ECE 461/561
EMBEDDED SYSTEM OPTIMIZATION
PROJECT 1: SPEED ANALYSIS AND OPTIMIZATION

OVERVIEW

In this project you will try to optimize[1] the execution speed of a program which decodes a JPEG image and displays it on the LCD. Use the starter program on Github at Project 1. The picoJPEG decoder (https://github.com/richgel999/picojpeg) was used for this project. There is an explanation of how to decode JPEG images available online (https://www.impulseadventure.com/photo/jpeg-huffman-coding.html), with related articles on that website.

PERFORMANCE GOALS

Optimize the program to reduce its execution time. Keep track of the development time you spend on each optimization and include that in the report.

To get full performance credit, you'll need to meet the following time goals for image decode and display, which includes the overhead of the profiler running at 1000 samples/second, but no validation (ENABLE_PIXEL_HASH is 0, described below).

- ECE 461: 500 samples
- ECE 561: 435 samples

Faster or slower times will result in extra or partial credit. The 461 and 561 students with the fastest time will each receive an additional 5 point performance bonus on the project score.

VALIDATION

Be sure to validate your code as you make changes to it in your optimization process. The symbol ENABLE_PIXEL_HASH in LCD.h controls validation. When it is 1 (or any non-zero value), each pair of pixel data bytes written to the LCD is used to update a hash stored in the variable pixel_data_hash and later returned by LCD_JPEG. The main function checks the return value of the LCD_JPEG function call against the correct value (CORRECT_IMAGE_HASH, defined in JPEG_image.h) to confirm the correct pixel data bytes have been sent to the display. A message is displayed at the bottom of the LCD indicating success or failure.

- When developing an optimization, first set ENABLE_PIXEL_HASH to 1 to confirm correctness. Once the optimization is successful, clear ENABLE_PIXEL_HASH to 0 in order to measure timing performance without the hashing overhead. Repeat as needed.
- If you modify code which sends pixel data bytes to the LCD, make sure it updates pixel_data_hash correctly or else validation will fail.

[1] You will improve the program's speed, but probably not *optimize* it.

DEVELOPMENT PROCESS SUGGESTIONS

- Build the program, run it, and examine the profile for the top functions.
- Use the debugger to look at mixed source and object code in the Disassembly window. Look for mismatches and extra work. The debugger will start up faster if you load the code into the simulator rather than your MCU (check Target Options -> Debug -> Use Simulator).
- The profiling support script (Scripts\update_regions_MDK.bat) may be enhanced to disassemble the object file (using the tool **fromelf**) into a text file (*.dis.txt). This listing does not have interleaved C source code listings (which the debugger does provide).
- The compiler may aggressively inline functions, giving the top function many more samples than you would expect, and not giving a very fine-grain view into where the time is used. To prevent this, you can tell the compiler not to inline a particular function into any calling function. Do this with the __attribute__((noinline)) modifier to the function definition before the function declaration. See the ARMCC User Guide for more information, and similar calls (__declspec(noinline) and #pragma no_inline). For example:

```
// Red: add new function definition preventing inlining of huffDecode into any calling functions
static uint8 huffDecode(const HuffTable* pHuffTable, const uint8* pHuffVal) __attribute__((noinline));
// Original function declaration
static uint8 huffDecode(const HuffTable* pHuffTable, const uint8* pHuffVal)  {
        // code for function body is here
}
```

- Use conditional compilation to enable or disable the inlining of functions listed above, making it easier to switch back and forth between two versions of code (faster code vs. more detailed profile)
- Experiment with enabling cross-module optimization and MicroLib.
- Add the **--remarks** compiler command line option in Target Options -> C/C++. Look in the build output window for remarks related to optimization which may help you guide you. Ctrl-F will help you search the text in the build output window.

OPTIMIZATION SUGGESTIONS

- You are allowed to modify any code in the program. This includes adding new functions and changing existing functions.
- Within a hot function, look for loops. Hotspots are the most-frequently executed loop bodies, which typically are innermost (most deeply nested). For a full-screen JPEG, 320*240 = 76,800 pixels are computed and plotted. Examine how the **pixel information** is sent to the LCD and look for inefficiencies to eliminate.
- Pixels are made of three components: eight bits each of red, green, and blue. Try to use the full 32 bit width of the registers to parallelize operations.
- Rather than perform read/modify/write operations on PDOR to modify GPIO port outputs, use the PSOR, PCOR and PTOR registers.
- The huffDecode function in picojpeg.c has the following comment.

```
// This func only reads a bit at a time, which on modern CPU's is not terribly efficient.
// But on microcontrollers without strong integer shifting support this seems like a
// more reasonable approach.
```

  The ARM Cortex-M0+ can shift by 1 to 31 bits in a single clock cycle. Can you modify the huffDecode function to read more than one bit at a time? Feel free to look online for a solution and adapt it to use the existing HuffTables of picojpeg. Be sure to document your work in your report.
- The function decodeNextMCU returns eight bits of data for each pixel component. However, only the 5 most-significant bits of the R and B components and the 6 most-significant bits of the G component are used for the LCD. Can decodeNextMCU do less work and skip those bits?

DEALING WITH OBJECT CODE SIZE LIMITS

If a build fails due to excessive code size (as seen below), delete most of the entries in RegionTable in region.c and rebuild.

```
.\Objects\LCDs_Profiler.axf: error: L6047U: The size of this image (33488 bytes) exceeds the
maximum allowed for this version of the linker
Finished: 0 information, 0 warning, 0 error and 1 fatal error messages.
".\Objects\LCDs_Profiler.axf" - 1 Error(s), 7 Warning(s).
Target not created.
```

In order to automatically reduce ROM size, the profiling support script (Scripts\update_regions_MDK.bat) has been modified to omit region table entries for functions beginning with _, __, Q_ or PIT_ by using the –x prefix option. Feel free to omit more functions as needed. Refer to the GetRegions documentation (on Github at Tools\GetRegions) for further information.

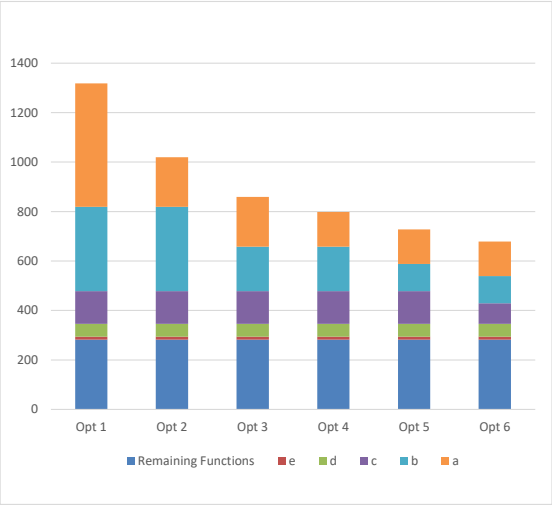
REFERENCE: JPEG FILE PREPARATION

You do not need to modify the JPEG file. However, if you wish to display a different JPEG in the future, the JPEG image was processed as follows:

- Resized to fit LCD (320x240 pixels)
- Deleted EXIF information using a tool (https://www.verexif.com/en/, or http://www.exifpurge.com/)
- Converted to C file (JPEG_image.c) declaring a uint8_t array using bin2header.exe (executable at https://sourceforge.net/projects/bin2header/). Added JPEG_image.c to project.
- Created header file JPEG_image.h.

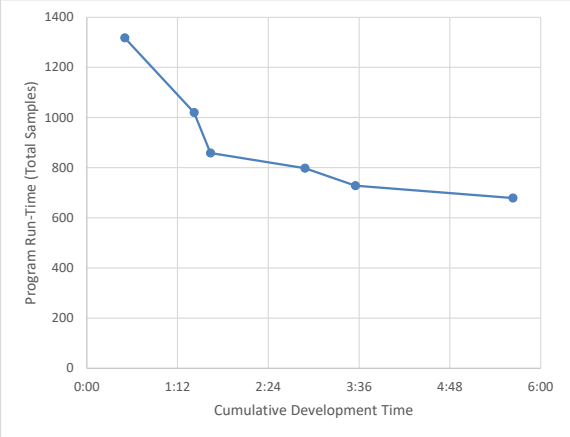
DELIVERABLES

- Archive of entire project directory, including source code and subdirectories.
- Project Report (PDF)
    - Introduction
    - Execution Time Optimization (in order performed, including dead ends (ineffective attempts))
        - Address the following points for each optimization you tried
            - What optimization did you perform? Why did you expect it to help, and by approximately how much?
            - How much time did implementing the optimization correctly take? This includes planning, coding, testing, debugging, etc.
            - What were the before and after execution sample counts? How much did the performance actually improve (speed-up factor)? If the amount of improvement was unexpected, examine and explain.
    - Analysis and discussion of results
        - Create a summary stacked column chart showing the execution profile for each successful optimization step, as shown here. To make the data easier to understand, put the five[2] functions with the most samples at the top of each column, in order of decreasing initial sample count. Group the remaining functions into a single item (blue, “Remaining Functions”).



[2] You can list more than the top five functions if you like.

- Which optimizations improved performance the most? Which offered the best improvement/development time?
- Plot the program execution time or sample count (Y) as a function of development time (X).



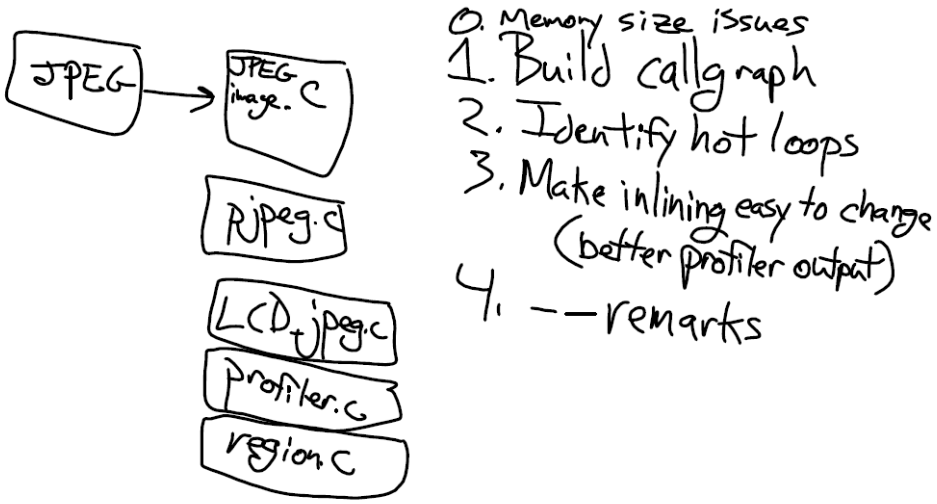
- Lessons learned in this project, and how you might do things differently next time
  - Technical issues (processor, peripherals, compiler, tools, assembly code, etc.)
  - Changes to your own development process

SCORING

- Functionality: 30%
- Speed: 30%
- Report: 40%

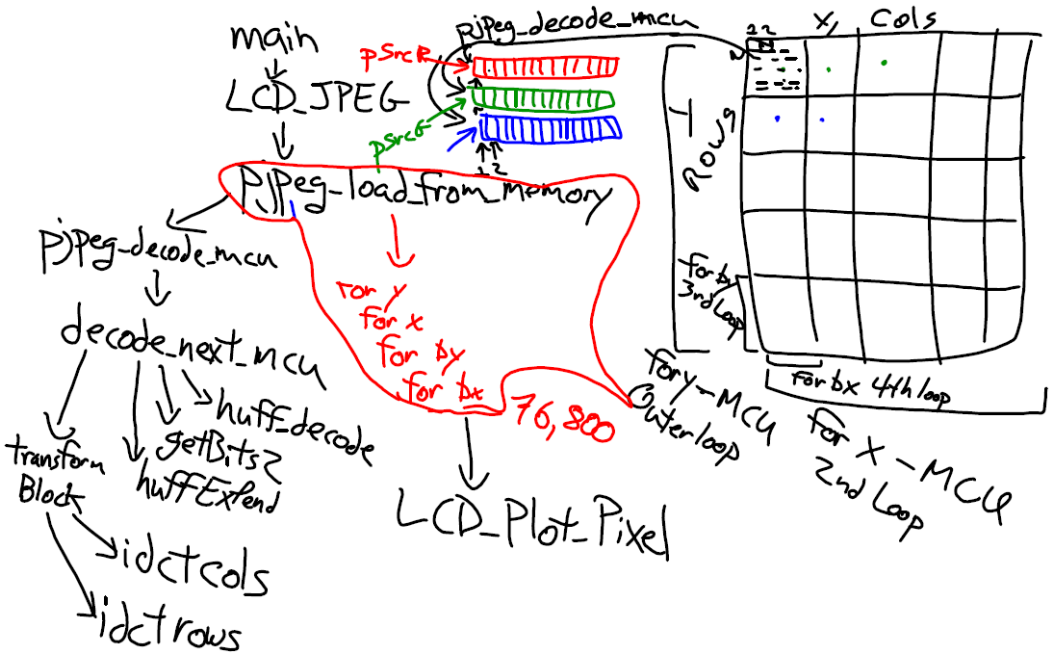
Extra credit will be granted for exemplary work in any categories.

APPENDIX: DIAGRAMS FROM 2/25/2019 LECTURE

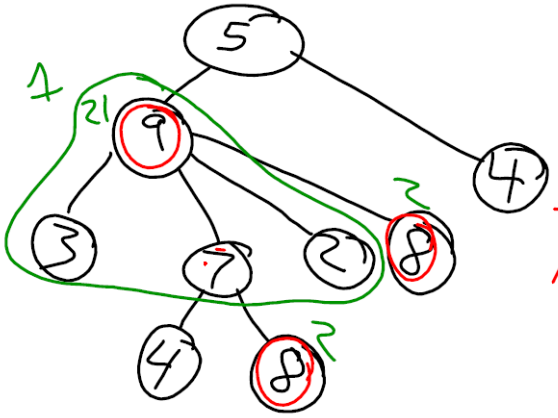


ECE 561 Dean Lec. 14

ECE 461-561 lecture 13 notes.notebook February 25, 2019



Low Opt. ⟷ High
Easy to read — Hard to read
-O2
-O3



Manage Complexity by enabling/disabling inlining

See total Performance impact

See which functions take time

ECE 561 Dean Lec. 14
ECE 461-561 lecture 13 notes.notebook February 25, 2019

Memory Size Limits 32KB

- bit-mapped Fonts — smaller font
- JPEG — compressed
- Regions — -X
  - if can't build, delete most entries