## Project Plan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Milestone** | **Task** | **Best** | **Likely** | **Worst** | **total hours(likely)** | **Date** |
|  | **Develop and document project plan** | **1** | **2** | **2** |  |  |
| **Submit project plan** |  |  |  |  |  | **2019-02-12** |
|  | **Git repository** | **1** | **1** | **1** |  |  |
|  | **UART peripheral setup, ISR, and driver** | **2** | **3** | **4** |  |  |
|  | **RTC peripheral setup and driver** | **3** | **4** | **6** |  |  |
|  | **example code on the FRDM board** | **1** | **1** | **1** |  |  |
|  | **Initial makefile, main() and framework** | **2** | **3** | **5** |  |  |
|  | **Basic command line prompt and parse** | **1** | **2** | **3** |  |  |
|  | **Help function** | **1** | **1** | **2** |  |  |
| **Command prompt with help** |  |  |  |  | **15** | **2019-02-18** |
|  | **Allocate** | **1** | **1** | **2** |  |  |
|  | **Free** | **1** | **1** | **2** |  |  |
|  | **Invert** | **1** | **1** | **2** |  |  |
|  | **Display** | **2** | **2** | **3** |  |  |
|  | **Modify** | **1** | **1** | **2** |  |  |
| **Display and modify** |  |  |  |  | **6** | **2019-02-21** |
|  | **Time stampe generation from RTC** | **1** | **2** | **3** |  |  |
|  | **Execution time measurement** | **1** | **1** | **2** |  |  |
| **Timed functions** |  |  |  |  | **3** | **2019-02-22** |
|  | **PRNG algorithm and experiment** | **2** | **3** | **4** |  |  |
|  | **Write pattern** | **1** | **1** | **2** |  |  |
|  | **Verify pattern** | **1** | **1** | **2** |  |  |
| **Feature complete** |  |  |  |  | **5** | **2019-02-25** |
|  | **non-blocking input from terminal** | **2** | **2** | **4** |  |  |
|  | **time function** | **1** | **1** | **2** |  |  |
|  | **overall test** | **2** | **3** | **4** |  |  |
| **Port to Linux** |  |  |  |  | **6** | **2019-03-01** |
|  | **Use MakeFile to Generate FRDM Executable** | **0(aborted)** | **6** | **8** |  |  |
|  | **Compensation for RTC** | **0(aborted)** | **6** | **8** |  |  |
| **Optional work** |  |  |  |  | **12(optional)** | **2019-03-06** |
|  | **Final report** | **3** | **4** | **5** |  |  |
| **Submit final report** |  |  |  |  | **39(51 optional)** | **2019-03-08** |

## Documentation for the syntax

**Help**

Syntax: C0

Response:

* Print the usages of each command

**Exit (only available on the Linux side)**

Syntax: C1

Response:

* N/A

**Allocate memory**

Syntax: C2 <N of words in Hex>

Response:

* Valid or invalid parameter set
* Block index, size in terms bytes and words, and the start address if allocated successfully.

Error handling:

* Ignore the case with more than one parameter input
* Ignore the case with larger than the MAX\_SIZE value in the <N of words in Hex>.

**Free memory**

Syntax: C3 < index of the block>

Response:

* Valid or invalid parameter set
* Block index freed if done successfully.

Error handling:

* Ignore the case with more than one parameter input
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.

**Display memory words**

Syntax:

* C4 <index of the block> <offset> <range in terms of words(4-byte)>
* C4 <addr> <range in terms of words(4-byte)> -a

Response:

* Valid or invalid parameter set
* For block index, offset, and range option, display the 32-bit width data in HEX and corresponding addresses within the range if the range is inside of the allocated boundary.
* For absolute address option, display the 32-bit width data in HEX and corresponding address anyway and show warning if the range is outside of the allocated boundary.

Error handling:

* Ignore the case with more than three parameters input
* (For block index, offset, and range option)Ignore the case if the range is outside of the allocated boundary and print the error message.
* (For absolute address option)Ignore the case if the input address is not aligned with the 32-bit and print the error message.
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.
* Ignore the case with larger than the MAX\_SIZE value in the <offset> or <range in terms of words(4-byte)>.
* Ignore the case with zero value in <range in terms of words(4-byte)>.

**Write memory words**

Syntax:

* C5 <index of the block> <offset> <data in HEX>
* C5 <addr> <data in HEX> -a

Response:

* Valid or invalid parameter set
* For block index, offset option, write the 32-bit width data to the corresponding location if the location is inside of the allocated boundary.
* For absolute address option, write the 32-bit width data to the corresponding address anyway and show warning if the address is outside of the allocated boundary.

Error handling:

* Ignore the case with more than three parameters input
* (For block index, offset option)Ignore the case if the location is outside of the allocated boundary and print the error message.
* (For absolute address option)Ignore the case if the input address is not aligned with the 32-bit and print the error message.
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.
* Ignore the case with larger than the MAX\_SIZE value in the <offset>.

**Invert block**

Syntax:

* C6 <index of the block> <offset> <range in terms of words(4-byte)>
* C6 <addr> <range in terms of words(4-byte)> -a

Response:

* Valid or invalid parameter set
* For block index, offset, and range option, perform the bitwise inversion on the 32-bit width data in HEX and corresponding addresses within the range if the range is inside of the allocated boundary.
* For absolute address option, perform the bitwise inversion on the 32-bit width data in HEX and corresponding address anyway and show warning if the range is outside of the allocated boundary.
* Display the time consumed to implement the inversion(the logging overhead is not included).

Error handling:

* Ignore the case with more than three parameters input
* (For block index, offset, and range option)Ignore the case if the range is outside of the allocated boundary and print the error message.
* (For absolute address option)Ignore the case if the input address is not aligned with the 32-bit and print the error message.
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.
* Ignore the case with larger than the MAX\_SIZE value in the <offset> or <range in terms of words(4-byte)>.
* Ignore the case with zero value in <range in terms of words(4-byte)>.

**Write pattern**

Syntax:

* C7 <index of the block> <offset> <range in terms of words(4-byte)> <seed>
* C7 <addr> <range in terms of words(4-byte)> <seed> -a

Response:

* Valid or invalid parameter set
* For block index, offset, and range option, write the random number pattern within the range if the range is inside of the allocated boundary.
* For absolute address option, write the random number pattern anyway and show warning if the range is outside of the allocated boundary.
* Display the time consumed to implement the pseudo random sequence generation and memory writing(the logging overhead is not included).

Error handling:

* Ignore the case with more than four parameters input
* (For block index, offset, and range option)Ignore the case if the range is outside of the allocated boundary and print the error message.
* (For absolute address option)Ignore the case if the input address is not aligned with the 32-bit and print the error message.
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.
* Ignore the case with larger than the MAX\_SIZE value in the <offset> or <range in terms of words(4-byte)>.
* Ignore the case with zero value in <range in terms of words(4-byte)>.
* Ignore the case with larger than the RANDOM\_MAX value in <seed>.

**Check pattern**

Syntax:

* C8 <index of the block> <offset> <range in terms of words(4-byte)> <seed>
* C8 <addr> <range in terms of words(4-byte)> <seed> -a

Response:

* Valid or invalid parameter set
* For block index, offset, and range option, check the random number pattern within the range if the range is inside of the allocated boundary and print the checking result with the input seed.
* For absolute address option, check the random number pattern anyway and show warning if the range is outside of the allocated boundary and print the checking result with the input seed.
* Display the time consumed to implement the pseudo random sequence checking (the logging overhead is not included).

Error handling:

* Ignore the case with more than four parameters input
* (For block index, offset, and range option)Ignore the case if the range is outside of the allocated boundary and print the error message.
* (For absolute address option)Ignore the case if the input address is not aligned with the 32-bit and print the error message.
* Ignore the case with larger than the MAX\_BLOCK value in the < index of the block >.
* Ignore the case with larger than the MAX\_SIZE value in the <offset> or <range in terms of words(4-byte)>.
* Ignore the case with zero value in <range in terms of words(4-byte)>.
* Ignore the case with larger than the RANDOM\_MAX value in <seed>.

## Question

1. **How well did your effort estimates match with actual effort? Provide examples of both inaccurate and accurate estimates and discuss any contributing factors to the success or failings of your effort estimates.**

The estimated times for are met for functionality implementation. We have encountered some problems while porting the Linux code to the FRDM board. Despite we have put in the worst case estimation for the first allocation function, all of other functionality implements have taken us only for best case or even less thanks for similar designed structure and quite scalable main structure of the whole system.

The way NXP to implement SDK library functions is slightly different from what I have used before. Several settings, like heap size, matching the width of several data types, need to be explored before we can successfully port the code. So we are taking the worst case (slightly longer) hours to implement the FRDM board. Also for some unexpected situation(WeiTing had headed back China on last weekend), the last part of the project has been prolonged a little bit.

1. **Were there any tasks that you forgot to include in your project plan?**

I was expecting RTC to implement the timing function for the FRDM. It turns out to be easier to use the systick to implement the timing function.

Actually, I also misperceived the idea of the order of implementation of Linux and FRDM sides. I wants to implement the FRDM part first and port that to Linux later. However, the Linux side turns out to be easier to implement first.

1. **What is the largest block of memory you can allocate on each platform? Discuss.**

For the Linux side, I leveraged a memory testing code from the webpage: <https://stackoverflow.com/questions/8367001/how-to-check-heap-size-for-a-process-on-linux>.

In the program, the malloc tries increasing the memory size with a step of 100MB. The maximum memory size is determined by the last tried memory size when the pointer returned as NULL. The test result on my virtual machine is around 8.5 GB which corresponds with the memory I have assigned for the virtual machine. Therefore, the maximum block size on my virtual machine is around 8.5 GB.

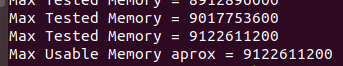


Figure 1 heap size checking result for Linux

For the FRDM board, the maximum block size is determined by the heap size setting which can be found in the linker file. The default value is 0x400 bytes or 1KB.

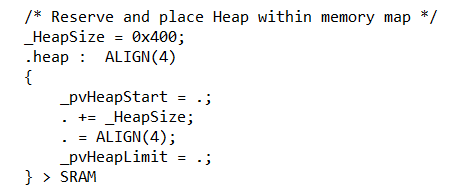


Figure 2 heap size setting inside the linker file

1. **What happens if you try to read outside of the allocated block?**

For the block index and offset option, the program just ignores the command prompts user an error saying that the combination of the block index and the offset is out of the range of what has been assigned.

For the absolute address option, the program would read anyway. If the range of the address is out of the allocated memory, the program would prompt user a warning that the address range is out of the boundary of what has been assigned.

1. **What happens if you try to write outside of the allocated block?**

For the block index and offset option, the program just ignores the command prompts user an error saying that the combination of the block index and the offset is out of the range of what has been assigned.

For the absolute address option, the program would write anyway. If the range of the address is out of the allocated memory, the program would prompt user a warning that the address range is out of the boundary of what has been assigned.

1. **Analyze the time it takes to invert memory for different memory block sizes. What is the relationship between time and size?**

For Linux side, the time consumed is generally proportional to the range of the block sizes with some jitters maybe due to the overhead and other background application running in the background.

1. **What are the limitations of your time measurements?**

For Linux side, the resolution of the different clocks can be gotten from the clock\_getres function. The one I am using is the CLOCK\_REALTIME, which is said to be one nanosecond resolution on my virtual machine environment and Ubuntu 18.04.

For the FRDM side, the timing function is achieved by systick, which generates the interrupt at a constant interval. The driven clock for the systick is the main clock source of the core. The clock ticks between each of the interrupt is set to be. Therefore the time interval between each interrupt is which would give us 30.51 microseconds resolution. To get the highest accuracy, it is better to keep the systick the highest priority of the interrupt which could preempt other interrupt service routine like UART. Fortunately, the SysTick interrupt is one of the exception interrupt which always has higher priority than the NVIC.

1. **What improvements you can make to the invert function, your program or your build, to make it run faster? How much improvement did you achieve?**  
   We can use the largest width variables in the program to take advantages of the highest addressable width the CPU or the system supports. With this method, we can lower our memory reading times, which means the CPU could load more bits and invert at once.

With higher optimization level like -O3(I am using the -O0 to eliminate the risk of compilation issue), the compiler would optimize more aggressively on the assembly code and might gain some efficiency.

1. **What algorithm did you choose for your pattern generator? How does it generate its pattern? What are its strengths and weaknesses?**

The algorithm is linear congruential generator (LCG) which is defined as the equation below:

Where m (m>0)is the "modulus", a(0< a <m) is the "multiplier", c(0<= c <m) is the "increment", is the "seed" or "start value".

There are several strengths of this algorithm. With appropriate choice of parameters, the pseudo random number sequence’s period is known and long. LCGs are fast and require minimal memory, which is a desirable property in an embedded system where the computation and memory resources are limited.

There are several weakness of this algorithm. The algorithm and the randomness of the generated number sequence is very sensitive to the parameters. We need to be very careful about the parameters selection. I am using the same parameter set as that in the glibc, in which the modulus is , the multiplier is 1103515245, and the increment is 12345. The parameter set also sets the upper bond on the seed and generated number, which is (less than the modulus). Since it is a pseudo random generator, it produces predictable outcomes given the same seed or the start value. Therefore, this algorithm is not recommended in an application with requirement of high-quality randomness, like cryptographic application for instance.