# ECEN 5813 - Huffman Coding for Debug Messages
This repository contains the code for Huffman Coding Algorithm for UART messages on FRDM KL25z development board.

## 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:
<b>UART0 module</b>: Initialize the UART0 module on KL25Z to be able to sent out data on the serial port.
<b>Huffman Coding Algorithm module</b>: This module will have all the Huffman coding logic and to be called before transmitting the string on to the serial port.
<b>Circular Queue Module</b>: This module will have the implementation of Circular Queue which will act as a buffer, required to store the transmitting data.
<b>Test Huffman Module</b>: This module will have the automated test cases to verify the Huffman Coding Algorithm.
<b>Test Circular Queue Module</b>: 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](project_plan.xlsx) is the modular approach that has been followed during the execution of this project.

## Generation of Huffman Map Table
A [python script](huffman%20code%20generator/huffman.py) is used to make the [huffman table for encoding](huffman%20code%20generator/encodeHuffman.txt) and [huffman table for decoding](huffman%20code%20generator/decodeHuffman.txt) using a [corpus](huffman%20code%20generator/corpus.txt). This process is very flexible to do. All the user has to add sentences in the [corpus](huffman%20code%20generator/corpus.txt) 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.
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```.
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
```

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
```

<b>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. </b>

## Functionalities
Following commands are present in the system.
Author: returns the name of the author.
dump <start address> <end address>: returns the hexdump of data from requested memory.
info: return a string in the format "Version 1.0 built on <hostname> at <YYYY-MM-DD_HH:mm:ss> Commit <commit_id>"
compare: Huffman compression stats
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 <b>115200 Baud Rate, no parity and 1 stop bit mode</b>.

## 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.
![overview](Screenshots/pes3.png)

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

![kl25z_software_flow](Screenshots/pes1.png)

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.

![PC_software_flow](Screenshots/pes2.png)

### Test Results
The test plan for verifying the Huffman implementation and integration is given in the [Test Plan](test_plan.xlsx).

Unit test cases were written to verify each step. Following are the tests that are executed on system start-up.
Test Huffman Coding Implementation: If system is in ```DEBUG``` mode, [huffman test APIs](huffman_coding/source/test_huffman.c) will execute on the KL25Z, which will execute the Huffman Encoding and Decoding and uses assert statements while comparing the two results.
Test Circular Queue Implementation: If system is in ```DEBUG``` mode, [circular queue test APIs](huffman_coding/source/test_cbfifo.c) 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.

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

![cmd1](Screenshots/author_cmd.jpg)

![cmd2](Screenshots/dump_cmd.jpg)

![cmd1](Screenshots/info_cmd.jpg)

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.

![cmd1](Screenshots/compare_cmd.jpg)

### 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

```
Description: This file deals deals with the functions in the read_serial.h file.
Show the related statistic to compare HUFFMAN CODING compression.
*******************************************************************************
*/
#include <stdio.h> #include <string.h>
//include Linux headers
#include <fcntl.h>     // Contains file controls like O_RDWR   #include <errno.h>     // Error
integer and strerror() function #include <termios.h>      // Contains POSIX terminal control
definitions #include <unistd.h>       // write(), read(), close() functions


#include "read_serial.h"

#define
NUMBER_OF_COMMANDS (
6)
#define
COMMAND_LENGTH        (
15)
#define
NUMBER_OF_TEST_CASES        (
17)
#define
BUFFER_LENGTH   (
40)
#define
USER_MESSAGE_LENGTH (
50)
#define
ASCII_BACKSPACE_CHARACTER (
8)

int main() {
//define am integer to hold serialport value int serialPort;
// Create new termios struct, we call it 'tty' for convention struct termios tty;

//configure serial port, open the port, set baudrate serialPort = configureSerialPort(&tty);
if(serialPort == -1){
return -1;

}
//store received messages
char  msg[RECEIVED_MESSAGE_LENGTH];
//clearing the buffer
memset(msg,  0  ,  RECEIVED_MESSAGE_LENGTH);

//receive and decode the bytes
decodeMessages(serialPort,  msg,  RECEIVED_MESSAGE_LENGTH); printf("%s",  msg);
memset(msg,  0  ,  RECEIVED_MESSAGE_LENGTH);

//user message string
char  userInput[USER_MESSAGE_LENGTH]; int  userIndex  =  0;
char ch; while(1){
printf("\r\n>"); userIndex  =  0;
while  (userIndex  <  USER_MESSAGE_LENGTH){ scanf("%c",  &ch);

if  (ch  ==  ASCII_BACKSPACE_CHARACTER){
//ignore  backspace  character if(userIndex  >  0)
userIndex--;
}
else  if  ((ch  ==  '\r')  ||  (ch  ==  '\n')){
//break  if  newline  character  found break;
}
else{
//copy  the  character  and  increment  the  index userInput[userIndex]  =  ch;
userIndex++;
}
}

if  (strncmp("compare",  userInput,  userIndex)  ==  0){
//display  the  stats getStats();
}
else{
//added  \r\n  at  the  end  of  the  string  for  decoding  on  the  kl25z userInput[userIndex++]  =  '\r';

userInput[userIndex++]  =  '\n';

//initialize  counters  =  0 int  countWriteBytes  =  0;
int  msgLength  =  strlen(userInput);

char * cmd = userInput;
//wait  till  all  the  bytes  are  sent  to  the  serial  port while(msgLength  >  0){
//send  data
countWriteBytes  =  write(serialPort,  cmd,  msgLength); msgLength  -=  countWriteBytes;
cmd += countWriteBytes;
}

//receive  and  decode  the  bytes
decodeMessages(serialPort,  msg,  RECEIVED_MESSAGE_LENGTH); printf("%s",  msg);
memset(msg,  0  ,  RECEIVED_MESSAGE_LENGTH);
}
}

//close  the  serial  port close(serialPort); return  0;
};
/****************************************************************************
```

```
**
@file  read_serial.c
********************************************************************************
**
* Date: 04-30-2022
Author:Ayush  Gupta
Description: This file deals with the reception and decoding of huffman encoded
serial  data.
********************************************************************************
*/

#include  "read_serial.h" #include  <stdio.h>
#define  MIN(X,  Y)     (((X)  <  (Y))  ?
(X) : (Y))
#define  MASK(X)      ((1<<X)  -  1)
#define  INVALID_HUFFMAN_SYMBOL      (0xFFFFFFFF)

#define  BITS_IN_A_BYTE    (8)

//variables  to  keep  a  track  of  data  incoming int  countBytes  =  0;
int  countDecodedBytes  =  0;

//Huffman  table:  symbol,  encoded  messages  and  length  of  encoded  messages typedef  struct
{
uint8_t  symbol; char  code[20]; int nBits;
} decodeHuffmanCode_t;


/*************************************************************************
@name  decodehuffmanCode
*
@description huffman characters, encoded bits and length of encoded bits.
**********************************************************************/ decodeHuffmanCode_t
decodehuffmanCode[]  =  {
{' ' ,      "00"     ,2        },
{'0' ,     "10"     ,2        },
{'\r',     "11010"          ,5        },
{'\n',     "11011"          ,5        },
{'i' ,      "01011"          ,5        },
{'p' ,     "010000"        ,6        },
{'r' ,      "010001"        ,6        },
{'E' ,     "010100"        ,6        },
{'1' ,     "011100"        ,6        },
{'m' ,    "011110"        ,6        },
{'a' ,     "011111"        ,6        },
{'t' ,      "110000"        ,6        },
{'o' ,     "110001"        ,6        },
{'e' ,     "110010"        ,6        },
{'d' ,     "111111"        ,6        },
{'n' ,     "111101"        ,6        },
{'>' ,     "1110100"      ,7        },
{'l' ,      "1110101"      ,7        },
{'I' ,      "1110110"      ,7        },
{'u' ,     "1110111"      ,7        },
{'4' ,     "1110011"      ,7        },
{'C' ,    "1110001"      ,7        },
{'_' ,    "1111001"      ,7        },
{'9' ,    "0110000"      ,7        },
{'D' ,    "0110001"      ,7        },

{'2'      ,         "0101010"      ,7        },
{'A'      ,         "0101011"      ,7        },
{'T'      ,         "0110011"      ,7        },
{'f'      ,         "0110100"      ,7        },
{'s'      ,         "0110101"      ,7        },
{'M'      ,         "0110111"      ,7        },
{'3'      ,         "0111011"      ,7        },
{':'      ,         "11111011"     ,8        },
{'h'      ,         "11111000"     ,8        },
{'8'      ,         "11111001"     ,8        },
{'N'      ,         "11100000"     ,8        },
{'R'      ,         "11100001"     ,8        },
{'B'      ,         "11100100"     ,8        },
{'v'      ,         "11100101"     ,8        },
{'c'      ,         "11110000"     ,8        },
{'H'      ,         "01001001"     ,8        },
{'G'      ,         "01001100"     ,8        },
{'y'      ,         "01001101"     ,8        },
{'L'      ,         "01001110"     ,8        },
{'x'      ,         "01001111"     ,8        },
{'w'      ,         "01100100"     ,8        },
{'U'      ,         "01100101"     ,8        },
{'S'      ,         "01101100"     ,8        },
{'b'      ,         "01101101"     ,8        },
{'5'      ,         "01110100"     ,8        },
{'z'      ,         "11001100"     ,8        },
{'P'      ,         "11001101"     ,8        },
{'O'      ,         "11001111"     ,8        },
{':'      ,         "11111011"     ,8        },
{'h'      ,         "11111000"     ,8        },
{'N'      ,         "11100000"     ,8        },
{'R'      ,         "11100001"     ,8        },
{'B'      ,         "11100100"     ,8        },
{'v'      ,         "11100101"     ,8        },
{'c'      ,         "11110000"     ,8        },
{'Z'      ,         "010010000"    ,9        },
{'Y'      ,         "010010001"    ,9        },
{'Q'      ,         "010010100"    ,9        },
{'K'      ,         "010010101"    ,9        },
{'X'      ,         "010010110"    ,9        },
{'W'      ,         "010010111"    ,9        },
{'7'      ,         "011101010"    ,9        },
{'k'      ,         "011101011"    ,9        },
{'V'      ,         "110011101"    ,9        },
{'j'      ,         "111100010"    ,9        },

{'F'  ,  "111100011"     ,9  },
{'g'  ,  "1111101000"  ,10},
{'6'  ,  "1111101001"  ,10},
{'J'  ,  "1111101010"  ,10},
{'q'  ,  "1111101011"  ,10},
{'-'  ,  "1100111000"  ,10},
{'.'  ,  "11001110010",11},
{HUFFMAN_END_CODE_SYMBOL,  "11001110011",11}
};


/***********************************************************************
@name  intToStr
*
@param in
str  -  copy  integer  into  passed  string
code  -  code  to  find  in  the  huffman  table
len  -  length  of  the  code
*
@description convert intger into string of bits.
**********************************************************************/ static  void  intToStr(char  *  str,
uint32_t  code,  size_t  len){
int  index  =  0;
//store  the  least  significant  bit  into  str while(index  <  len){
str[index]  =  ((code  >>  (len  -  index  -  1))  &  1)  +  '0'; index++;
}
str[index]  =  '\0';
}

/***********************************************************************
@name  getHuffmanSymbol
*
@param in
code  -  code  to  find  in  the  huffman  table
len  -  length  of  the  code
*
@description find if a code is present in the Huffman Table based on the
code  and  length  of  code  passed  in.
**********************************************************************/ static  uint32_t
getHuffmanSymbol(uint32_t  code,  size_t  len){
char  temp[20];

//iterate  over  huffman  table  to  find  if  the  code  is  available  or  not

for(int  i  =  0;  i  <  sizeof(decodehuffmanCode)/sizeof(decodeHuffmanCode_t); i++){


table

//if  len  is  equal  to  the  length  of  number  of  bits  in  huffman  encoded

if(len  ==  decodehuffmanCode[i].nBits){
//convert  integer  to  bit  string intToStr(temp,  code,  len);
//if  code  matches  with  huffman  encoded  code  return  symbol if  (strncmp(temp,
decodehuffmanCode[i].code,

decodehuffmanCode[i].nBits)  ==  0){
return  decodehuffmanCode[i].symbol;
}
}
}
return  INVALID_HUFFMAN_SYMBOL;
}

/************************************************************************
@name  getStats
*
@description get Huffman stats.
************************************************************************/ void getStats(){
printf("Number  of  received  bytes:  %d\r\n",  countBytes); printf("Uncompressed  message  length
in  bytes:  %d\r\n",  countDecodedBytes); printf("Percentage  compression:  %0.2f\r\n",
((countBytes*100)/(float)countDecodedBytes));
}

/************************************************************************
@name  setTTYFlags
*
@param in
tty  -  pointer  to  structure  termios
*
@description intialize the termios structure for UART reception.
************************************************************************/ static  void  setTTYFlags(struct
termios*  tty){
//clear parity bit, disabling parity (most common) tty->c_cflag  &=  ~PARENB;
//clear stop field, only one stop bit used in communication (most common)
tty->c_cflag  &=  ~CSTOPB;
//clear  all  bits  that  set  the  data  size tty->c_cflag  &=  ~CSIZE;

//8 bits per byte (most common) tty->c_cflag  |=  CS8;
//disable  RTS/CTS  hardware  flow  control  (most common)
tty->c_cflag  &=  ~CRTSCTS;
//turn  on  READ  &  ignore  ctrl  lines  (CLOCAL  =
1)
tty->c_cflag  |=  CREAD  |  CLOCAL;
//clear  ICANON  flag
tty->c_lflag  &=  ~ICANON;
//disable echo
tty->c_lflag  &=  ~ECHO;
//disable erasure
tty->c_lflag  &=  ~ECHOE;
//disable  new-line  echo tty->c_lflag  &=  ~ECHONL;
//disable interpretation of INTR, QUIT and SUSP tty->c_lflag  &=  ~ISIG;
//turn  off  s/w  flow  ctrl
tty->c_iflag  &=  ~(IXON  |  IXOFF  |  IXANY);
//disable any special handling of received bytes
tty->c_iflag  &=  ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL);
//prevent special interpretation of output bytes (e.g. newline chars) tty->c_oflag  &=  ~OPOST;
//prevent conversion of newline to carriage return/line
feed
tty->c_oflag  &=  ~ONLCR;
//wait for up to 1s (5 deciseconds), returning as soon as any data is received.
tty->c_cc[VTIME]  =  5;
//set vmin to 0
tty->c_cc[VMIN]  =  0;
```

```
}

/************************************************************************
@name  configureSerialPort
*
@description Refer read_serial.h for more details.
***********************************************************************/ int  configureSerialPort(struct
termios*  tty){
//  Open  the  serial  port)
int  serialPort  =  open(SERIAL_PORT_KL25Z,  O_RDWR);

//  Read  in  existing  settings,  and  handle  any  error if(tcgetattr(serialPort,  tty)  !=  0)  {

printf("Error  %i  from  tcgetattr:  %s\r\n",  errno,  strerror(errno)); return  -1;
}

//set tty flags setTTYFlags(tty);

// Set in/out baud rate cfsetispeed(tty,  BAUD_RATE); cfsetospeed(tty,  BAUD_RATE);

// Save tty settings, also checking for error if  (tcsetattr(serialPort,  TCSANOW,  tty)  !=  0)  {
printf("Error  %i  from  tcsetattr:  %s\n",  errno,  strerror(errno)); return  -1;
}
return  serialPort;
}

/************************************************************************
@name  readSerialPortData
*
@param in
serialPort  -  open  the  serial  port  and  assign  it  to  serialPort
data  -  data  to  be  read  from  the  serial  port
dataCapacity  -  length  of  data  array
*
@description read data from serial port and store it in data.
***********************************************************************/ static  int  readSerialPortData(int
serialPort,  uint8_t*  data,  int  dataCapacity){
//making all entries 0 memset(data,  0,  dataCapacity);

//read  call  on  serial  port
int  nBytes  =  read(serialPort,  data,  dataCapacity);

//if  rx  bytes  <  0,  means  error  occured. if  (nBytes  <=  0)  {
printf("Error  reading:  %s",  strerror(errno)); return  0;
}
return  nBytes;
}

/************************************************************************

@name decodeMessages
*
@description Refer read_serial.h for more details.
***********************************************************************/ void  decodeMessages(int
serialPort,  char*  msg,  int  msgLen){
```

```
//temporary buffer to read serial data uint8_t  readBuf[RECEIVED_MESSAGE_LENGTH];
uint32_t getCharFromHuffmanCode = INVALID_HUFFMAN_SYMBOL; static  uint32_t
codeValueToSearch  =  0;
// Allocate memory for read buffer, set size according to your needs static  int  bufIndex  =  0;
static  int  msgIndex  =  0;
static  uint8_t  bitsToConsiderInCurrByte  =  0;

bool breakLoop = false; bool  endDecoding  =  false; uint32_t  temp  =  0; while(1){
int  nBytes  =  readSerialPortData(serialPort,  readBuf,  sizeof(readBuf)); if(nBytes  ==  0){
return;
}

//increment  the  countBytes  to  keep  track  of  received  bytes countBytes  +=  nBytes;
//initialize  buffer  index  =  0 bufIndex  =  0;

//iterate  over  the  received  data,  take  a  chunk  and  decode  it while(nBytes > 0){
//codeValueToSearch:  total  number  out  of  it  a  part  of  it  will  be decoded using Huffman Table
codeValueToSearch  =  (codeValueToSearch  <<  BITS_IN_A_BYTE)  | readBuf[bufIndex];
//total  number  of  bits  in  picture  =  number  of  bits  in codeValueToSearch
bitsToConsiderInCurrByte  =  bitsToConsiderInCurrByte  +  BITS_IN_A_BYTE; while(1){
//pick  encoded  bits  of  size  [HUFFMAN_MIN_LENGTH  -  i]  and  find symbol in the Huffman
Table
for(int  i  =  HUFFMAN_MIN_LENGTH;  i  <=  HUFFMAN_MAX_LENGTH;  i++){
//if  length  of  encoded  bits  (which  is  equal  to (bitsToConsiderInCurrByte  -  i))  is  less  than  0
if((bitsToConsiderInCurrByte - i) < 0){
//break the loop and proceed to next step

breakLoop = true; break;
}
//temp:  encoded  value  to  search  in  huffman  table
temp  =  codeValueToSearch  >>  (bitsToConsiderInCurrByte  -  i);
//get  the  symbol  if  temp  is  present  in  the  huffman  table getCharFromHuffmanCode  =
getHuffmanSymbol(temp,  i);



end of message

//if  the  found  symbol  is  HUFFMAN_END_CODE_SYMBOL,  means  the

if(getCharFromHuffmanCode  ==  HUFFMAN_END_CODE_SYMBOL){
//store  the  decoded  character  in  the  msg  buffer msg[msgIndex]  =  getCharFromHuffmanCode;
msgIndex++;


countDecodedBytes  +=  strlen(msg);

//reset the variables, to decode next message msgIndex  =  0;
nBytes  =  0;
codeValueToSearch  =  0;
bitsToConsiderInCurrByte  =  0;
//break all loops and wait for next message endDecoding = true;
return;
}
else  if(getCharFromHuffmanCode  !=  INVALID_HUFFMAN_SYMBOL){
```

```
//store the decoded character in the msg buffer msg[msgIndex] = getCharFromHuffmanCode;
msgIndex++;
//remove the found bits and rearrange the data codeValueToSearch = codeValueToSearch &
MASK((bitsToConsiderInCurrByte - i));
//update the number of bits to consider in the next

iteration


bitsToConsiderInCurrByte = bitsToConsiderInCurrByte - i; break;
}
}

//if bits left to consider is still greater than HUFFMAN_MIN_LENGTH, keep decoding
otherwise break
if((bitsToConsiderInCurrByte < HUFFMAN_MIN_LENGTH) ||

(breakLoop)){


bufIndex++; nBytes--;

breakLoop = false; break;
}
//stop decoding if(endDecoding){
break;
}
}
//stop decoding if(endDecoding){
endDecoding = false; break;
}
}
}
}


/******************************************************************************
**
@file read_serial.h
******************************************************************************
**
* Date: 04-30-2022
Author:Ayush Gupta
Description: This file deals with the reception and decoding of huffman encoded
serial data.
******************************************************************************
*/

//include header files #include <stdio.h> #include <string.h> #include <stdbool.h> #include
<stdint.h>

//include Linux headers
#include <fcntl.h>     // Contains file controls like O_RDWR   #include <errno.h>     // Error
integer and strerror() function #include <termios.h>       // Contains POSIX terminal control
definitions #include <unistd.h>        // write(), read(), close() functions
```

```
//serial port on UBUNTU system
#define  SERIAL_PORT_KL25Z       ("/dev/ttyS8")

#define  HUFFMAN_END_CODE_SYMBOL'\0'
#define BAUD_RATE B115200
#define  HUFFMAN_MIN_LENGTH  (2)
#define HUFFMAN_MAX_LENGTH  (11)
#define  RECEIVED_MESSAGE_LENGTH  (100)

/***********************************************************************
@name  configureSerialPort
*
@param in
serialPort  -  open  the  serial  port  and  assign  it  to  serialPort
tty  -  pointer  to  structure  termios
*
@description configure the Serial Port, open the port, set tty flags,
and  set  the  baud  rate
***********************************************************************/ int  configureSerialPort(struct
termios*  tty);


/***********************************************************************
@name decodeMessages
*
@param in
serialPort  -  open  the  serial  port  and  assign  it  to  serialPort
msg  -  message  string
msgLen - message length
*
@description decode the received message from serial port.
***********************************************************************/ void  decodeMessages(int
serialPort,  char*  msg,  int  msgLen);

/***********************************************************************
@name  getStats
*
@description get Huffman stats.
***********************************************************************/ void getStats();
################################################################################
###    ##
# @file huffman.py
################################################################################
###    ##
# Date:04-30-2022
#  Author:       Ayush  Gupta

#  Description:  This  file  convert  the  given  list  of  strings  in  their  characters and
#         their  respective  encoded  Huffman  codes.
#  Refernce:    https://www.programiz.com/dsa/huffman-coding
################################################################################
###    ##

from  collections  import  Counter endodedHuffmanFile = "encodeHuffman.txt"
```

```
dedodedHuffmanFile = "decodeHuffman.txt"

class  NodeTree(object):
def     init   (self,  left=None,  right=None): self.left  =  left
self.right = right

def  children(self):
return  self.left,  self.right

def str (self):
return  self.left,  self.right


def  huffman_code_tree(node,  binString=''): '''
Function to find Huffman Code '''
if type(node) is str: return  {node:  binString}
(l,  r)  =  node.children() d = dict()
d.update(huffman_code_tree(l,  binString  +  '0')) d.update(huffman_code_tree(r,  binString  +  '1'))
return d


def make_tree(nodes): '''
Function to make tree
:param nodes: Nodes
:return: Root of the tree '''
while  len(nodes)  >  1: (key1,  c1)  =  nodes[-1]
(key2,  c2)  =  nodes[-2]

nodes  =  nodes[:-2]
node  =  NodeTree(key1,  key2) nodes.append((node,  c1  +  c2))
nodes  =  sorted(nodes,  key=lambda  x:  x[1],  reverse=True) return  nodes[0][0]


if     name       ==  '   main   ':

#read  the  cprpus  and  add  a  few  \0  lines  to  complete  the  collection f  =  open("corpus.txt",
"r")
corpus  =  f.readlines() corpus.append("\0\r\n") corpus.append("\0\r\n") corpus.append("\0\r\n")
corpus.append("\0\r\n") corpus.append("\0\r\n") string = "".join(corpus)

#close the file f.close()

#initialize new lists for both encoded and decoded huffman tables encodeHuffmanTable = []
decodeHuffmanTable  = []

#get the frequency of each symbol freq  =  dict(Counter(string))
#sort the dictionary on the basis of frequency
freq  =  sorted(freq.items(),  key=lambda  x:  x[1],  reverse=True)

#build huffman tree on the basis of frequencies node  =  make_tree(freq)
#traverse each node and store the path in encoding list encoding  =  huffman_code_tree(node)
for i in encoding:
#append  data  in  the  encoded  and  decoded  lists encodeHuffmanTable.append("{'"  +  f'{i}'  +
"',0b"  +  f'{encoding[i]}'  +  ","
+  f'{len(encoding[i])}'  +  "},\n")
```

```
decodeHuffmanTable.append("{'" + f'{i}' + "'," + "'"+ f'{encoding[i]}' +
"'" +"," + f'{len(encoding[i])}' + "},\n")

# store the encoded huffman in encodeHuffman.txt fe = open(endodedHuffmanFile, "w")
fe.writelines(encodeHuffmanTable)
```

```
ECEN 5813 - Principles of Embedded Software Final Project\r\n
Clock initialized\r\n Uart initialized\r\n
\0\r\n
ECEN5813 Principles of Embedded Software Final Project\r\n
author\r\n Ayush Gupta\r\n
Author\r\n Ayush Gupta\r\n
hexdump 0 128\r\n
Invalid Command: hexdump\r\n
dump 0 64\r\n
\r\n
0000_0000  00 30 00 20 D5 00 00 00 43 01 00 00 89 09 00 00\r\n
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00\r\n
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00\r\n
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00\r\n
dump 0 128\r\n
I 00 30 00 20 D5 00 00 00 43 01 00 00 89 09 00 00\r\n
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00\r\n
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00\r\n
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00\r\n
dump 0 128\r\n I>\r\n

>\r\n
dump 0 64\r\n
\r\n
0000_0000 00 30 00 20 D5 00 00 00 43 01 00 00 89 09 00 00\r\n
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00\r\r\n\n
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00\r\r\n\n
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00\r\r\n\n
print\r\n
Invalid Command: print\r\n
printz\r\n
Invalid Command: printz\r\n
info\r\n
ERR: BUILD_COMMIT is not defined\r\n ERR: HOSTNAME is not defined\r\n ERR: DATE_TIME is
not defined\r\n
Version 1.0 built on HOSTNAME at DATE_TIME Commit BUILD_COMMIT\r\n
abcdefghijklmnopqrstuvwxyz\r\n
Invalid Command: a,bcdefghijklmnopqrstuvwxyz\r\n
>\r\n
ABCDEFGHIJKLMNOPQUVWXYZ\r\n
Invalid Command: ABCDEFGHIJKLMNOPQRSTUVWX,YZ\r\n 1234567890
!@#$%^&*()[]
```

```
/*******************************************************************************
****
@file  huffman.h

********************************************************************************
****
```

```
* Date:         04-27-2022
Author:Ayush Gupta
Description: PES - Final Project. This file handles all the operations related to
huffman.
It is to be used for ECEN 5813 "Principles of Embedded Software"

Course Assignment.

*********************************************************************************
***/

#ifndef SOURCE_HUFFMAN_H_
#define  SOURCE_HUFFMAN_H_

#include <stdint.h>

//HUFFMAN encoded minimum & maximum lengths
#define HUFFMAN_MIN_LENGTH (2)
#define HUFFMAN_MAX_LENGTH (11)
#define HUFFMAN_END_CODE_SYMBOL('\0')

/***********************************************************************
@name encodeHuffman
*
@param in
msg: message string
buffer: buffer to store encoded message
nBytes: number of bytes to encode
*
@description encode the msg string passed to the function using HUFFMAN
CODING algorithm. returns the length of huffman encoded
message.
***********************************************************************/
int encodeHuffman(const char* msg, uint8_t* buffer, size_t nBytes);


/***********************************************************************
@name decodeHuffman
*
@param in
msg: message string to store the decoded bytes
buffer: encoded message buffer
nBytes: number of bytes to decode
*
@description Refer huffman.h for more information.
***********************************************************************/
void decodeHuffman(uint8_t* buffer, size_t nBytes, char * msg);

#endif /* SOURCE_HUFFMAN_H_ */
/********************************************************************************
****
@file huffman.h

*********************************************************************************
****
```

```
* Date:          04-27-2022
Author:Ayush  Gupta
Description: PES - Final Project. This file handles all the operations related to
huffman.

It is to be used for ECEN 5813 "Principles of Embedded Software"
Course Assignment.

***********************************************************************************
***/

#ifndef SOURCE_HUFFMAN_H_
#define   SOURCE_HUFFMAN_H_

#include  <stdint.h>

//HUFFMAN  encoded  minimum  &  maximum  lengths
#define  HUFFMAN_MIN_LENGTH (2)
#define HUFFMAN_MAX_LENGTH (11)
#define  HUFFMAN_END_CODE_SYMBOL('\0')

/*************************************************************************
@name  encodeHuffman
*
@param  in
msg:  message  string
buffer: buffer to store encoded message
nBytes: number of bytes to encode
*
@description  encode  the  msg  string  passed  to  the  function  using  HUFFMAN
CODING  algorithm.  returns  the  length  of  huffman  encoded
message.
*************************************************************************/
int  encodeHuffman(const  char*  msg,  uint8_t*  buffer,  size_t  nBytes);


/*************************************************************************
@name  decodeHuffman
*
@param  in
msg:  message  string  to  store  the  decoded  bytes
buffer: encoded message buffer
nBytes:  number  of  bytes  to  decode
*
@description  Refer  huffman.h  for  more  information.
*************************************************************************/
void  decodeHuffman(uint8_t*  buffer,  size_t  nBytes,  char  *  msg);

#endif /* SOURCE_HUFFMAN_H_ */
/**********************************************************************************
****
@file  uart.c

***********************************************************************************
****
```

```
* Date:        03-21-2022
Author:Ayush  Gupta
Description: PES - Assignment 6. This file handles all the operations related to

UART.
It is to be used for ECEN 5813 "Principles of Embedded Software"
Course  Assignment.

**************************************************************************************
****/

#include  "MKL25Z4.h"


#include  <stdbool.h>
#include  <string.h>

#include  "uart.h" #include  "fifo.h" #include  "cli.h" #include  "huffman.h"
/*************************************************************************************
*****
UART  Configuration
*

**************************************************************************************
****/
#define UART_OVERSAMPLE_RATE


#define SYS_CLOCK

#define  BACKSPACE_ASCII_VALUE

(15)



(8)



(24e6)



#define  DATA_SIZE

#define  PARITY

#define        STOP_BITS


(8)
```

```
(None)

(1)

#if  (DATA_SIZE  ==  8)
#define  BIT_MODE

(0)

#if  (PARITY  ==  None)
#define  PARITY_ENABLE

(0)

#if  (STOP_BITS  ==  1)

#define  STOP_CONFIG

#elif  (STOP_BITS  ==  2)
#define  STOP_CONFIG

#endif

#define  TEMP_BUFFER_SIZE

(256)

(0)

(1)

//initialize  RX,  Tx  buffer  with  the  default  parameters cbfifo_t  rxBuffer,  txBuffer;
```

```
/*******************************************************************************
*****
@name:  uart0Init
*
@description:  Refer  uart.h  for  more  detail.

********************************************************************************
****/
void  uart0Init(uint32_t  baudRate){ uint16_t sbr;

//Enable clock gating for UART00 and PORTA SIM->SCGC4  |=  SIM_SCGC4_UART0_MASK;
SIM->SCGC5  |=  SIM_SCGC5_PORTA_MASK;

//Disabling transmitter and receiver before init UART0->C2  &=  ~UART0_C2_TE_MASK  &
~UART0_C2_RE_MASK;

//Set UART clock to 24MHz clock
SIM->SOPT2  |=  SIM_SOPT2_UART0SRC(1);

//Set pins to UART0 Tx and Rx
PORTA->PCR[1]  =  PORT_PCR_ISF_MASK  |  PORT_PCR_MUX(2);     //Rx PORTA->PCR[2]  =
PORT_PCR_ISF_MASK  |  PORT_PCR_MUX(2);   //Tx

//Set  baud  rate  and  oversampling  ratio
sbr  =  (uint16_t)((SYS_CLOCK)/(baudRate  *  UART_OVERSAMPLE_RATE)); UART0->BDH  &=
~UART_BDH_SBR_MASK;
UART0->BDH  |=  UART0_BDH_SBR(sbr  >>  8); UART0->BDL  =  UART0_BDL_SBR(sbr);
UART0->C4  |=  UART0_C4_OSR(UART_OVERSAMPLE_RATE  -  1);



bit

// Disable interrupts for RX active edge and LIN break detect, select two stop

UART0->BDH  |=  UART0_BDH_RXEDGIE(0)  |  UART0_BDH_SBNS(STOP_CONFIG)  |

UART_BDH_LBKDIE(0);

//Don't  enable  loopback  mode,  use  8  data  bit  mode,  don't  use  parity UART0->C1  =
UART0_C1_LOOPS(0)  |  UART0_C1_M(BIT_MODE)  |
UART0_C1_PE(PARITY_ENABLE) | UART0_C1_PT(0);
//Don't invert transit data, do enable interrupt for errors
UART0->C3  =  UART0_C3_TXINV(0)  |  UART0_C3_ORIE(0)  |  UART0_C3_NEIE(0)

|  UART0_C3_FEIE(0)  |  UART0_C3_PEIE(0);

//Clear error flags
UART0->S1  =  UART0_S1_OR(1)  |  UART0_S1_NF(1)  |  UART0_S1_FE(1)  |
UART0_S1_PF(1);

//Configure  to  send  LSB  first
UART0->S2  =  UART0_S2_MSBF(0)  |  UART0_S2_RXINV(0);
//Initializing Queue for further use fifoInit(&txBuffer); fifoInit(&rxBuffer);

//Enable UART0 interrupts NVIC_SetPriority(UART0_IRQn,  2);
NVIC_ClearPendingIRQ(UART0_IRQn); NVIC_EnableIRQ(UART0_IRQn);

//Enable  receiver  interrupts UART0->C2  |=  UART_C2_RIE(1);

//Enable UART transmitter and receiver
UART0->C2  |=  UART0_C2_TE(1)  |  UART0_C2_RE(1);
}


/*************************************************************************************
*****
@name:     sys_write
*
@description:  Refer  uart.h  for  more  detail.

*************************************************************************************
****/
int     sys_write(int  handle,  char  *buf,  int  size){
if(buf  ==  NULL){
//returns  an  error  when  buf  is  pointing  to  NULL
return  size;
}

uint8_t  data[512];

int  nBytesStored  =  0;
//wait if the TX Buffer is full.
while(txBuffer.bufferFullFlag){};


//  encode  data  using  HUFFMAN  coding  algorithm
int  bytesToStore  =  encodeHuffman(buf,  data,  size);
//store data in TX Buffer
nBytesStored = cbfifo_enqueue(data, bytesToStore, &txBuffer);


if(!(UART0->C2  &  UART0_C2_TIE_MASK)){ UART0->C2  |=  UART0_C2_TIE(1);
}

if(nBytesStored  <  bytesToStore){
return -1;
}
return 0;
}

/*************************************************************************************
*****
@name:     sys_readc
*
@description:  Refer  uart.h  for  more  detail.

*************************************************************************************
```

```
****/
int     sys_readc(void){
char ch;
//check  if  RX  buffer  is  empty
if  (QIsEmpty(&rxBuffer)  ==  0){ cbfifo_dequeue(&ch, 1, &rxBuffer); return  ch;
}
return -1;
}

/*********************************************************************************
*****
@name:  serialIn
*
@description:  Refer  uart.h  for  more  detail.

**********************************************************************************
****/
size_t  serialIn(void  *buf,  size_t  count){
return  cbfifo_dequeue(buf,  count,  &rxBuffer);
}


/*********************************************************************************
******
@name  putstr
*
@param  in
s  -  string  to  be  sent  on  serial  terminal.
*
@description  Function  displays  the  string  on  serial  terminal  and  returns  the number  of
characters  sent.
*
**********************************************************************************
**/
int putstr(char* s)
{
int  i  =  0;
while  (*s)  {    //  output  characters  until  NULL  found putchar(*s++);

i++;
}
return  i  +  1;
}


/*********************************************************************************
******
@name  getUserInput
*
@param  in
inputString  -  character  buffer  to  store  user  input.
*
@description  gets  the  user  input  from  serial  terminal  and  store  it  in  the  passed string.
*
**********************************************************************************
```

```
**/
void  getUserInput(char  *inputString){
char ch;
int i; i  =  0;
while(1){
ch  =  getchar(); putchar(ch);
if (ch == '\r')
break;  if  (ch  !=  8){
*(inputString  +  i)  =  ch; i++;
}
}
putchar('\r');
putchar('\n');

}


/*******************************************************************************
*****
@name:  userInputProcessing
*
@description:  Refer  uart.h  for  more  detail.

********************************************************************************
****/
void  userInputProcessing(void){
char  userInput[TEMP_BUFFER_SIZE];
char ch = '@'; int  i  =  0; while(1){
//wait  for  incoming  character while(QIsEmpty(&rxBuffer)){}; cbfifo_dequeue(&ch, 1, &rxBuffer);
if(!(UART0->C2  &  UART0_C2_TIE_MASK)){

UART0->C2  |=  UART0_C2_TIE(1);
}
if((ch == '\r') || (ch == '\n')){
break;
}

//handling backspace character
if  (ch  !=  BACKSPACE_ASCII_VALUE){
userInput[i] = ch; i++;

}
else{



if(i  !=  0){
i--;

}
}
}
userInput[i] = '\0'; processCommand(userInput); memset(userInput,  0,  TEMP_BUFFER_SIZE);
i = 0;
return;
}



/*****************************************************************************************
*****
@name:  UART0_IRQHandler
*
@description:  UART0  IRQ  Handler.

******************************************************************************************
****/
void  UART0_IRQHandler(void)  { uint8_t ch;
if  (UART0->S1  &  UART0_S1_RDRF_MASK)  { ch  =  UART0->D;
if  (!(rxBuffer.bufferFullFlag))  {
//if receive buffer is not full cbfifo_enqueue(&ch, 1, &rxBuffer); if(!(UART0->C2  &
UART0_C2_TIE_MASK)){
UART0->C2  |=  UART0_C2_TIE(1);
}
}
}

if((UART0->C2  &  UART0_C2_TIE_MASK)  &&  (UART0->C2  &  UART0_S1_TDRE_MASK)){
//transmit  the  character  back  on  the  terminal,  if  txBuffer  is  not  empty
if(cbfifo_dequeue(&ch,  1,  &txBuffer)  ==  1){ UART0->D  =  ch;
if(!(UART0->C2  &  UART0_C2_TIE_MASK)){ UART0->C2  |=  UART0_C2_TIE(1);
}
}

else{


}
}
}


//TX  buffer  is  empty,  disable  TX  interrupts UART0->C2  &=  ~UART0_C2_TIE_MASK;


/************************************************************************************
@file  fifo.h
************************************************************************************
*
*
*  Date:          03-21-2022
Author:Ayush  Gupta
Description:  PES  -  Assignment  6.  This  file  handles  all  the  operations  related  to
Circular Buffer.
It  is  to  be  used  for  ECEN  5813  "Principles  of  Embedded  Software"
Course  Assignment.
************************************************************************************/
#ifndef  SOURCE_FIFO_H_
#define  SOURCE_FIFO_H_
```

```
#include <stdio.h> #include <stdint.h> #include <stdbool.h>

#define FIFO_CAPACITY    (256)

/******************************************************************************
Structure has the metadata for circular buffer.
It has an array to store the data, rpt & wptr to track the position
for enqueue and dequeue, nElements to keep track of number of elements
enqueued and bufferFullFlag boolean to track the buffer full condition.
* ****************************************************************************/
typedef struct cbfifo{
uint8_t data[FIFO_CAPACITY]; uint16_t rptr;
uint16_t wptr; uint16_t nElements;
volatile bool bufferFullFlag;
} cbfifo_t;

/******************************************************************************
@name fifoInit
*
@param in
fifo: ptr to fifo in question
*
@description: fifo initialization.
*****************************************************************************/
void fifoInit(cbfifo_t* fifo);

/******************************************************************************
@name isEmpty
*

@param in
cbfifo - fifo in question
*
@description - returns a 0 if the fifo is not empty otherwise returns -1.
*****************************************************************************/
int QIsEmpty(cbfifo_t* cbfifo);

/*****************************************************************************
Enqueues data onto the FIFO, up to the limit of the available FIFO
capacity.
*
Parameters:
buf     Pointer to the data
nbyte   Max number of bytes to enqueue
*
Returns:
The number of bytes actually enqueued, which could be 0. In case
of an error, returns (size_t) -1.
*****************************************************************************/ size_t cbfifo_enqueue(void
*buf, size_t nbyte, cbfifo_t* cbfifo);


/*****************************************************************************
Attempts to remove ("dequeue") up to nbyte bytes of data from the
FIFO. Removed data will be copied into the buffer pointed to by buf.
```

```
*
Parameters:
buf      Destination for the dequeued data
nbyte  Bytes of data requested
*
Returns:
The number of bytes actually copied, which will be between 0 and
nbyte.
*
To further explain the behavior: If the FIFO's current length is 24
bytes, and the caller requests 30 bytes, cbfifo_dequeue should
return the 24 bytes it has, and the new FIFO length will be 0. If
the FIFO is empty (current length is 0 bytes), a request to dequeue
any number of bytes will result in a return of 0 from
cbfifo_dequeue.
*****************************************************************************/ size_t cbfifo_dequeue(void
*buf, size_t nbyte, cbfifo_t* cbfifo);


/*****************************************************************************
Returns the number of bytes currently on the FIFO.
*
Parameters:
none
*
Returns:
Number of bytes currently available to be dequeued from the FIFO
*****************************************************************************/ size_t cbfifo_length(cbfifo_t*
cbfifo);

#endif /* SOURCE_FIFO_H_ */
/******************************************************************************
@file fifo.c
******************************************************************************
*
*
* Date:          03-21-2022
Author:Ayush Gupta
Description: PES - Assignment 6. This file handles all the operations related to
Circular Buffer.
It is to be used for ECEN 5813 "Principles of Embedded Software"
Course Assignment.
******************************************************************************/

#include "fifo.h"
#include "MKL25Z4.h"

/***********************************************************
@name incrementPtr
*
@param ptr
interger which represents the write and read pointers
*
Returns ptr value between 0 to Buffer Maximum size by
incrementing the ptr value by 1 and roll over to 0 when
```

```
ptr value exceeds Maximum buffer size.
* **********************************************************/
static uint8_t incrementPtr(uint8_t ptr){
if ((ptr + 1) == FIFO_CAPACITY)
return 0;
else
ptr++;
return ptr;
}

/***********************************************************
@name fifoInit
*
@param in
fifo: ptr to fifo in question
*
Refer fifo.h for more details.
* **********************************************************/
void fifoInit(cbfifo_t* fifo){ fifo->nElements = 0;
fifo->rptr = 0;
fifo->wptr = 0;
fifo->bufferFullFlag = false;
for(int i = 0; i<FIFO_CAPACITY; i++){ (fifo->data)[i] = 0;
}

}

/***********************************************************
@name QIsEmpty
*
@param in
fifo: ptr to fifo in question
*
Check if the buffer is empty or not based on the value
of wptr, rptr and bufferFullFlag. Returns a -1 if the buffer
is empty and 0 otherwise.
* **********************************************************/
int QIsEmpty(cbfifo_t* cbfifo){
if ((cbfifo->wptr == cbfifo->rptr) && (cbfifo->bufferFullFlag == 0))
return -1;
else
return 0;
}

/***********************************************************
@name QWriteByte
*
@param value
data to be stored in the FIFO - one byte
*
@param cbfifo
ptr to the FIFO.
*
Check if the buffer is full or not, and if it not full add
the new element in the queue and return the number of elements
```

```
added.
* **********************************************************/
int QWriteByte(uint8_t value, cbfifo_t* cbfifo){
if (cbfifo == NULL){
return -1;
}

//check if buffer is full or not
if (!(cbfifo->bufferFullFlag)){
//if not full add the value in the buffer and return success as status code and increment
wptr
cbfifo->data[cbfifo->wptr] = value; NVIC_DisableIRQ(UART0_IRQn);
cbfifo->wptr = incrementPtr(cbfifo->wptr); NVIC_EnableIRQ(UART0_IRQn);
cbfifo->nElements++;


}
else{
//printf("Buffer is full\r\n");
while(!cbfifo->bufferFullFlag){};
//if not full add the value in the buffer and return success as status code and increment
wptr
cbfifo->data[cbfifo->wptr] = value; NVIC_DisableIRQ(UART0_IRQn);
cbfifo->wptr = incrementPtr(cbfifo->wptr);

NVIC_EnableIRQ(UART0_IRQn);
cbfifo->nElements++;
//        return -1;
}

//check if the buffer is full and if it is full assign the bufferFullFlag as true
if((cbfifo->wptr == cbfifo->rptr) && (cbfifo->rptr != 0 && cbfifo->wptr!=0) ) {
NVIC_DisableIRQ(UART0_IRQn);
cbfifo->bufferFullFlag = true; NVIC_EnableIRQ(UART0_IRQn);
}
return 0;
}

/**********************************************************
@name QReadByte
*
@param cbfifo
ptr to the FIFO.
*
Check if the buffer is empty or not, and if it not empty,
remove the element from the FIFO and return its value.
* **********************************************************/ uint8_t QReadByte(cbfifo_t* cbfifo){
uint8_t value;
//if not empty return the value pointed by rptr and increment it


NVIC_DisableIRQ(UART0_IRQn);
value = cbfifo->data[cbfifo->rptr]; cbfifo->rptr = incrementPtr(cbfifo->rptr); cbfifo->bufferFullFlag
= false; NVIC_EnableIRQ(UART0_IRQn);
cbfifo->nElements--;
```

```
return value;
}

/*******************************************************************************
***
Dequeue an element from Circular Buffer. Returns the number of bytes that's
been removed from the buffer.
*
See fifo.h for more detail.
*
********************************************************************************
/
size_t cbfifo_dequeue(void *buf, size_t nbyte, cbfifo_t* cbfifo){

int nCopiedElements = 0;
//check if buf points to the valid position in the memory
if (buf == NULL){
return (size_t) -1;
}

//check if nbyte is more than 0

if (nbyte <= 0){
return (size_t)0;
}

while(nbyte > 0){
//check if buffer is empty or not
if (QIsEmpty(cbfifo) != 0){
//printf("FIFO is empty\n");
break;
}
else{
*((uint8_t *)buf + nCopiedElements) = QReadByte(cbfifo); nCopiedElements++;
}
nbyte--;
}
return (size_t)nCopiedElements;
}

/*******************************************************************************
***
Enqueue an element from Circular Buffer. Returns the number of bytes that's
been added to the buffer.
*
See cbfifo.h for more detail.
*
********************************************************************************
/
size_t cbfifo_enqueue(void *buf, size_t nbyte, cbfifo_t* cbfifo){
int nPushedBytes = 0;
uint8_t* temp = (uint8_t*)buf;
//check if buf points to the valid position in the memory
if (buf == NULL){
return (size_t) -1;
}

//check if nbyte is less than or equal to 0
if (nbyte <= 0){
return (size_t)0;
}

while(nbyte > 0){
if (QWriteByte(*(temp+nPushedBytes), cbfifo) == 0){ nPushedBytes++;
nbyte--;

}
else{

}
}



break;


return (size_t)nPushedBytes;
}

/*****************************************************
Returns the number of elements in the Circular Buffer
*
See cbfifo.h for more detail.
* **************************************************/ size_t cbfifo_length(cbfifo_t* cbfifo){
return (size_t)cbfifo->nElements;
}


/******************************************************************************
****
@file hexdump.c

*******************************************************************************
****
* Date:         03-21-2022
Author:Ayush Gupta
Description: PES - Assignment 6. This file handles all the operations related hex dump.
It is to be used for ECEN 5813 "Principles of Embedded Software"
Course Assignment.

*******************************************************************************
***/
#include "hexdump.h" #include <string.h> #include <stdio.h>

#define BYTES_ON_A_SINGLE_LINE
(16)
#define MAX_HEXDEMP_LIMIT_IN_BYTES
(640)
#define HEX_DUMP_LENGTH
```

(496)

```
/*****************************************************************************
******
@name  getNibble
*
@param  in
nibble  -  nibble  to  be  displayed  on  Serial  terminal.
*
@description  Function  returns  the  nibble  in  hex.
*
*****************************************************************************
**/
static  char  getNibble(uint8_t  nibble){
char ch;
if  ((nibble  <=  9)){
ch  =  (nibble  +  '0');
}
else{

ch  =  (nibble  -  10  +  'A');
}
return  ch;
}

char  hexString[HEX_DUMP_LENGTH];
/*****************************************************************************
******
@name  displayHexDump
*
@param  in
buf  -  buffer  to  be  displayed  on  serial  terminal.
nBytes  -  number  of  bytes  requested.
*
@description  Refer  hexdump.h  for  more  details.
*
*****************************************************************************
**/
void  displayHexDump(const  void  *buf,  size_t  nBytes){
//clear the buffer

memset(hexString,  0,  HEX_DUMP_LENGTH);
int  hexIndex  =  0;
if(nBytes  >  MAX_HEXDEMP_LIMIT_IN_BYTES){
//requested memory is outside the limit; print till MAX_HEXDEMP_LIMIT_IN_BYTES and ignore
the rest for now.
nBytes  =  MAX_HEXDEMP_LIMIT_IN_BYTES;
}

int  charsOnALine  =  0;
uint8_t  *startingLoc  =  (uint8_t  *)buf; uint8_t  *endLoc  =  (uint8_t  *)buf  +  nBytes;
while(startingLoc  <  endLoc){
//break  if  hexdump  limit  reached
if(hexIndex  >=  HEX_DUMP_LENGTH){
break;
}
if((charsOnALine  ==  BYTES_ON_A_SINGLE_LINE)  ||  (charsOnALine  ==  0)){
//move cursor to the next line hexString[hexIndex++] = '\r'; hexString[hexIndex++] = '\n';




0xF0000000) >> 28);

0x0F000000) >> 24);

0x00F00000) >> 20);

0x000F0000) >> 16);

//print  starting  location  of  the  new  line  in  the  format  0000_0000 hexString[hexIndex++]  =
getNibble(((uint32_t)(startingLoc)  &

hexString[hexIndex++]  =  getNibble(((uint32_t)(startingLoc)  & hexString[hexIndex++]  =
getNibble(((uint32_t)(startingLoc)  & hexString[hexIndex++]  =  getNibble(((uint32_t)(startingLoc)  &
hexString[hexIndex++] = '_';


0x0000F000) >> 12);

0x00000F00) >> 8);

0x000000F0) >> 4);

0x0000000F));

hexString[hexIndex++]  =  getNibble(((uint32_t)(startingLoc)  & hexString[hexIndex++]  =
getNibble(((uint32_t)(startingLoc)  & hexString[hexIndex++]  =  getNibble(((uint32_t)(startingLoc)  &
hexString[hexIndex++]  =  getNibble(((uint32_t)(startingLoc)  &


//2 spaces in between address and data hexString[hexIndex++] = ' '; hexString[hexIndex++] = ' ';
charsOnALine = 0;

}



4));





}
```

```
hexString[hexIndex++]  =  getNibble((((uint8_t)(*startingLoc)  &  0xF0)  >>

hexString[hexIndex++]  =  getNibble(((uint8_t)(*startingLoc)  &  0x0F)); hexString[hexIndex++] = ' ';

startingLoc++; charsOnALine++;

//move cursor to the next line hexString[hexIndex++]  =  '\r'; hexString[hexIndex++] = '\n';
//string end character hexString[hexIndex++] = '\0';

//print the remaining bytes printf("%s",  hexString); hexIndex = 0;


}

/*
main.c  -  application  entry  point
*
Author  Howdy  Pierce,  howdy.pierce@colorado.edu
*/
#include  <stdio.h> #include  "sysclock.h" #include  "uart.h" #include  "test_cbfifo.h" #include
"test_huffman.h"



int main(void)
{
//clock initialization given sysclock_init();

//uart  initialization  call uart0Init(BAUD_RATE);

#ifdef DEBUG
//test  the  Huffman  implementation testHuffman();
//test  the  FIFO  implementation testFIFO();
#endif

printf("ECEN5813  Principles  of  Embedded  Software  Final  Project\r\n");
//  enter  infinite  loop
while (1) {
userInputProcessing();
}

return 0 ;
}
/*********************************************************************************
****
@file  test_huffman.h

*********************************************************************************
****
*  Date:          04-27-2022
Author:Ayush  Gupta
Description:  PES  -  Final  Project.  This  file  has  the  test  cases  to  test  huffman algorithm
It  is  to  be  used  for  ECEN  5813  "Principles  of  Embedded  Software"
Course  project.
```

```
***********************************************************************************
***/

#ifndef   TEST_HUFFMAN_H_
#define  TEST_HUFFMAN_H_



/**********************************************************************************
*****
@testHuffman
*
@description This function tests the huffman algorithm implementation for above mentioned
test cases.
*

***********************************************************************************
***/
void  testHuffman();


/**********************************************************************************
*****

@testHuffman
*
@description send a list of strings on serial port to analyze the HUFFMAN coding.

***********************************************************************************
***/
void  analyzeHuffmanEncoding();

#endif /* TEST_HUFFMAN_H_ */
/**********************************************************************************
****
@file  test_huffman.c

***********************************************************************************
****
*  Date:          04-27-2022
Author:Ayush  Gupta
Description: PES - Final Project. This file has the test cases to test huffman algorithm
It is to be used for ECEN 5813 "Principles of Embedded Software"
Course  project.

***********************************************************************************
***/


#include  <stdio.h> #include  <assert.h> #include  <string.h> #include  <stddef.h> #include
"huffman.h"


#define NUMBER_OF_TEST_CASES (17)
#define BUFFER_LENGTH (40)
```

```
/***********************************************************************************
*****
This array has all the combinations possible to test the HUFFMAN coding algorithm.
*
Note: 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 not be decoded and lost.
*

************************************************************************************
***/
char huffmanInputs[NUMBER_OF_TEST_CASES][BUFFER_LENGTH+1] = {

"Final Project",        //normal string "Embedded Software",       //normal string "Principles of
Embedded",     //normal string
"eeeeeeeeeeee",      //string with one character repeated all its length "ECEN 5813",
        //alphanumeric string
"r",      //one character string
">",     //punctuation character
"ECEN>",      //combination of alphabets and punctuation "abcdefghijklmnop", //string with all
unique character
"Huffman Coding API for Final Project", //a normal debug message length "random_script
from my mind",       //normal string
"FRDM KL25Z Development board",        //normal string
"",       //blank string
"1234567891011121314151617", //numbers ":::::::::::::::::::::::::::::::::::", //punctualtion
"ahahhvdfbkr-ngrpgkfnjkd-lnfdjbf",    //alphanumeric string "KL25z -       a 32-bit micro-
controller"      //normal string
};


/***********************************************************************************
*****
@testHuffman
*
@description Refer to test_huffman.h for more details.

************************************************************************************
***/
void testHuffman(){
//define buffers to store the encoded message and de-compressed message uint8_t
encodedString[BUFFER_LENGTH+1];
char outputsString[BUFFER_LENGTH+1];

//iterate over the test cases to test the huffman algorithm
for(int i = 0; i<NUMBER_OF_TEST_CASES; i++){
//clear both the buffers memset(encodedString, 0, BUFFER_LENGTH+1);
memset(outputsString, 0 , BUFFER_LENGTH+1);

//encode the message
int count = encodeHuffman(huffmanInputs[i], encodedString,
strlen(huffmanInputs[i]));
//decode the message
decodeHuffman(encodedString, count, outputsString);




0);
}
}

//compare the input and output strings
assert(strncmp(outputsString, huffmanInputs[i], strlen(huffmanInputs[i])) ==


/*********************************************************************************
*****
@testHuffman

*
@description Refer to test_huffman.h for more details.

*********************************************************************************
***/
void analyzeHuffmanEncoding(){
for(int i = 0; i<NUMBER_OF_TEST_CASES; i++){ printf("%s\r\n", huffmanInputs[i]);
}
printf("Compare");
}
```