ECE 461/561
EMBEDDED SYSTEM OPTIMIZATION
PROJECT 1 REPORT

Name: Cristian Hellmer

Unity ID: crhellme

Introduction

In this project, we were tasked with optimizing a program which displays images held on an SD card sequentially. These optimizations are focused on improving the execution time of the program. Throughout the execution of the program, a profiler was ran taking periodic samples at a frequency of 1 kHz. The profiler is also aware of which function is being ran when a sample is taken. These samples are accumulated continuously until the program ends. At the end of the program, profiler information containing the number of samples taken during the entire program execution and individual samples obtained for each function are displayed on the LCD. This profiling information was very useful in evaluating the effectiveness of the optimizations. Major program components include the uLibSD SPI SD card interface, PetitFatFS file system, and picoJPEG JPEG decoder.

Execution Time Optimization

Code Version	Total # Samples	Top Function		2nd Function		3rd Function	
		# Samples	Name	# Samples	Name	# Samples	Name
Initial Code	18147	5470	DMA_OpenChannelsForSpi	2277	getBit	1786	huffDecode
1	18132	5545	DMA_OpenChannelsForSpi	2264	getBit	1753	huffDecode
2	14899	5487	DMA_OpenChannelsForSpi	2872	huffDecode	2374	decodeNextMCU
3	14862	5520	DMA_OpenChannelsForSpi	2816	huffDecode	2375	decodeNextMCU
4	10668	2802	huffDecode	2378	decodeNextMCU	1370	DMA_OpenChannelsForSpi
5	10353	2826	huffDecode	2391	decodeNextMCU	1375	DMA_OpenChannelsForSpi
6	10267	4382	decodeNextMCU	1494	getBits	1337	DMA_OpenChannelsForSpi
7	9134	4377	decodeNextMCU	1522	getBits	735	LCD_Write_Rectangle_Pixel
8	9033	4225	decodeNextMCU	1464	getBits	782	LCD_Write_Rectangle_Pixel
9	8974	4164	decodeNextMCU	1427	getBits	753	LCD_Write_Rectangle_Pixel
10	8515	4111	decodeNextMCU	1479	getBits	489	getChar
11	8307	3984	decodeNextMCU	1453	getBits	489	getChar
12	8199	3976	decodeNextMCU	1485	getBits	486	DMA_OpenChannelsForSpi
13	7967	3874	decodeNextMCU	1446	getBits	476	getChar
14	7386	2105	decodeNextMCU	2036	huffDecode	727	getBits

Code version 1 optimization: The optimization performed for this version of the code was switching the value of the LESS_INLINING macro to 0 instead of 1. I expected this to help minimally because inlining can be used by the compiler to optimize the program. The LESS_INLINING macro is used to force certain functions to not be inlined by the compiler using __attribute__((noinline)). Implementing this optimization took approximately 2 minutes. The before execution count was 18147 and the after execution count was 18132. The speed-up factor was 1.0008. This was expected considering the program was quite unoptimized at this point and I would doubt inlining could remedy these issues effectively especially since O3 optimization was not in place.

Code version 2 optimization: The optimization performed for this version of the code was changing compiler settings such O3 optimization, optimize for time, and –fpmode=fast. I expected this to help a lot because the compiler is fully capable of optimizing code with extreme effectiveness. Implementing this optimization took approximately 1 minute. The before execution count was 18132 and the after execution count was 14899. The speed-up factor was 1.2170. This was expected considering the built-in optimization does a very good job at speeding up unoptimized code.

Code version 3 optimization: The optimization performed for this version of the code was removing the debug bits used for gaining a better understanding of the program being ran. I expected this to help minimally because the modification of the debug bits is not a time intensive operation even though they are changed very frequently. Implementing this optimization took approximately 30 minutes. The before execution count was 14899 and the after execution count was 14862. The speed-up factor was 1.0025. This was expected.

Code version 4 optimization: The optimization performed for this version of the code was raising the baud rate of the SPI communications. I expected this to help a lot considering the SPI communication is how images are read from the SD card. The SD card is also read for a large percentage of the program. Implementing this optimization took approximately 20 minutes. The before execution count was 14862 and the after execution count was 10668. The speed-up factor was 1.3931. This was expected.

Code version 5 optimization: The optimization performed for this version of the code was replacing the accessing of GPIO ports used in the code with fast GPIO access. I expected this to help minimally because I envision GPIO ports as fast all the time. Implementing this optimization took approximately 10 minutes. The before execution count was 10668 and the after execution count was 10353. The speed-up factor was 1.0304. This was expected.

Code version 6 optimization: The optimization performed for this version of the code was replacing the existing huffDecode function with one that processes 2 bits at a time. This function was found in an ECE 560 project. I expected this to help a lot because, at the time, huffDecode was the function with the most samples. Implementing this optimization took approximately 15 minutes. The before execution count was 10353 and the after execution count was 10267. The speed-up factor was 1.0084. This was unexpected. I believe the improvement was minimal because I overestimated the amount of times huffDecode was called. After the implementation of the new function, huffDecode disappeared from the top functions by sample number. It is unknown to me why it was the number 1 function before this optimization.

Code version 7 optimization: The optimization performed for this version of the code was buffering the last sector read by the SD_read function to be compared to the sector being read later. This is beneficial because oftentimes the data being read after another read is from the same sector. I expected this to help a reasonable amount due to the fact that data is often read from the same sector. Implementing this optimization took approximately 20 minutes. The before execution count was 10267 and the after execution count was 9134. The speed-up factor was 1.1240. This was expected.

Code version 8 optimization: The optimization performed for this version of the code was implementing LUTs in the functions getExtendTest and getExtendOffset in place of lengthy case statements. I expected this to help a small amount due to the fact that these functions were not dominating the runtime in the first place. Implementing this optimization took approximately 25 minutes. The before execution count was 9134 and the after execution count was 9033. The speed-up factor was 1.0112. This was expected.

Code version 9 optimization: The optimization performed for this version of the code was replacing lengthy conditional statements containing many OR comparisons with conditional statements with AND comparisons in the functions idctRows and idctCols. I expected this to help minimally due to the fact that these functions were not dominating the runtime in the first place and because this only changes 2 lines. Implementing this optimization took approximately 15 minutes. The before execution count was 9033 and the after execution count was 8974. The speed-up factor was 1.0066. This was expected.

Code version 10 optimization: The optimization performed for this version of the code was replacing the GPIO_Write(cmd) macro on LCD_driver.h with code that uses hardware support to set and clear GPIO bits instead of software. I expected this to help a reasonable amount because the macro is used frequently when communicating to the LCD display. Implementing this optimization took approximately 5 minutes. The before execution count was 8974 and the after execution count was 8515. The speed-up factor was 1.0539. This was expected.

Code version 11 optimization: The optimization performed for this version of the code was replacing getBit function with a faster version of the function that optimized the if-else statement to more efficiently execute instructions. I expected this to help a reasonable amount because the getBit function is predominant in the sample count. Implementing this optimization took approximately 15 minutes. The before execution count was 8515 and the after execution count was 8307. The speed-up factor was 1.0250. This was expected.

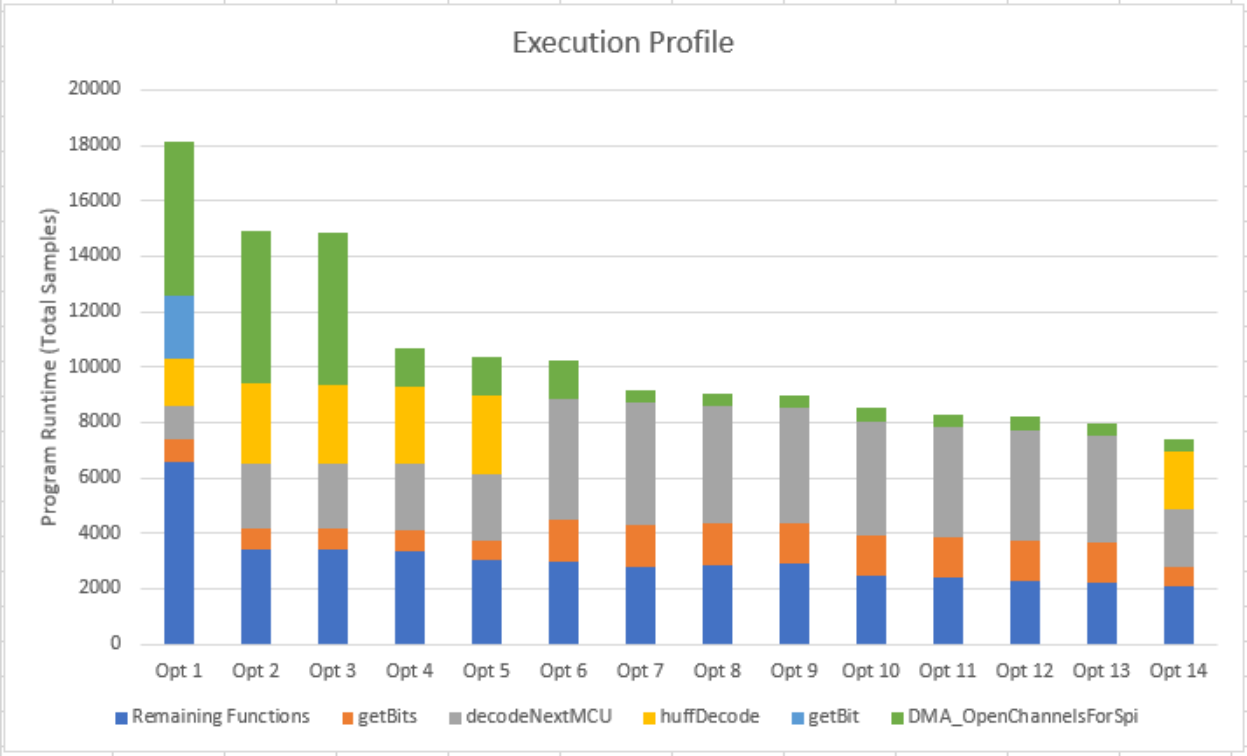
Code version 12 optimization: The optimization performed for this version of the code was doing the math required from the idctRows and idctCols functions using the native memory type of 32 bit int. I expected this to help a reasonable amount because the idctRows and idctCols functions are called frequently. Implementing this optimization took approximately 15 minutes. The before execution count was 8307 and the after execution count was 8199. The speed-up factor was 1.0132. This was expected.

Code version 13 optimization: The optimization performed for this version of the code was implementing the use of CMD18 for SD card interaction rather than CMD17. I expected this to help a reasonable amount because the SD card interaction is a very large part of the function of the program. Implementing this optimization took approximately 1 hour. The before execution count was 8199 and the after execution count was 7967. The speed-up factor was 1.0291. This was unexpected. I believe the small increase is due to the fact that changing sectors to go one above the previous is slightly less common than anticipated.

Code version 14 optimization: The optimization performed for this version of the code was replacing calls to functions in the functions huffDecode and getOctet with the code held in that function specified for the appropriate values which were being passed. I expected this to help a reasonable amount because the huffDecode and getOctet functions are used a lot during the program. Implementing this optimization took approximately 47 minutes. The before execution count was 7967 and the after execution count was 7386. The speed-up factor was 1.0787. This was expected.

Analysis and discussion of results

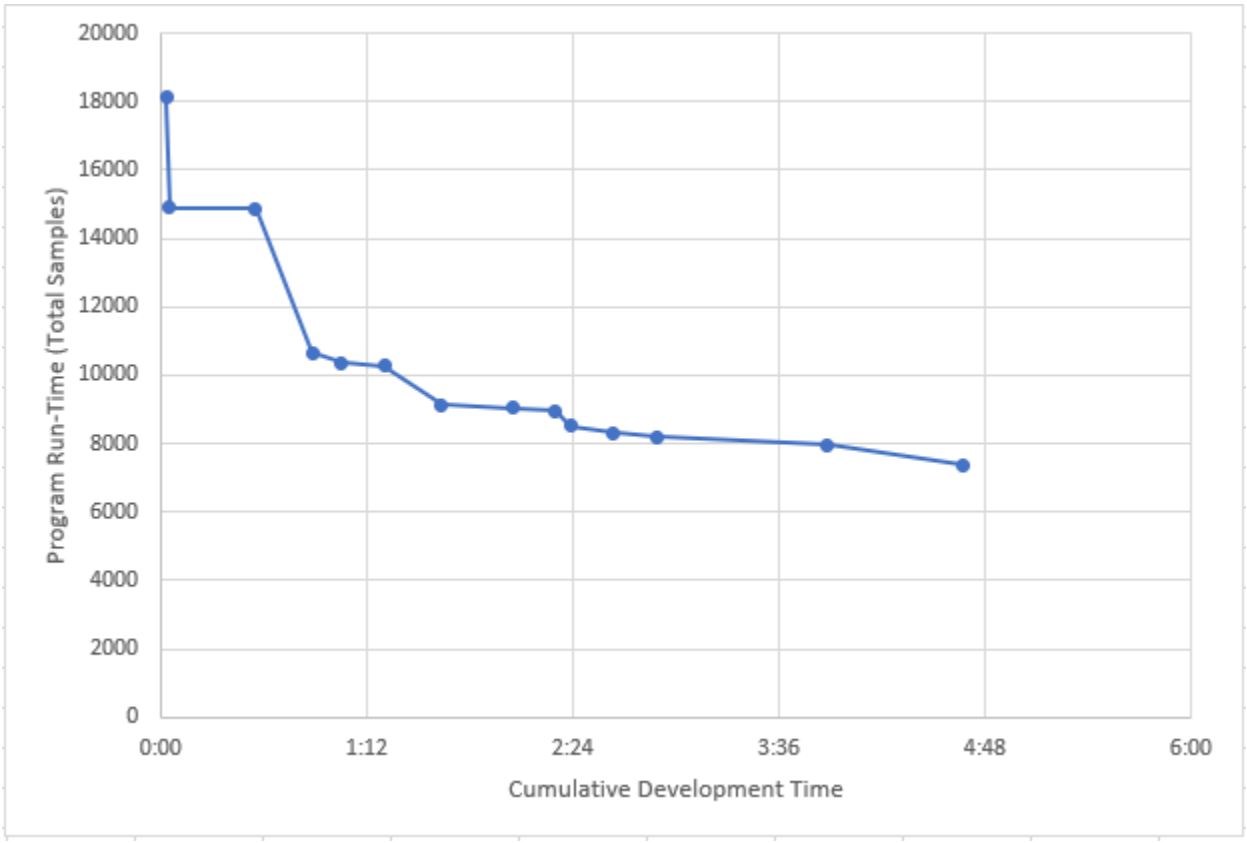
- Create a summary stacked column chart showing the execution profile for each successful optimization step.



- Which optimizations improved performance the most? Which offered the best improvement/development time?

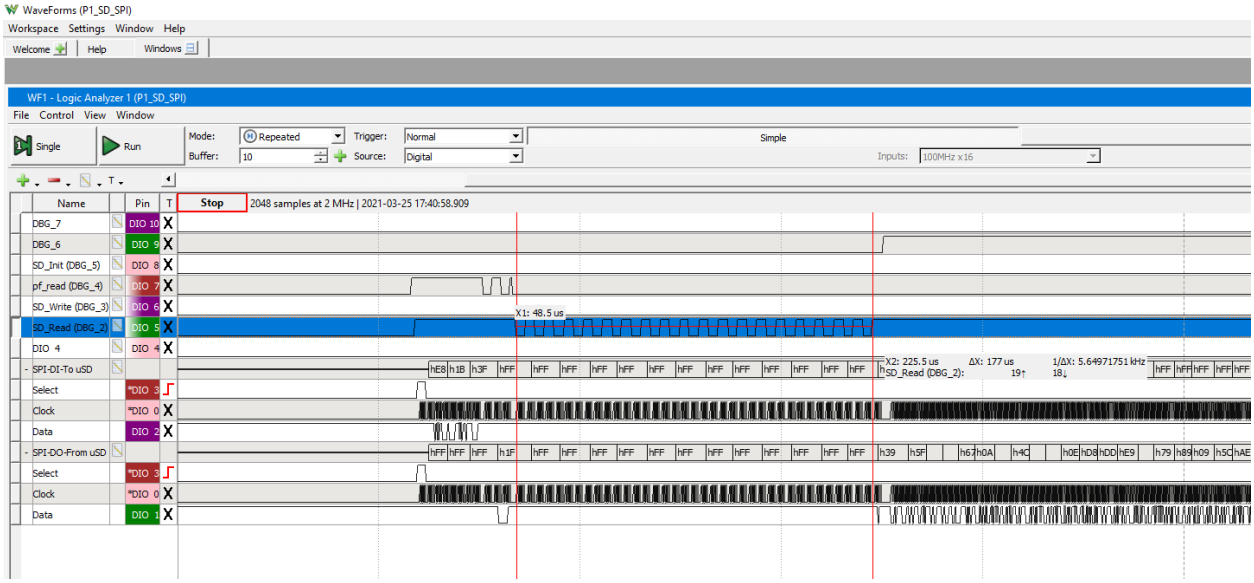
The optimizations which improved performance the most were enabling the compiler options for O3 optimization, optimization for time, and fast floating-point approximation, and raising the SPI baud rate to its maximum. The best improvement/development time was the enabling of the compiler options for O3 optimization, optimization for time, and fast floating-point approximation. This optimization took the least amount of time but offered the second-best improvement of any optimization.

- Plot the program execution time or sample count (Y) as a function of development time (X).



SD Card Read Access Time

My SD Card read access time was measured to be 177 us.



Retrospective

During this project, I learned that optimization of a program can sometimes require extensive knowledge about the operation of the system as a whole and sometimes there are glaring programing mistakes which can be fixed with minimal effort. I also learned that the time spent on an optimization does not always correlate to the time saved during the program runtime. Because of this, I will effectively evaluate the expected increase in performance vs the expected time to develop a solution before jumping into optimizing future programs. Luckily, there were no technical issues experienced during the completion of this project.