ECE 461/561 LAB SOLUTION: “OPTIMIZING” EXECUTION SPEED

GETTING STARTED

0. What is your unity ID (e.g. agdean)?

- Scroll to the top of the Build Output window, as shown below. The first line indicates the details (e.g. version, update, build numbers) of the compiler used.

1. Copy that line into your report.
**** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'*

BASELINE PERFORMANCE

2. Enter the total sample count, and the names and sample counts for the top three functions (those which dominate the execution time).
2091 Total Samples
*788 LCD_24S_Write_*
537 LCD_Write_Rect
377 LCD_Text_Print
3. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
2091 ms/100 = 20.91 ms

CHECKING BUILD CHAIN OPTIMIZATION SETTINGS

4. What is the currently selected optimization level?
default
5. Is the code optimized for time?
no

- Change the optimization level to 3 and select optimize for time. Rebuild the program three times and download it. Run the program.

6. Enter the profile information below.

Item	**Name**	**Sample Count**
Total Samples	n/a	*1977*
Top Function	*LCD_24S_Write*	*801*
2nd Function	*LCD_Write_Rect*	*471*
3rd Function	*LCD_Text_Print*	*293*

7. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
19.77 ms

LAZY DISPLAY UPDATES

LCD_ functions should dominate the execution time. There are many, many different ways to try to speed them up, but let's start with something easy: Don't update the roll or pitch information on the display if it hasn't changed since the last time it was drawn.

- Change the definition of LAZY_TEXT_UPDATE from 0 to 1.

8. Examine the code controlled by LAZY_TEXT_UPDATE. Why should this change speed up the program?
The text is only updated if the roll changes by more than 0.1 degrees. If the board is stable, then the text won't be updated and the code will finish sooner.

- Build the program (three times) and download it. Run the program five times.

9. What are the minimum and maximum total observed sample counts? What is the difference between them?
Approximate: 863, 738, 866, 846, 660. Max diff = 206.

- Change the definition of LAZY_GRAPHICS_UPDATE from 0 to 1.

10. Examine the code controlled by LAZY_GRAPHICS_UPDATE. Why should this change speed up the program?
The up arrow indicator is only updated (erased and redrawn) if the roll changes by more than 0.1 degrees. If the board is stable, then the indicator won't be updated and the code will finish sooner.

- Build the program (three times) and download it. Run the program five times.

11. What are the minimum and maximum total observed sample counts? What is the difference between them?
Approximate: 115, 162, 203, 224, 186. Max diff = 109.

INPUT DATA VARIABILITY

EVALUATING IMPACT ON RUN TIME

The code's execution time depends on data derived from input conditions such as board orientation and movement. Drawing the arrow on the LCD depends on the angle of the lines. A horizontal or vertical line is just one rectangle, so it can be drawn quickly. A diagonal line is multiple squares, each drawn as a rectangle. Any other kind of line is multiple rectangles of two different sizes.

- Run the program several times and observe the extremes of the sample count. Either keep the board steady for the whole test run or move it around.

12. What is the minimum observed total sample count? What input conditions caused it?
Approximate: Steady board (see 11 above).
13. What is the maximum observed total sample count? What input conditions caused it?
Approximate: 1940, 1836, 1766, 1818, 1822, 1822.
14. What is the difference between the minimum and maximum?
Approximate: 1940 – 115 = 1825.

- Let's try to be extremely consistent with test conditions to determine repeatability. Place the board on a stable, steady object. Run the program ten times and record each total sample count.

15. What are the minimum and maximum total observed sample counts? What is the difference between them?
    Approximate: 215-113 = 102

SIMULATING INPUT DATA

The accelerometer is so sensitive that it is essentially impossible to duplicate the physical conditions exactly. This will affect the execution time and therefore the profile measured and may mislead us. So, let's simulate the input data.

- Change the definition of SIM_ACCEL_XYZ_DATA from 0 to 1. Change SIM_ACCEL_STEP 0.01. Examine the code affected by these changes.

16. What does this do to the code?
    It makes the code call Simulate_XYZ_Data() which overwrites the acc_X, acc_Y and acc_Z variables with values which simulate the board slowly rotating. Each call to Simulate_XYZ_Data simulates rotating the board counterclockwise by of 0.01 radians.
17. Will the program still read acceleration data from the accelerometer?
    Yes, but that data will be overwritten in Simulate_XYZ_Data.

- Rebuild the program (three times) and download it. Then run the program several times and observe the extremes of the sample count. Either keep the board steady for the whole test run or move it around. Either way, the sample count should be consistent and steady.

18. What is the minimum observed total sample count? What input conditions caused it?
    1249
19. What is the maximum observed total sample count? What input conditions caused it?
    1251
20. What is the difference between the minimum and maximum?
    2

- Continue to use simulated input data for these experiments (keep SIM_ACCEL_XYZ_DATA as 1) until instructed otherwise.

EVALUATING PROFILE

- Rerun the program.
- Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*1249*
Top Function	*LCD_24S_Write_*	*512*
2nd Function	*LCD_Write_Rect*	*217*
3rd Function	*LCD_Text_Print*	*124*

21. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    12.49 ms

ECE 561 ONLY: LAZY UPDATES: TEXT VS. GRAPHICS BREAK-DOWN

Let's see about how much time text updates or graphics updates take compared with the total program time (aka sample count) from the Evaluating Profile section above. We'll do this by disabling parts of the code.

- Disable the call to Draw_Indicator(roll) in the main loop by redefining UPDATE_GRAPHICS as 0. Rebuild, download and run the program.

22. What is the total sample count? We can call this $N_{TextUpdates}$.
752

- Re-enable the call to Draw_Indicator(roll) in the main loop by redefining UPDATE_GRAPHICS as 1. Disable the text update code in the main loop by redefining UPDATE_TEXT as 0. Rebuild, download and run the program.

23. What is the total sample count? We can call this $N_{GraphicsUpdates}$.
597

- Disable both updates by redefining UPDATE_GRAPHICS and UPDATE_TEXT as 0. Rebuild, download and run the program.

24. What is the total sample count? We can call this $N_{Remainder}$.
103
25. Determine the percentage of time spent in the activities shown in the table below. Use the differences between these three total sample counts from this section and the total sample count of the previous section (we'll call it $N_{Complete}$). For example, text updates take ($N_{TextUpdates}$ - $N_{Remainder}$)/ $N_{Complete}$ of the program's time.

Program Activity	**Percentage of Total Samples = Percentage of Time**
Updating text	*(752-103)/1249 = 52.0%*
Updating graphics	*(597-103/1249 = 39.6%*
Remainder (everything else)	*103/1249 = 8.25%*

- Redefine UPDATE_GRAPHICS and UPDATE_TEXT to 1 to re-enable them.

ACCELERATING LCD UPDATES

The top few functions are associated with the LCD, so let's work on that first.

FASTER DATA WRITES ON LCD BUS

One optimization option speeds up data writes on the LCD bus.

- Change the definition of LCD_BUS_DEFAULTS_TO_DATA from 0 to 1.

26. Examine the code controlled by LCD_BUS_DEFAULTS_TO_DATA. Why should this change speed up the program?
The LCD bus signals (eight data lines and four control lines) are changed by software. One control line (data/not command, at position LCD_D_NC_POS) tells the LCD controller whether the byte on the data lines is to be interpreted as data (1) or a command (0). The code performs many more data writes than command writes.

The starter code always writes this signal, which takes time (at least one instruction). However, most of the operations are data writes, not command writes. The improved code leaves this signal configured for data (1). When writing data, the signal is left unchanged. However, before writing a command, the signal is changed to indicate a command (0), the write is performed, and then the signal is changed back to data (1), as it's very likely the next write will be data.

- Build the program (three times) and download it. Run the program three times.

27. What are the minimum and maximum total observed sample counts? What is the difference between them?
    902, 898, 898 898 898 898. Max diff is 4.

Because we are using simulated accelerometer data, the sample count should be relatively consistent regardless of board orientation.

28. Enter the profile information from the last run below.

Item	**Name**	**Sample Count**
Total Samples	n/a	*898*
Top Function	*LCD_Write_Rect*	*285*
2nd Function	*LCD_Plot_Pixel*	*224*
3rd Function	*LCD_Text_Print*	*145*

29. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    8.98 ms.

ELIMINATE LCD_PLOT_PIXEL

There should be the function LCD_Plot_Pixel in the profile. We learned in class that plotting one pixel at a time is very inefficient because of the set-up overhead.

- Change the definition of DRAW_RUNS_AS_RECTANGLES from 0 to 1.

30. Examine the code controlled by DRAW_RUNS_AS_RECTANGLES. Why should this change speed up the program?
    The LCD_Draw_Line function draws a line between two endpoints. Some lines consist of one or more multi-pixel segments ("runs") where all of the segment's pixels are on the same row (or column). The starter code draws each pixel with a call to LCD_Plot_Pixel. That function uses multiple writes to the LCD controller to specify the command, the pixel's addresses and its color data.
    The improved code draws each run as a very narrow rectangle (horizontal or vertical). This takes advantage of the LCD controller's ability to place data for successive pixels in successive memory locations, avoiding the need to send additional commands and pixel addresses.

- Build the program (three times) and download it. Run the program three times.

31. What are the minimum and maximum total observed sample counts? What is the difference between them?
    809 809 810. Max diff is 1.

Because we are using simulated accelerometer data, the sample count should be relatively consistent regardless of board orientation.

32. Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*810*
Top Function	*LCD_Write_Rect*	*294*
2nd Function	*LCD_Fill_Recta*	*153*
3rd Function	*LCD_Text_Print*	*149*

33. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    8.98 ms.

USE RUNS IN CHARACTER BITMAPS

Some of the LCD accesses come from updating the LCD with text. The function LCD_Text_PrintChar in LCD_Text.c draws a character (or other symbol) on the LCD using the LCD_Start_Rectangle and LCD_Write_Rectangle_Pixel functions. It determines whether a pixel should be the foreground or background color by using bitmap for the character. You can find more information in the **Speed – Shield Optimizations** slides and lecture.

- Change the definition of USE_TEXT_BITMAP_RUNS from 0 to 1.

34. Examine the code controlled by USE_TEXT_BITMAP_RUNS. Why should this change speed up the program?
    Both version of the code use the LCD_Start_Rectangle and LCD_Write_Rectangle_Pixel functions to minimize pixel addressing overhead.
    The starter code sends one pixel of each character at a time. The improved code looks at the bitmap for runs of pixels of the same value (foreground or background). Each run is drawn with a single call to LCD_Write_Rectangle_Pixel with an argument of how many times to write the pixel. This reduces function calling/return overhead, and the number of times it takes to step through the bitmap horizontally.

- Build the program (three times) and download it. Run the program three times.

35. What are the minimum and maximum total observed sample counts? What is the difference between them?
    696 698 698. Max diff is 2.
36. Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*698*
Top Function	*LCD_Write_Rect*	*230*
2nd Function	*LCD_Fill_Recta*	*168*
3rd Function	*LCD_Text_Print*	*108*

37. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    6.98 ms

TOOL OPTIMIZATION SETTINGS

At this point, we realize that there are some optimization settings we forgot to try before diving into the code. So let's try them now.

- Open the target options (Alt-F7) and select the C/C++ tab. Add --fpmode=fast to the Misc Controls box and press OK.
- Build the program (three times) and download it. Run the program three times.

38. What are the minimum and maximum total observed sample counts? What is the difference between them?
    682 686 686. Max diff is 4.
39. Did this change help speed up the program?
    A little, but not much.

- Open the target options (Alt-F7) and select the Target tab. Check the Use MicroLIB box and press OK.
- Build the program (three times) and download it. Run the program three times.

40. What are the minimum and maximum total observed sample counts? What is the difference between them?
41. Did this change help speed up the program?

- Open the target options (Alt-F7) and select the Target tab. Check the Use Cross-Module Optimization box and press OK. Cross-module optimization uses the output one compiler/linker pass to guide the next pass, often resulting in better optimization due to the tools having additional information.

42. Build the program (three times) and download it. Each build will take much more time because the compile/link process will be repeated several times.

- Run the program three times.

43. What are the minimum and maximum total observed sample counts? What is the difference between them?
44. Did this change help speed up the program?

Cross-module optimization and MicroLIB don't seem to help this program, so don't use them. Clear their checkboxes in the Target Options Target tab.

ACCELERATING I2C COMMUNICATION

CHECK DATA RATE

- Open the latest datasheet for the accelerometer **MMA8451Q-rev10.pdf** (located in the lab's References directory). The communication speed of I^{2}C communications is determined by the SCL signal's frequency. to find the maximum frequency of SCL.

45. Look at Table 4 (I^{2}C Slave Timing Values). What is the maximum SCL clock frequency?
    400 kHz

- The I2C peripheral's SCL clock frequency is set by dividing the bus clock signal by the SCL divider. In the **KL25 Subfamily Reference Manual.pdf**, Table 38-41 shows how the ICR value (first column) determines the values of the SCL divider (second column) and other fields. You can calculate the I^{2}C baud rate using the equation in Section 38.3.2. The bus speed (clock frequency) is 24 MHz. Note that there is a hardware bug in the MCU's I2C peripheral, so the I2C_F MULT field must be kept at 00 (for a mul value of 1). Other MULT values may cause communication failure.

46. Examine the function i2c_init. What numerical value is loaded into ICR?
    0x20 = 32

47. What is the I^2C baud rate resulting from this ICR valule?
    ICR 0x20 (32) results in SCL divider of 160
    *I2C baud rate = 24 MHz/(1*160) = 150 kHz*
48. Which SCL divider will get us closest to the maximum SCL frequency for the chip?
    Need divider of 24 MHz/400 kHz = 60. Closest safe available value is 64.
49. What SCL frequency do you expect, given that divider value?
    24 MHz/64 = 375 kHz
50. The SCL divider is selected by the ICR field of the peripheral's F register. What ICR value is needed for that SCL divider?
    Valid divider = 0x12 (18) for maximum safe SCL frequency.

- Change initialization code in the function **i2c_init** (in i2c.c) to use this new ICR value.

- We want to ensure I2C communications still work, so disable data simulation by defining SIM_ACCEL_XYZ_DATA as 0 again. Rebuild the program three times and download it. Run it and verify that it still works correctly. The display should be updated as you move the board.

51. Does the program still work correctly?
    Yes.

- We'll need to stabilize the input data to continue our timing analysis. Define SIM_ACCEL_XYZ_DATA as 1. Build the program (three times) and download it. Run the program three times.

52. What are the minimum and maximum total observed sample counts? What is the difference between them?
    642 640 640. 2 samples
53. Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*640*
Top Function	*LCD_Write_Rect*	*234*
2nd Function	*LCD_Fill_Recta*	*151*
3rd Function	*LCD_Text_Print*	*91*

54. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    6.4 ms

HOW MANY I2C TRANSACTIONS?

- Change the definition of READ_FULL_XYZ from 0 to 1 to switch to use the single I^2C message.

- We want to ensure I2C communications still work, so disable data simulation by defining SIM_ACCEL_XYZ_DATA as 0 again. Rebuild the program three times and download it. Run it and verify that it still works correctly. The display should be updated as you move the board.

55. Does the program still work correctly?
    Yes

- We'll need to stabilize the input data to continue our timing analysis. Define SIM_ACCEL_XYZ_DATA as 1. Build the program (three times) and download it. Run the program three times.

56. What are the minimum and maximum total observed sample counts? What is the difference between them?
    625 629 629. Max diff = 4 samples
57. Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*629*
Top Function	*LCD_Write_Rect*	*236*
2nd Function	*LCD_Fill_Recta*	*161*
3rd Function	*LCD_Text_Print*	*93*

58. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    6.29 ms

ECE 561 ONLY: PUSH DATA RATE FURTHER

A previous version of the datasheet for the accelerometer (**MMA8451Q-rev7.1.pdf**, in the References folder) suggests a higher I^2C clock speed is possible.

- Try pushing the clock frequency as high as possible (with the system still working) by reducing ICR and therefore increasing the frequency of SCL. We want to ensure I2C communications still work, so disable data simulation by defining SIM_ACCEL_XYZ_DATA as 0 again. Rebuild the program three times and download it. Run it and verify that it still works correctly. The display should be updated as you move the board.

59. What is the minimum working SCL divider?
    20
60. What is the maximum working SCL frequency?
    24 MHz/20 = 1.2 MHz
61. What is the ICR value for this frequency?
    0

- We'll need to stabilize the input data to continue our timing analysis. Define SIM_ACCEL_XYZ_DATA as 1. Build the program (three times) and download it. Run the program three times.

62. What are the minimum and maximum total observed sample counts? What is the difference between them?
    612 and 614, 2 samples.
63. Enter the profile information below.

Item	Name	Sample Count
Total Samples	n/a	*614*
Top Function	*LCD_Write_Rect*	*233*
2nd Function	*LCD_Fill_Recta*	*169*
3rd Function	*LCD_Text_Print*	*90*

64. What is the average time per loop (total samples * 1 ms/sample)/(NUM_TESTS)?
    6.14 ms