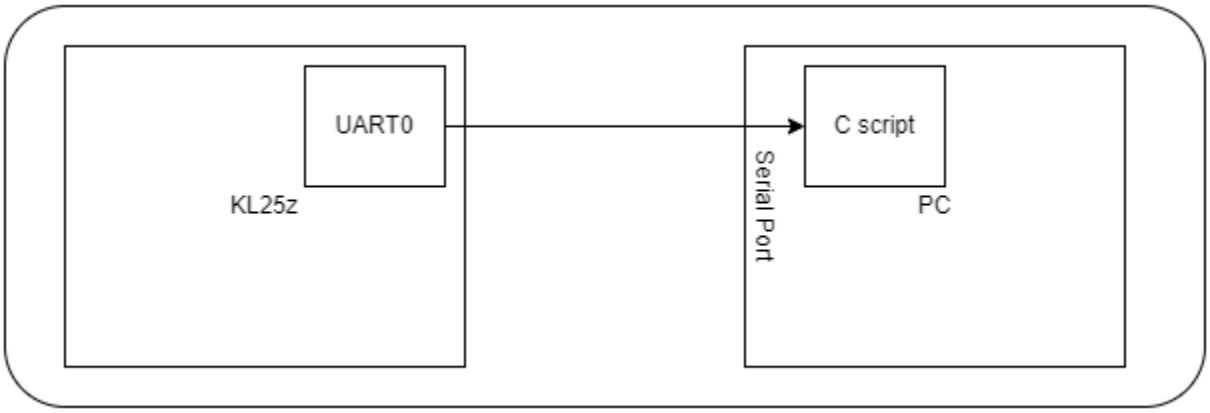

Figure 1 System Overview

KL25z: Encode the outgoing messages using HUFFMAN coding technique. The string is currently being compressed in the __sys_write function associated RedLib library.

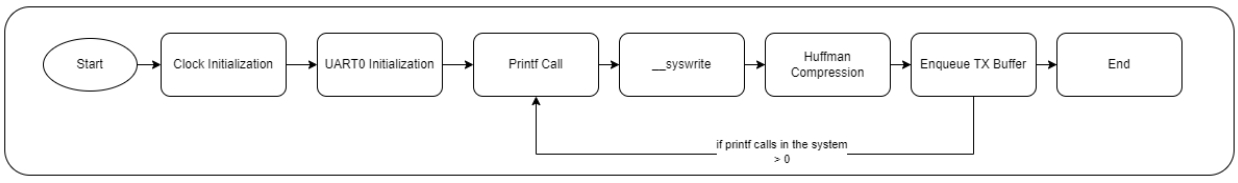

Figure 2 Huffman encoding algorithm

PC Front: On the serial front the data is being decompressed using the HUFFMAN TABLE (symbol, code & number of bits). If HUFFMAN_END_SYMBOL ('\0') is detected, the remaining bits are discarded.

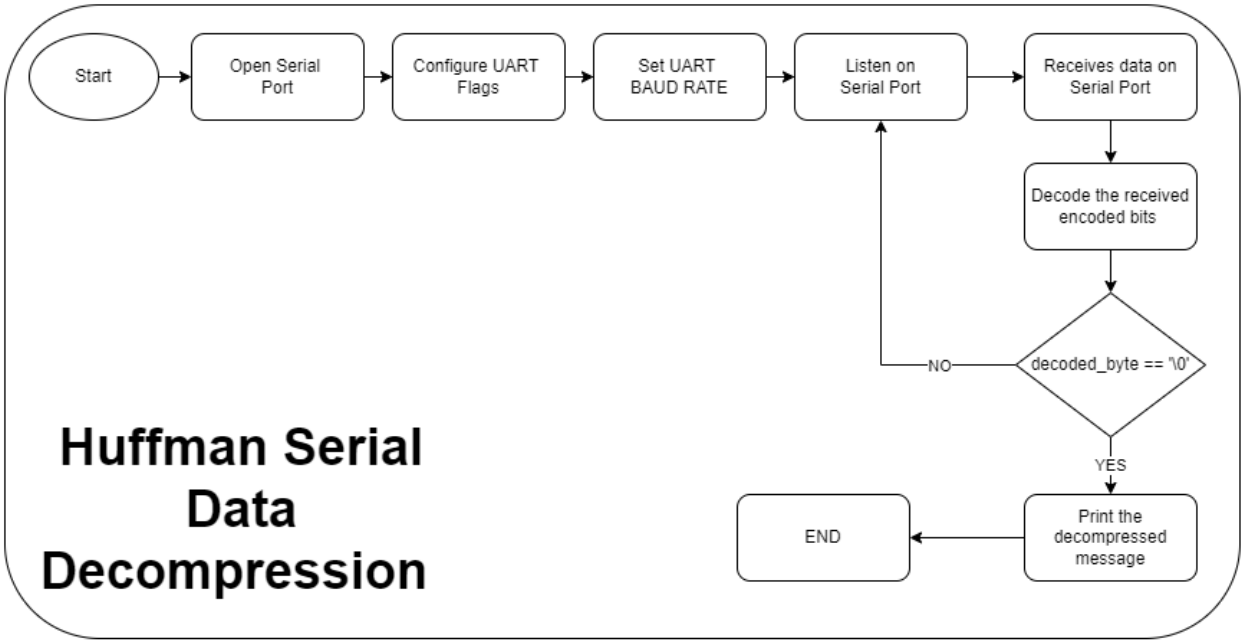

Figure 3 Huffman Data decompression flow diagram

Test Results

The test plan for verifying the Huffman implementation and integration is given in the Test Plan.

1. Unit test cases were written to verify each step. Following are the tests that are executed on system start-up.
    a. Test Huffman Coding Implementation: If system is in DEBUG mode, huffman test APIs will execute on the KL25Z, which will execute the Huffman Encoding and Decoding and uses assert statements while comparing the two results.
    b. Test Circular Queue Implementation: If system is in DEBUG mode, circular queue test APIs will execute on the KL25Z, which will perform the various scenarios on Circular queue and uses assert statements while comparing the results.

Once these tests are passed, the system proceeds to the command processor part.

Initial scope of this project was to implement the Huffman algorithm. As that part is completed, I jumped on to the assignment 6 and extended it by integrating it with my implementation of Huffman Algorithm to compress the data.

2. The commands are kept as it is from assignment 6. Here are a few screenshots showing the working of various commands.

```
root@Ayush:/mnt/d/Coursework/StudyMaterial/PES/Final Project/huffman_coding/serial receiver# ./readSerial
ECEN5813 Principles of Embedded Software Final Project

>author
Ayush Gupta

>Author
Ayush Gupta
```

Figure 4 author command

```
>dumo 0 80
Invalid Command: dumo

>dump 0 80

0000_0000  00 30 00 20 D5 00 00 00 43 01 00 00 9D 0B 00 00
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00
0000_0040  4F 01 00 00 57 01 00 00 5F 01 00 00 67 01 00 00

>dump 0 128

0000_0000  00 30 00 20 D5 00 00 00 43 01 00 00 9D 0B 00 00
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00
0000_0040  4F 01 00 00 57 01 00 00 5F 01 00 00 67 01 00 00
0000_0050  6F 01 00 00 77 01 00 00 7F 01 00 00 87 01 00 00
0000_0060  8F 01 00 00 97 01 00 00 9F 01 00 00 A7 01 00 00
0000_0070  79 21 00 00 B7 01 00 00 BF 01 00 00 C7 01 00 00
```

Figure 5 Hex Dump

```
>info
Version 1.0 built on Ayush at 2022-05-02_16:20:10 Commit a20eabf388c84cd75fa401c06d6f784525d0677d
```

Figure 6 Info Command

3. A new command which is added to the system is compare. Upon receiving this command the stats related to Huffman will be displayed on the terminal. Below is the picture depicting the same.

```
>compare
Number of received bytes: 430
Uncompressed message length in bytes: 1061
Percentage compression: 40.53
```

Figure 7 Compression percentage

Code Explanantion

https://user-images.githubusercontent.com/89823539/166399931-e08a477d-55db-49d2-a3ed-e65b5351f48d.mp4

Demo

https://user-images.githubusercontent.com/89823539/166399957-317955ea-bc14-4a01-a92c-930349ffb0e4.mp4

Stats

https://user-images.githubusercontent.com/89823539/166399960-c058b555-ed9f-4b0e-b59c-b196044652ef.mp4

Things to remember

Keep in mind that the characters in the strings below should be a part of the corpus passed to python script. Otherwise, there will be no encoded bits and the program will stuck in a loop.

'\0' is used as a HUFFMAN ending character, anything after HUFFMAN_LAST_SYMBOL will be discarded and hence will be lost.

System Limitations & Possible Future Improvements

The corpus that is selected for this application is a bit limited to what it can encode. As the size increases the number of encoded bits increases and with that the time of encoding the bits increases exponentially. A high-quality corpus can be chosen to make improvements. For this application, this corpus can work.

The dump function can print upto 130 bytes of data. This is due to the fact that for dump commands I am currently sending one encoded string which is limited by the size of buffers defined in the system. Although the change is minimalistic as all the buffer lengths are defined using a MACRO which are configurable in a single go.

Huffman encoding and decoding can be implemented on both the ends to compress the data further.

References

https://en.wikipedia.org/wiki/Huffman_coding

https://brilliant.org/wiki/huffman-encoding/

https://www.cyberciti.biz/faq/find-out-linux-serial-ports-with-setserial/

https://www.mouser.com/pdfdocs/FRDM-KL25Z.pdf

https://www.programiz.com/dsa/huffman-coding