ECE-561

Project-2

BOARD INCLINATION AND COMPASS COMPENSATION

submitted by

Narasimha Sridhar Srirangam
nsriran@ncsu.edu
200084499

The motive of this project is to decrease the execution time taken to calculate the tilt of the freedom board and optimize time taken to calculate roll and pitch and also error compensation caused to a Compass reading due to this tilt, had it been tilted by same amount as the board.

TABLE SHOWING VARIOUS TIMINGS AFTER OPTIMIZATIONS.

Changes	Profile ticks	time taken per tick per reading in uSec
no changes	10360	1036
optimised ofr time, mode o-3	10317	1031.7
fpmode fast	6550	655
added f to pi and count	6507	650.7
changed tan2() to tan2f()	3327	332.7
changed ICR value in i2c.c from 12h to 0Ah	2453	245.3
changed tanf to sinf in pitch calculation	2215	221.5
changed ICR value in i2c.c from 12h to 02h	1812	181.2
Added heading and made optimizations	3400	340
changed sin and cos to fast processor functions	2916	291.6
removed sin and cos processor functions	3400	340
added look-up tables for sin and cos	2566	256.6
added i2c wait function with reset capability, and decreased ICR to 0x00	2157	215.7

OPTIMISATIONS

The initial project base had profile ticks of 10360 i.e the code being profiled took 1036uSecs to execute.
The target was to decrease this time to 200uSecs.
The code being profiled read the accelerometer value and converts it to roll and pitch.
The following are the list of optimizations and their affects.

1. Optimised for time,mode o-3.
   In the Project settings,under the C/C++, select the optimize for time tab, and select o-3 level of optimization. When this is used, the compiler optimizations are applied.

2. fpmode fast
   In the Project settings,under the C/C++, in MISC Controls, enter —fpmode fast.
   This performs more aggressive optimizations on floating point operations.This causes loss join accuracy but decreases execution time.

3. Initially Pi was defined as 3.14259265 and counts_per_g as 16384.0
   This is modified by adding the letter “f” to the end of both definitions.
   This forces the calculations using these variables to be single floating point operations.

4. Defining (180/Pi) as a constant and multiplying with it, instead of using the expression (180/pi) in every equation. Although trivial, this change in expressions does speedup the execution. Before this change, (180/Pi) was multiplied several times in the code. Instead of considering this as a constant, the processor computes its value every time it encounters this expression.This leads to unnecessary computations. Instead, if (180/Pi) is defined (#define) as a constant, and this value is substituted in the expressions, this extra computation is bypassed.

5. Instead of dividing with Counts_Per_G, we can multiply with (1/Counts_per_g) which can be defined explicitly using #define. This brings minor but noticeable speedup.

6. Changing The Accelerometer to fast read mode. This decreases the ADC resolution from 14bits to 8bits. The default resolution of the ADC is 14 bits .i.e each of accX, accY and accZ are 14 bits each. The processor communicated with ADC using i2 protocol. The data from ADC is read by reading single bytes or repeated reading of single bytes. Since the original resolution is 14 bits , the data won’t fit in 8bits(1 byte). Hence 2 bytes are used for each 14bit value and zeroes are appended t the left. We need 3 values from the ADC namely X,Y and Z accelerations. Since each is 2 bytes each, we need to read 6 bytes in total. When the MMA resolution is reduced to 8 bits, each of X,Y and Z are only 1byte now. They can be read in 3 cycles(1 cycle per byte), instead of the former 6 cycles. Hence this read code executes at twice the speed. Due to decrease in resolution, we lose accuracy by a small factor.

7. tan2() funtion is changed to tan2f() . This forces the Tan calculations to use single point float for calculations.

8. The ICR value is 0x12 in the default base. This is decreased in order to increase the I2C Communication speed. By decreasing this value, the divider for the bus clock(24Mhz) is decreased. As the Divider value is decreased, the clock to the I2C module increases

making the code run faster, since reading accelerometer is dependent on i2c speed.

9. Pitch was initially calculated using two argument Tan function. This is modified, to Sine function.

Heading calculations are added for profiling after this point. This function takes values for Xraw,Yraw and Zraw of magnetometer and calculates compensation according to accelerometer values, and finally, computes the Heading after tilt compensation.

1. The fast processor functions for sin and cos are used. These are specified by ARM and can increase the processing speed. Although these are used, they are later replaced by Look-up tables.

2. Look-up tables for sine and cosine are used to calculate values for trigonometric computations. When look-up tables are used, the value of argument to the function is located in the lookup table and its corresponding value is returned. i.e for every X in the domain of the table, there is a sin(X) value in the table. Since the processor accesses the table and retrieves the corresponding value, no calculations are involved. Time required is governed by the latency of the memory where the look-up is stored.

3. When i2c speed is increased to maximum value, the code gets stuck at the macro i2c wait.This causes the board to freeze at that statement. In order to prevent this from happening, we need to either run the i2c at slow and stable speeds or write a macro to reset i2c when it gets stuck. Since timing is important for this project, i2c is reset when it gets stuck. Due to this, the reading it was taking just before it got stuck is lost. But since we are doing multiple readings, this loss is compensated by multiple other readings, which occur at rapid intervals.
   The following is the methodology used for i2c reset.
   - A variable is initialised and placed in i2c wait function. There is a while loop in i2c wait which waits for the read or write to complete. The count variable is incremented for the duration the execution is inside this while loop.
   - Measure the count value over several tests.
   - Fix a threshold value for this count (200 for my project)
   - insert a IF condition inside the while loop to call a RESET function and return an error code if this count crosses the threshold.
   - Reset the i2c flags [2].
   - In all the functions that are calling the function i2c wait, implement an error check to start the read or write again if the error code is received. Reset the function also if required, which occurs in the case of repeated_read.
   - Hence, repeated_read returns an error code when it receives the i2c error code.
   - This error code from i2c propogates to read_full_xyz() function.
   - Implement a check function in read_full_xyz() to restart if error code is received.

The above implementation ensures that i2c wait doesn't execute longer than required.

Table showing top five functions in execution time profile

1- Initial Project Base. **Profile ticks - 10360**

Functions with top 5 values	
dmul	2631
i2c_repeated_read	2160
i2c_read_setup	1120
double_epilogue	1104
ddiv	766

2 - Changed optimization mode to O-3, and added —fpmode fast in misc. controls.
Profile Ticks - 6550

dmul	1624
i2c_repeated_read	1528
i2c_read_setup	949
__ARM_scalbn	444
fdiv	269

3 - Changed Pi, Counts_per_G to single point float values, and optimised their calculations using constants.
Profile Ticks - 6507

dmul	1535
i2c_repeated_read	1477
i2c_read_setup	1000
_double_epilogue	443
_dsqrt	350

4 - Changed trigonometric functions to use single float calculations
Profile ticks -3327

i2c_repeated_read	2731
i2c_read_setup	565
fmul	4

5 - Increasing i2c speed

Profile ticks - 1812

i2c_read_setup	537
i2c_repeated_read	463
fmul	253
fadd	201
_float_epilogue	92

Values obtained in Debugger mode

1 - Default Working

profile_ticks	1883	unsigned long
num_lost	31	unsigned int
roll	0.909380496	float
pitch	0	float
azi	357.813538	float
data[0]	0x0000	unsigned short

2 - Trigger 1 is asserted

roll	0	float
pitch	0	float
azi	299.635742	float

3 - Trigger 2 is asserted

roll	-7.58695269	float
pitch	-13.5547838	float
azi	273.357147	float

Development Effort

Phase	Estimated Time(hours)	Actual Time(hours)
Understanding Project Base code	2	3
Understanding I2C funtioning	2	2
Modifying I2C speed and testing	1	3
Making multiple changes in calculations	2	2
Understanding Accelerometer and changing resolution to 8 bytes	1	2
Understanding Heading calculation and coding	1	3
Sine and Cosine look-up tables coding and testing	2	2
I2C reset function- Research and Coding	1	3
Debugging and Testing	3	5
Soldering Headers on Board	0.5	1.5
Report	2	3
Total	17.5	29.5

References

[1] - Teaching Embedded System Design and Optimization with ARM Cortex-M0+ microcontrollers- By Dr. Alexander Dean.
http://www.cesr.ncsu.edu/agdean/Teaching/Dean%20ARM%20LiB%20FTF%20Presentation.pdf

[2] - Kinetis K10: Reset Module.
https://community.freescale.com/thread/320357

[3] - AN3192 Application note : Using LSM303DLH for a tilt compensated electronic compass

[4] -Tilt-Induced-Error Compensation for 2-Axis Magnetic Compass with 2-Axis Accelerometer
Li Xisheng, Kang Ruiqing, Shu Xiongying, Yu Guanghua ,University of Science and Technology Beijing, Beijing 100083, China

[5] -KL25 Sub-Family Reference Manual - By Freescale

[6] - MMA8451Q Reference Manual