PROJECT 2 REPORT

Michael Meli
ECE 561

Optimizations – Inclination Measurement and Calculations

The following table shows the top five functions in execution time profile before optimizations:

Function Name	Ticks
i2c_wait	3143
__aeabi_dmul	2603
_double_epilogue	1191
__aeabi_ddiv	788
__aeabi_llsl	418

The following table shows the top five functions in execution time profile after optimizations:

Function Name	Ticks
i2c_wait	894
__aeabi_fdiv	401
i2c_read_setup	143
i2c_repeated_read	53
read_full_xyz	34

The following table explains the optimizations perform and the corresponding execution time improvements:

Optimization Performed	Profile Ticks (10,000 runs)	Improvement (in ticks)	Time Per Single Run (µs)
Initial (no optimization)	10418	------	1041.8
Force optimization for time and highest optimization level (O3) using compiler options	10415	3	1041.5
Set the --fpmode=fast (fast floating point mode) compiler option	6674	3741	667.4
Change the calculation of atan2() to atan2f(), which operates on floating point data instead of converting to double	3328	3346	332.8

Changed the I^2C clock speed to 1.2MHz using a SCL divider of 20 (0x00) and	2051	1277	205.1
Square root approximation using online guide [4] given in the project spec	1992	59	199.2
Put parentheses around the expression “180/PI” to allow the compiler to precompute the value	1955	37	195.5
atan2 approximation using the polynomial approximations given by [1] and [2] in the project spec	1538	417	153.8

As the table above shows, the final runtime of an inclination measurement and calculation was approximately 153.8 microseconds.

The top five optimizations were:

1. Setting --fpmode=fast. This is a compiler option that tells the compiler to try and replace double precision with single precision and try to make more aggressive optimizations on floating-point data.
2. Changing the calculation that used atan2 to atan2f. Since the data sent into this function were floating point, time would be spend converting this data to double, would then be operated on as doubles, and then reconverted back to float. Single precision is sufficient here, so changing to atan2f removed double precision math.
3. Increasing the I^2C baud rate. Much of the time of execution was spent busy waiting for I^2C transmissions and receives. Increasing the clock speed of the I^2C allowed this process to take less time, which reduced execution time.
4. Implementing an atan2 approximation. Atan2f was a relatively fast function, but was more accurate than was really necessary. The approximation used here was a polynomial approximation using floating point data, which was extremely fast and accurate enough.
5. Implementing a square root approximation. The square root calculation was done once per read/calculation. The IEEE single-precision floating point format is compatible with the operation $\sqrt{x} = 2^{\frac{\log_2 x}{2}}$, which allowed for an accurate and quick approximation.

Optimizations – Magnetometer Calculations to Determine Tilt-Compensated Heading

The following table shows the top five functions in execution time profile before optimizations:

Function Name	Ticks
__aeabi_dmul	10203
__aeabi_dadd	2313
_double_epilogue	1469
__aeabi_llsl	1415
__aeabi_ddiv	1346

The following table shows the top five functions in execution time profile after optimizations:

Function Name	Ticks
__aeabi_fmul	295
__aeabi_fadd	147
__aeabi_fdiv	101
sin_approx	93
calc_tilt_comp_heading	93

The following table explains the optimizations perform and the corresponding execution time improvements:

Optimization Performed	Profile Ticks (10,000 runs)	Improvement (in ticks)	Time Per Single Run (µs)
Initial (only optimizations were setting the –O3 highest optimization flag and enabling optimize for time)	8750	------	875
Set the --fpmode=fast (fast floating point mode) compiler option	2098	6652	209.8
Change the calculation of cos() and sin() to cosf() and sinf(), which operate on floating point data instead of converting to double	2056	42	205.6

Reused values for repeated trigonometric functions to reduce recalculation of the same values	1666	390	166.6
atan2 approximation using the polynomial approximations given by [1] and [2] in the project spec	1032	634	103.2
Implemented a cos approximation using a second degree Taylor Series polynomial	871	161	87.1
Implemented a sin approximation using a third degree Taylor Series polynomial	735	136	73.5

As the table above shows, the final runtime of a tilt-compensated heading calculation was approximately 73.5 microseconds.

The top five optimizations were:

1. Setting --fpmode=fast. This is a compiler option that tells the compiler to try and replace double precision with single precision and try to make more aggressive optimizations on floating-point data.
2. Implementing an atan2 approximation. Atan2f was a relatively fast function, but was more accurate than was really necessary. The approximation used here was a polynomial approximation using floating point data, which was extremely fast and accurate enough.
3. Reusing values for repeated trigonometric functions. In the calculation for tilt-compensated heading, there were a few sines and cosines of the same angle that were used multiple times. By storing these into a variable and then using the variable, I was able to reduce the number of intensive trigonometric calculations that needed to be performed.
4. Implementing a cosine approximation. The cosf math function is pretty fast and very accurate, but it is possible to get an “accurate enough” and fast approximation by implementing the Taylor Series polynomial $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$. The polynomial used was accurate mainly over the range $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, so I took advantage of the fact that cosine is a symmetrical function to fill in the rest of the possible values for x.
5. Implementing a sine approximation. Like I mentioned in #4 for cosine, the sinf math function is pretty fast and very accurate, but it is possible to get an “accurate enough” and fast approximation by implementing the Taylor Series polynomial $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$. The polynomial used was accurate only over the range $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, so I took advantage of the fact that sine is a symmetrical function to fill in the rest of the possible values for x.

Development Effort Tracking

Estimated person-hours required: 25 hours

Actual person-hours spent: 22 hours

A good deal of development time (around 10 hours) was spent attempting to fix the error where the I^2C bus would lock up at high baud rates. Another approximately 5 hours was spent attempting to set up SPI communications with the V2Xe compass, which was unsuccessful. The rest of the time was spent optimizing code for speed.