Joshua Ayden Blotnick, Abikamet Anbunathan | Project 1| ECEN5813 | March 11 2019

Project Plan

Milestone	Task	Best (Hrs)	Likely (Hrs)	Worst (Hrs)	Date
	Develop and document project plan	1	2	2	2019-02-12
Submit project plan					
	Build environment setup	0.5	0.5	1	2019-02-08
	Git repository	0.5	1	1	2019-02-15
	Initial makefile, main() and framework	3	4	5	2019-02-15
	Basic command line prompt and parse	1	2	2	2019-02-15
	Help function	0.5	1	1	2019-02-15
Command prompt with help					
	Allocate	1	1	2	2019-02-20
	Free	1	1	2	2019-02-20
	Display	1	1	2	2019-02-20
	Modify	1	2	3	2019-02-20
Display and modify					
	Execution time measurement	2	3	4	2019-02-22
	Invert	2	3	4	2019-02-22
Timed functions					
	Experiments and write ups	1	2	3	2019-02-27
	Write pattern	2	3	4	2019-02-27
	Verify pattern	2	3	4	2019-02-27
Feature complete					
	Final report	3	4	5	2019-03-01
Submit final report		22.5	33.5	45	

Command Line Function Syntax:

1. Allocate Memory [Number of Bytes]
2. Free Memory

3. Display Memory (optional -o option to specify offset instead of hex address) [Memory Address as an integer offset starting from 0 if -o option was chosen or as a hex value] (optional number of memory words to display)

   Ex: Display Memory -o 3 2 or Display Memory 0xAddress 2 -> will display 2 locations in memory starting from the 3rd offset address or the 0xAddress.

4. Write Memory (optional -o option to specify offset instead of hex address) [Memory Address as an integer offset starting from 0 if -o option was chosen or as a hex value] [Value to Write]

   Ex: Write Memory -o 3 356 or Write Memory 0xAddress 356 -> this will write the value 356 to the 3rd offset memory address or the 0xAddress.

5. Invert (optional -o option to specify offset instead of hex address) [memory address or offset]

   Ex: Invert -o 3 or Invert 0xAddress 3 -> will invert the 3rd offset memory address or the 0xAddress.

6. Wpattern(optional -o option to specify offset instead of hex address) [Memory Address as an integer offset starting from 0 if -o option was chosen or as a hex value] [seed value] (optional length of patterns to write to various memory addresses)

   Ex: Wpattern -o 3 4 5 or Wpattern 0xAddress 4 5 -> will write a pattern with a seed value of 4 to the 3rd, 4th, 5th, 6th, and 7th offset memory address or the 0xAddress 0xAddress+1, 0xAddress+2, 0xAddress+3, 0xAddress+4.

7. Verify Pattern(optional -o option to specify offset instead of hex) [Memory Address as an integer offset starting from 0 if -0 option was chose or as a hex address] [seed value] (optional length of patterns to verify at multiple addresses)

   Ex: Verify Pattern -o 3 4 5 or Verify Pattern 0xAddress 4 5 ->will verify a pattern with a seed value of 4 to the 3rd, 4th, 5th, 6th, and 7th offset memory address or the 0xAddress 0xAddress+1, 0xAddress+2, 0xAddress+3, 0xAddress+4.

8.

9. Help
10. Exit

Report questions

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.

   A: Overall, the time taken for each individual task was closely predicted, outside of a few mishaps in judgement, the project was able to be completed in a timely manner. For example: The time taken for every task included in the “command prompt with help” section was accurately predicted. Giving that this was a simple command line interface and that one just needed to parse the user’s input, it did not take very long and was easy to predict the timing. The Write Memory function time prediction, however, was inaccurate. This was due to the difficulty in restricting the user from placing an out of bounds address and figuring out a clever method to convert the hexadecimal string into a hex number used to calculate the bounds of the address.

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

   A: Porting the code over to the FRDM board as well as developing a clean LUT algorithm that would concisely propagate the user’s input to the correct .C file took a decent amount of time and was not included in the project plan.

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

   Allocation Limit is determined by 2 factors. Available memory and the memory management implementation of the OS
   Ubuntu 64 bits OS in Virtual Machine: Allocation Limit: 536870000 words ~ 4.1 Gb which was equal to the amount of RAM allocated for the VM.
   In FRDM: Allocation limit 252 words ~ 1 Kb which was probably the heap memory allocated by the RAM.

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

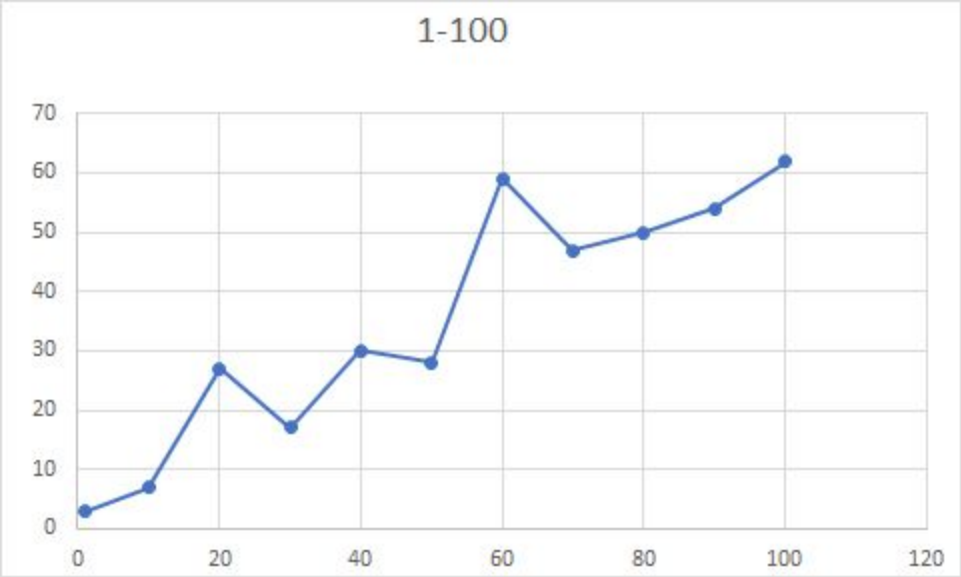
   A: Reading outside of an allocated block is not allowed by the program. The program checks the bounds of the starting address and the number of blocks the user allocated and determines if the address the user provided is within that range, if it is not a proper error is thrown. The hosting environment shouldn’t allow reading to random addresses as this could be another application's memory.

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

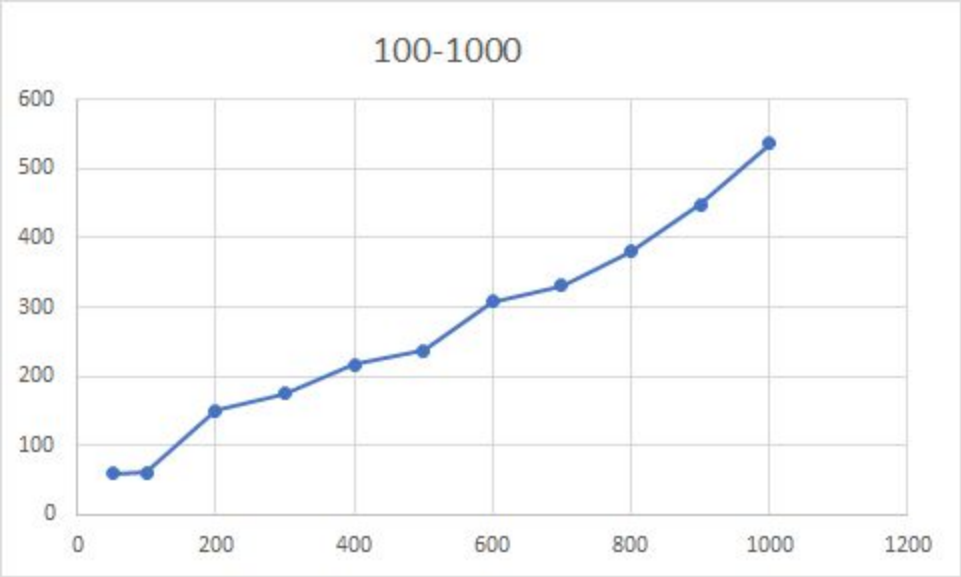
A: Writing outside of an allocated block is not allowed by the program. The program checks the bounds of the starting address and the number of blocks the user allocated and determines if the address the user provided is within that range, if it is not a proper error is thrown. The hosting environment shouldn't allow writing to random addresses as this could be another application's memory.

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

In General time taken directly proportional to the size of memory block inverted
However there was some anomalies observed on this linear trend for lower word sizes(E.g Inverting 30 words took less time than 20 words).



Xaxis: Number of Words(1 word = 4 bytes)
Yaxis: Time taken in microseconds



Xaxis: Number of Words(1 word = 4 bytes)
Yaxis: Time taken in microseconds

7. What are the limitations of your time measurements?

   1) FRDM uses a 32 bit timer so when number of clock ticks beyond 2^32 it would roll over resetting the timer

   2)Our time measurement is based on number of timer interrupts(which trigger every millisecond) which may have inaccuracies added due to the latency of the ISR (Interrupt Service Routine)

8. 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 tried two ways to Invert the function:

   a) Using a NOT bitwise operation

   b) Performing an XOR bitwise operation with 0xFF(Contains all 1 bits).

   Both ways of doing it corresponded to the same number of assembly instructions differed only in one instruction the NOT and the XOR instruction.
   We used NOT operation as it takes requires a smaller circuit at the LOGIC Gate level hence should take lesser time to execute compared to XOR(which uses more logic gates)

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

   A: The pseudo-random number algorithm utilized XORs the seed value and the current timestamp generated at the beginning of the program, multiplies this value by the address

one intends to write to, then modulos the value with sizeof(uint32_t/2) to ensure that the value to be written is able to fit in the memory location allocated.