Project_2 Report

Jeet Baru; jeet.baru@colorado.edu; SID – 107189037
Sudeep Kulkarni; suku8854@colorado.edu; SID – 106996755

Githublink : https://github.com/JeetBaru/Project2

Introduction & Objective

In this project we firstly devise a circular buffer that has functions to add and remove data in the buffer. Also functions for knowing the status of the buffer are defined. We then write unit tests for our functions defined in Project1. We use the CMOCKA libraries and write a makefile to check if all tests are successful. We then configure the UART for the FRDM board and try using it using polling/blocking statements. We now implement the circular buffer as an intermediate stage between UART and terminal. We also implemented a Logger system that will act as a generic framework to display and access data to and from the FRDM board. We have made all our code architecture independent and have used compile time switches to use the same code for BBB and the host machine as well.

Description

Circular Buffer:

cirbuff.c Structure of appropriate members has been created (head, tail, buff, count, size). In CircBuf_size size is allocated to a structure (defined in cirbuff.h) pointer on the heap, followed by buffer initialization, where memory of size uint8_t is being allocated to a pointer buff on the heap and head, tail are initially made to point to the first position, buff always points to first position in the circular buffer. is_buffer_empty and is_buffer_full will check whether the buffer is empty of full by comparing structure member count and size. If count zero buffer is empty, and if count is equal to size buffer is full. The size given to the buffer on the heap is destroyed by free(ptr). add_item, an item is added in the circular buffer by first calling initialize_buffer and then adding the item by dereferencing the head pointer and then incrementing the head pointer, this function takes care of wrap_around_condtion on itself. remove_item the data is first read at the tail position, then the tail is incremented by one place. This function also takes care of the wrap_around_condition. CircBuf_peak: This function gives value of the data at required index in the circular
buffer.

UNIT TESTS:

test_data.c -

- Unit tests for Invalid pointer, functionality big_to_little32 and little_to_big32 are included in this file. (stdlib, stdio, stdarg, stddef, setjmp, cmocka libraries are included in test_data.c). data.h and uni.h are included in the in the test_data.h.
- All the unit test functions call appropriate function which are defined in the .h files. For testing cmocka unit test framework is used.

Total Tests : 4

Test_memory.c

- memmove -
  Unit test functions for Invalid pointer, No overlap, Source in Destination, Destination in source is written and the output of "make test" will give PASS or FAIL for each of the unit tests by checking the functionality of the code and using cmocka unit test framework.
- memset -
  Unit test functions for Invalid pointer and Check_set are included in this file. The output of the command "make test" will give PASS or FAIL for each of the unit tests by checking the functionality of the code and using cmocka unit test framework.
- memzero -
  Unit test functions for Invalid pointer and Check_set are included in this file. The output of the command "make test" will give PASS or FAIL for each of the unit tests by checking the functionality of the code and using cmocka unit test framework.
- my_rev -
  Unit test functions for Invalid pointer, Odd reverse, even reverse and character set reverse included in the file and the output of the command "make test" will give either PASS or FAIL, based on the functionality of the code and cmocka unit test framework.

Total Tests :12

Cirbuff.c

- Unit tests for Allocate-free, Invalid Pointer, Non-Initialized buffer, Buffer full, buffer empty, wrapadd, wrapremove, overfill, overempty and Add-Remove are included in the file. The output of the command "make test" will either give PASS or FAIL depending upon the functionality of the code and cmocka unit test framework.

Total - 10 tests (stdlib, stdio, stdarg, stddef, setjmp, cmocka libraries included)

Test.c

- It has the conglomeration of all the unit test functions described above, when prompted a "make test" command it will give PASS or FAIL for each of the unit test functions included (total = 26). All the header files (cirbuff.h, data.h, memory.h uni.h) are included in this file.

Total - 26 tests (stdlib, stdio, stdarg, stddef, setjmp, cmocka libraries included)

Makefile

Changed the path of THIRD_PARTY_DIR to libraries. Then added the conditional loop which will select the source file on which the unit tests need to be performed. If TEST="data" then "make test" will run unit tests of test_data.c for TEST="memory" test_memory.c and if nothing is specified the same command will give all unit tests for all the test.c's. This make file also has log-host and log-bbb.

LOGGER APPLICATION:

The logger application has main use of interacting with devices that do not support standard input output functions like FRDM. For this we have written c programs stored in the folder named log. Description of each C file is given below

UART.c/h
In this C file we have written functions that help us to communicate to and from the device using UART. We have functions that can do this by both polling as well as interrupts. UART_init() is used to initialize the UART0 at PINS 1 and 2 of port A. Function sendbyte, receivebyte, sendnbyte use polling to transmit and receive data. send_n_bytes() receive_n_bytes() use interrupts to achieve the same objective. We have also written an IRQ handler to manage interrupts.

cirbuff.c
Refer to the previous explanation.

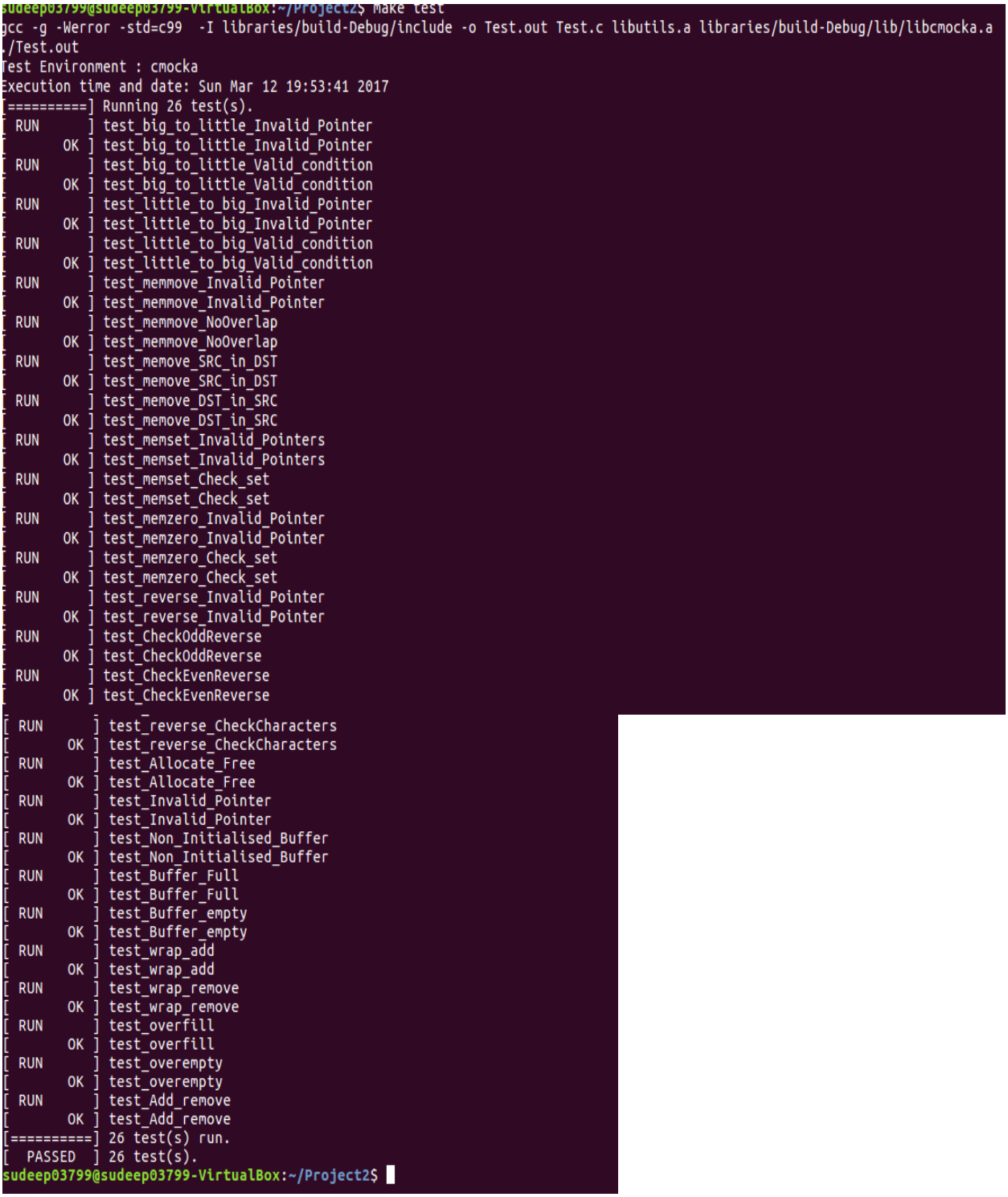
analysedata.c
This c file includes functions that perform important data calculations in order to obtain the functionality of identifying the input characters as alphabets, numbers, punctuations and misc. This is done by the function name analyse_data(). Another function defined in this c file is my_itoa(). Here data, base, destination address are passed as parameters and the function returns the starting address of the destination. This function converts the given data into an ascii string, while also converting the number into the specified base. The string stored at the destination address is null terminated.

Logger.c
This the most essential part of the whole application the functions in the c file are used to log different kinds of data strings, integer or just generic data. The function log_data(), takes a data pointer and length as parameters and logs the data uptil specified length using UART for FRDM and printf for the host machine. Function log_string() does not have length as its parameters and hence it calculates the length up and until the null character. Now the pointer and length are passed to log_data().
log_integer() converts an input integer data to an ascii string whose starting address is passed log_string(). log_flush() is used to flush out data from any given circular buffer through the transmit buffer and hence displaying contents of the buffer via UART for FRDM and printf for host machine.
Create_log_item() initializes log structure with values. It assigns log ID, length and PAYLOAD to the structure pointer passed as a parameter. log_item() is used to display or log items depending on the log structure initialization it passes the contents of log strctures log_data(), log_string(), log_integer().
In order to perform logging we always need to create a log item and then call log_item(). To overcome repeated calling of two functions we have defined a macro LOGIT.

OUTPUT SCREENSHOTS

UNIT TESTS:

When we run “make test” command the output of all 26 unit test functions is displayed on the screen below.



Test_data.c:
The unit tests for functions written in test_data.c are displayed on the screen when we run “make test TEST= “data”” command, the screenshot of the same is given below.

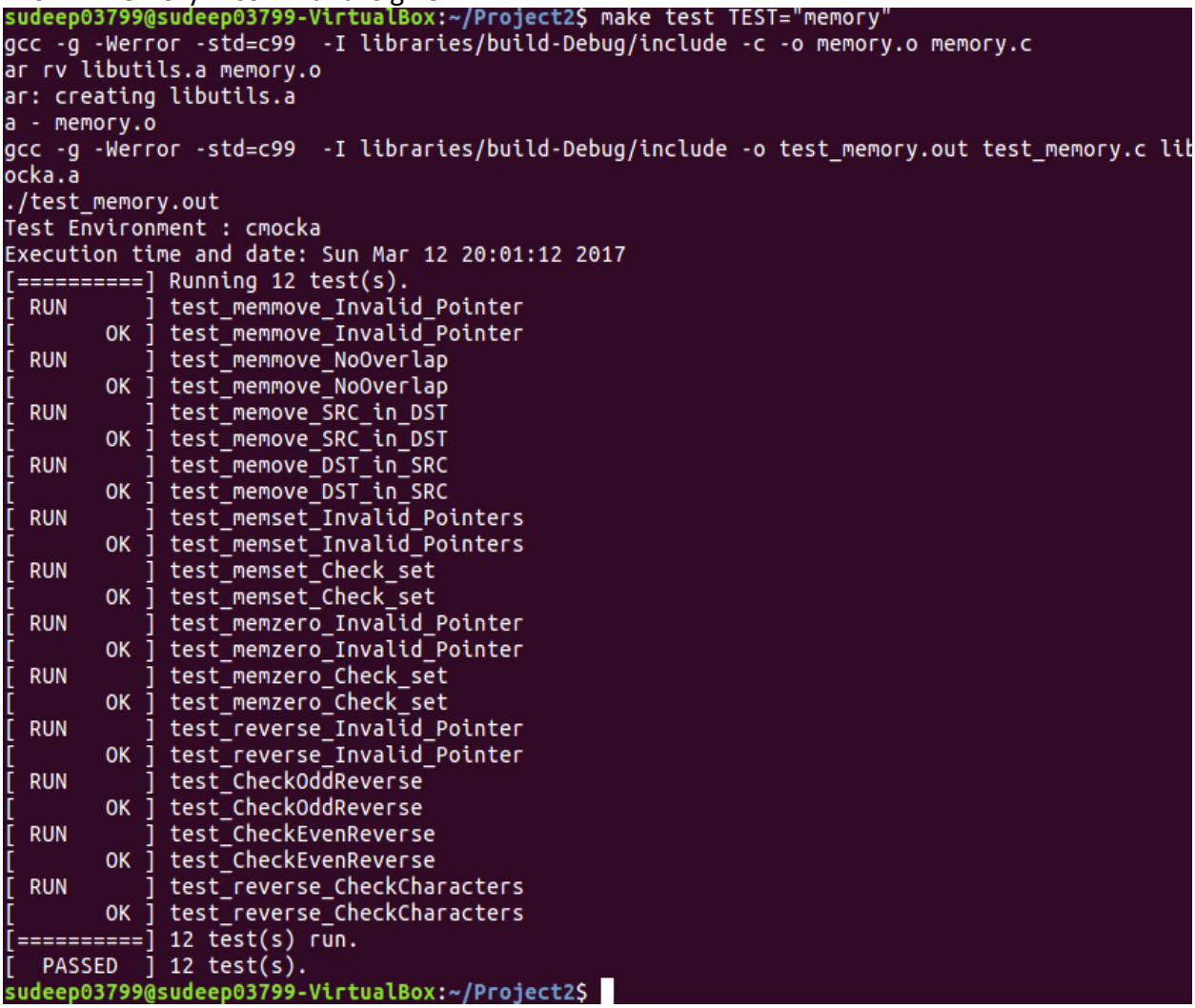
```
sudeep03799@sudeep03799-VirtualBox:~/Project2$ make test TEST="data"
gcc -g -Werror -std=c99  -I libraries/build-Debug/include -c -o data.o data.c
ar rv libutils.a data.o
ar: creating libutils.a
a - data.o
gcc -g -Werror -std=c99  -I libraries/build-Debug/include -o test_data.out test_data.c libutils.a libraries/build
.a
./test_data.out
Test Environment : cmocka
Execution time and date: Sun Mar 12 19:56:09 2017
[==========] Running 4 test(s).
[ RUN      ] test_big_to_little_Invalid_Pointer
[       OK ] test_big_to_little_Invalid_Pointer
[ RUN      ] test_big_to_little_Valid_condition
[       OK ] test_big_to_little_Valid_condition
[ RUN      ] test_little_to_big_Invalid_Pointer
[       OK ] test_little_to_big_Invalid_Pointer
[ RUN      ] test_little_to_big_Valid_condition
[       OK ] test_little_to_big_Valid_condition
[==========] 4 test(s) run.
[  PASSED  ] 4 test(s).
sudeep03799@sudeep03799-VirtualBox:~/Project2$
```

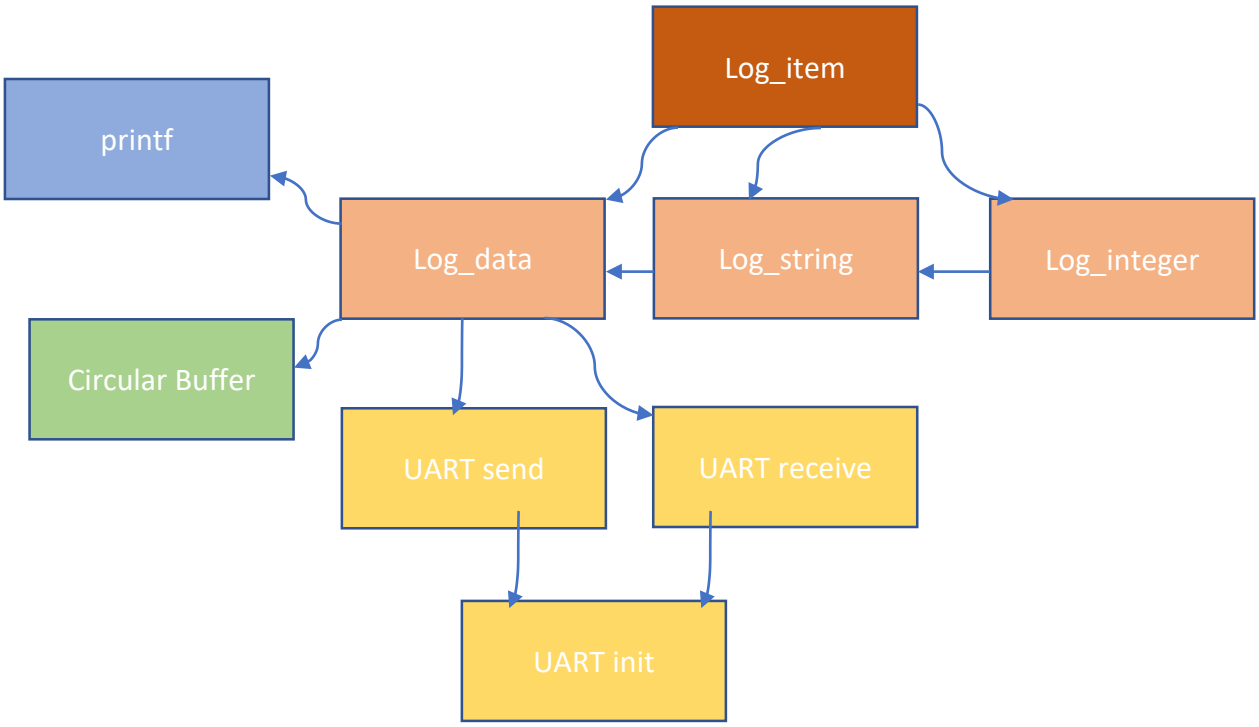
Test_cirbuff.c
The unit tests for functions written in test_data.c are displayed on the screen when “make test TEST= “cirbuff”” command is given.

```
sudeep03799@sudeep03799-VirtualBox:~/Project2$ make test TEST="cirbuff"
gcc -g -Werror -std=c99  -I libraries/build-Debug/include -c -o cirbuff.o cirbuff.c
ar rv libutils.a cirbuff.o
ar: creating libutils.a
a - cirbuff.o
gcc -g -Werror -std=c99  -I libraries/build-Debug/include -o test_cirbuff.out test_cirbuff.c libutils.a li
cmocka.a
./test_cirbuff.out
Test Environment : cmocka
Execution time and date: Sun Mar 12 19:57:33 2017
[==========] Running 10 test(s).
[ RUN      ] test_Allocate_Free
[       OK ] test_Allocate_Free
[ RUN      ] test_Invalid_Pointer
[       OK ] test_Invalid_Pointer
[ RUN      ] test_Non_Initialised_Buffer
[       OK ] test_Non_Initialised_Buffer
[ RUN      ] test_Buffer_Full
[       OK ] test_Buffer_Full
[ RUN      ] test_Buffer_empty
[       OK ] test_Buffer_empty
[ RUN      ] test_wrap_add
[       OK ] test_wrap_add
[ RUN      ] test_wrap_remove
[       OK ] test_wrap_remove
[ RUN      ] test_overfill
[       OK ] test_overfill
[ RUN      ] test_overempty
[       OK ] test_overempty
[ RUN      ] test_Add_remove
[       OK ] test_Add_remove
[==========] 10 test(s) run.
[  PASSED  ] 10 test(s).
sudeep03799@sudeep03799-VirtualBox:~/Project2$
```

Test_memory.c

The unit tests for functions written in test_data.c are displayed on the screen when “make test TEST= “memory”” command is given.





The software architecture diagram for the logger system

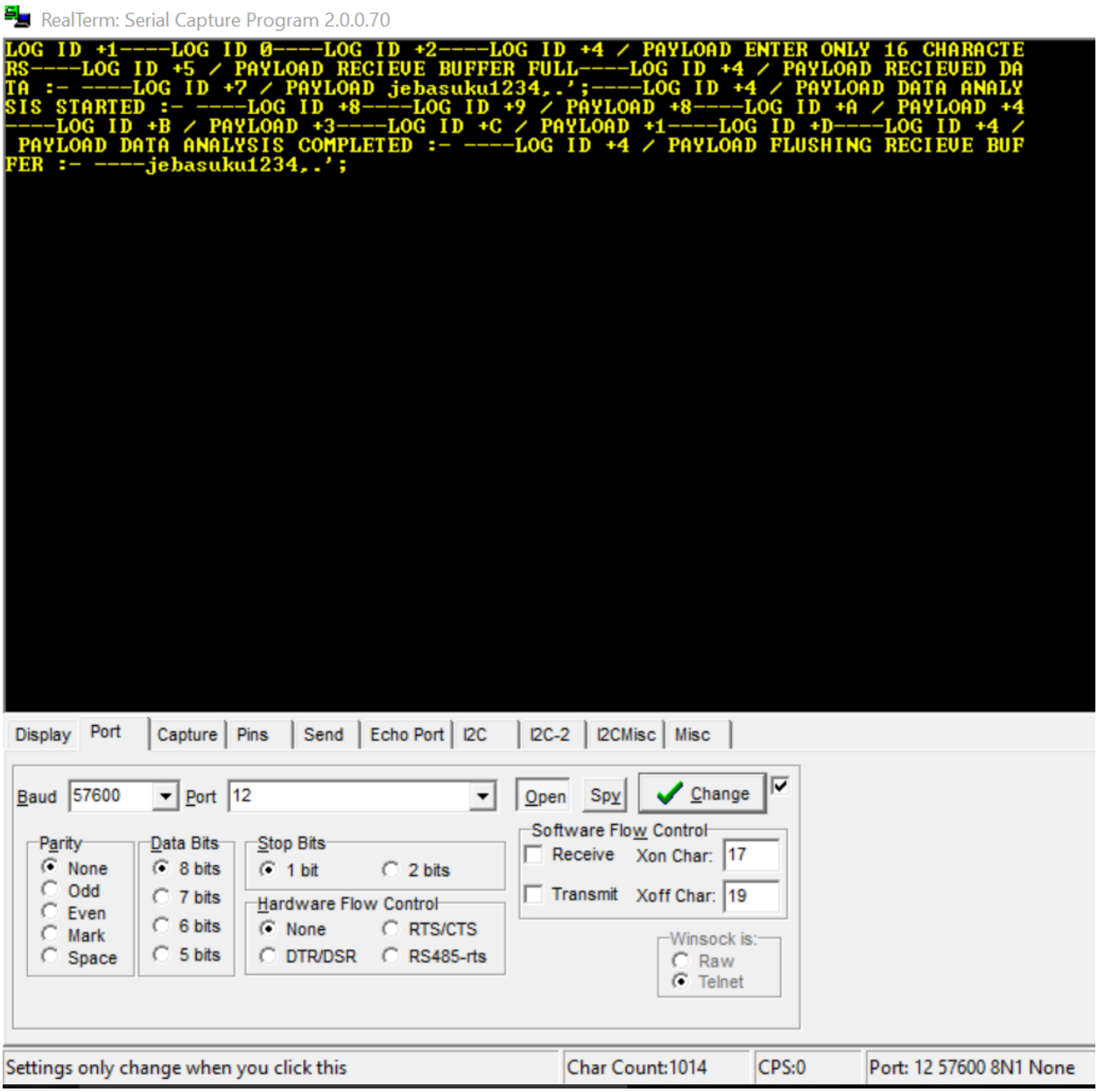
LOGGER:

We have designed a software system that can log efficiently in order to overcome the absence of printf on our freedomboard the software architecture system has a layout as described above. We have implemented our logger function on FRDM board and got the output as shown below

```
jeet@jeet-VirtualBox:~/Project2$ make log-host
gcc -DVERBOSE -w log/main.c log/logger.c log/analysedata.c log/cirbuff.c -o log.out
jeet@jeet-VirtualBox:~/Project2$ ./log.out
LOG ID 0----LOG ID +2----LOG ID +5 / PAYLOAD RECIEVE BUFFER FULL----LOG ID +4 / PAYLOAD RE
CIEVED DATA :- ----LOG ID +7 / PAYLOAD jebasuku1234,.#@----LOG ID +4 / PAYLOAD DATA ANALYS
IS STARTED :- ----LOG ID +8----LOG ID +9 / PAYLOAD +8----LOG ID +A / PAYLOAD +4----LOG ID
+B / PAYLOAD +2----LOG ID +C / PAYLOAD +2----LOG ID +D----LOG ID +4 / PAYLOAD DATA ANALYSI
S COMPLETED :- ----LOG ID +4 / PAYLOAD FLUSHING RECIEVE BUFFER :- ----jebasuku1234,.#@jeet
```

We have written a program that determines whether each character input is an alphabet, a number, punctuation mark or misc. Each log item has a predefined log ID and log Payload which is the data that is to be logged. As seen from the screenshot above we implemented it on our host machine and have obtained successful results.

The screenshot below shows similar implementation on FRDM board.



We also cross compiled our program for the Beaglebone Black successfully as depicted in the screenshot below.

```
sudeep03799@sudeep03799-VirtualBox:~/Project2$ make log-bbb
arm-linux-gnueabihf-gcc -DVERBOSE -w log/main.c log/logger.c log/analysedata.c log/cirbuff.c -o log.out
sudeep03799@sudeep03799-VirtualBox:~/Project2$ ls
cirbuff.c  cmocka.h  data.h     log      Makefile  memory.h  Redundant  test_cirbuff.c  test_memory.c
cirbuff.h  data.c    libraries  log.out  memory.c  Readme    Test.c     test_data.c     uni.h
sudeep03799@sudeep03799-VirtualBox:~/Project2$ ./log.out
bash: ./log.out: cannot execute binary file: Exec format error
sudeep03799@sudeep03799-VirtualBox:~/Project2$
```