EMBEDDED SYSTEM DESIGN
PROJECT 2: BOARD INCLINATION AND COMPASS COMPENSATION (V1.1)

OVERVIEW

For this project you will minimize the number of execution cycles needed to determine the Freedom board's inclination (roll and pitch) based on X, Y and Z acceleration components.

ECE 561 students will also interface with an electronic compass. Tilting the compass from a horizontal position introduces errors in its direction reading, so you will use the inclination information to compensate the direction.

ECE 461 AND 561

SYSTEM REQUIREMENTS

- Start with the KL25Z Freedom board and make modifications as needed.
- Determine the Inclination using the MMA8451 accelerometer. The X, Y and Z accelerations can be converted to roll and tilt angles (in degrees) using these equations:
  - roll = atan2(a_y, a_z)*180/π
  - pitch = atan2(a_x, (a_y^2 + a_z^2)$^{1/2}$)*180/ π

TESTING SUPPORT

TIME MEASUREMENT

Your program must indicate when it is actively performing computations by lighting one of the LEDs as shown below. When not active, your program must turn off the LEDs. The TAs will use these output signals to determine your code's execution time.

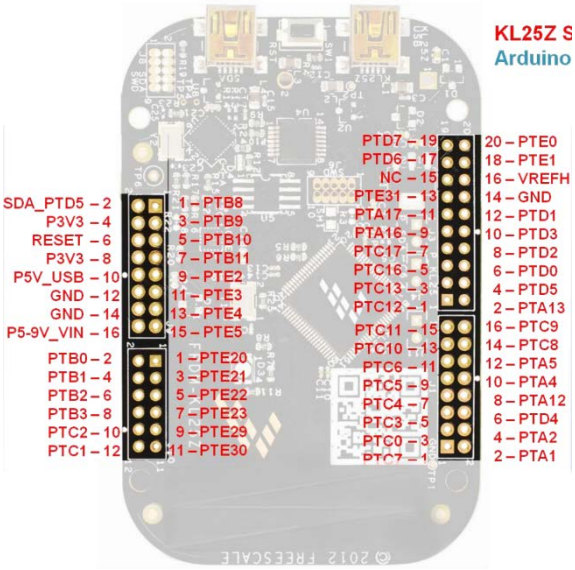
INPUTS AND OUTPUTS

In order to simplify testing, two input signals will be used to trigger program operation using specific fixed acceleration and/or magnetometer values. These constants will be defined in a header file provided to you for testing.

Use GPIO ports configured as inputs to implement triggers 1 and 2. Use the MCU's internal pull-up resistor for each port by setting PE=1 and PS=1 in that port bit's PCR. Connect the port bit to ground with a wire (solid, 22 gauge) in order to assert the input.

Use the following GPIO bit assignments:

		Port	Bit	Conn.	Pin	Test Pt.
In	**Trigger 1**	Port B	8	J9	1	
	Trigger 2	Port B	9	J9	3	
Out	**Red LED**	Port B	18			TP14
	Green LED	Port B	19			TP13
	Blue LED	Port D	1	J2	12	TP17



PROGRAM STRUCTURE

By default, your program should repeatedly read the accelerometer and compute the current roll and pitch. For ECE 561, your program should also read the compass and compute the tilt-compensated direction.

Your program must also respond to one or two input signals, polling these inputs while executing the default loop above. Interrupts are NOT recommended to detect the assertion of the triggers as that will complicate your development effort needlessly. Note that the trigger signals are active-low – they are asserted when 0.

ECE 461

Use the following general program structure in your main function:

```
while (1) {
   if trigger 2 asserted { // test case with fixed accel
      turn on blue LED
      set accelerometer readings to X_ACC_2, Y_ACC_2 and Z_ACC_2
      calculate roll and pitch
      turn off all LEDs
   } else { // regular operation
      turn on green LED
      read accelerometer
      calculate roll and pitch
      turn off all LEDs
   }
}
```

ECE 561

Use the following general program structure in your main function:

```
while (1) {
```

```
    if trigger 1 asserted { // test case with fixed mag values
       turn on red LED
       read accelerometer
       calculate roll and pitch
       set magnetometer readings to X_M_1, Y_M_1 and Z_M_1
       calculate tilt-compensated direction
       turn off all LEDs
    } else if trigger 2 asserted { // test case with fixed accel and mag
       turn on blue LED
       set accelerometer readings to X_ACC_2, Y_ACC_2 and Z_ACC_2
       calculate roll and pitch
       set magnetometer readings to X_M_2, Y_M_2 and Z_M_2
       calculate tilt-compensated direction
       turn off all LEDs
    } else { // regular operation
       turn on green LED
       read accelerometer
       calculate roll and pitch
       read magnetometer
       calculate tilt-compensated direction
       turn off all LEDs
    }
}
```

DATA VARIABLES

Your code must update the following data variables within the while(1) loop shown above:

Variable	Type	Location	Comments
acc_X, acc_Y, acc_Z	int16_t	mma8451.c	already exist
roll, pitch (radians)	float	mma8451.c	already exist
mag_X, mag_Y, mag_Z	*your choice*	main.c	
raw_direction, corrected_direction	float	main.c	

TEST CONSTANTS

Your code must use some or all of the following constants for testing, based on whether you are completing the ECE 461 or 561 project requirements.

Case	Accelerometer			Magnetometer (µT)			Correct Heading
	X	**Y**	**Z**	**X**	**Y**	**Z**	
default (no magnetometer)	(use actual readings)			0.84	21.48	-42.6	357.761°
default (with magnetometer)	(use actual readings)			(use actual readings)			
Trigger 1:	(use actual readings)			X_M_1	Y_M_1	Z_M_1	299.635°
				16.98	9.66	-42.3	(assuming roll = pitch = 0°)
Trigger 2:	X_ACC_2	Y_ACC_2	Z_ACC_2	X_M_2	Y_M_2	Z_M_2	95.0859°
	3840	-2080	15616	-13.8	16.2	-42.42	

MAGNETOMETER TEST DATA

Direction (°)	X (µT)	Y (µT)	Z (µT)
357.761	0.84	21.48	-42.6
40.426	-13.8	16.2	-42.42
124.455	-16.44	-11.28	-41.22
203.847	7.32	-16.56	-42.36
299.635	16.98	9.66	-42.3

These data constants can be used for testing your code's correct operation.

These values were derived from an Android smartphone, in which the magnetometers are aligned so the X axis points out the right side of the device (90 clockwise from straight ahead). To determine the direction of the top of the phone, 90° was subtracted from the arctan2-calculated value. You will need to convert either these input values or results based on the axis orientation of the module you used, shown below.

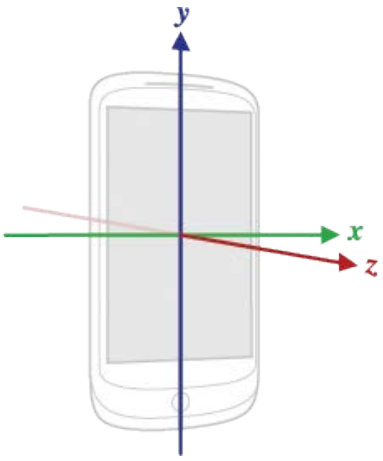


Figure 1. Android coordinate system (Android Open Source Project)

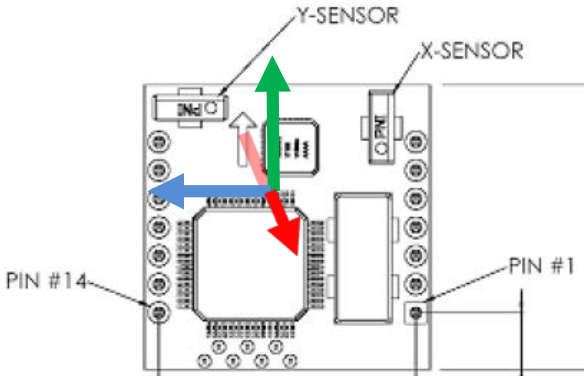


Figure 2. V2Xe sensor orientation. Red arrow is Z-axis, pointing up out of board (Courtesy PNI).

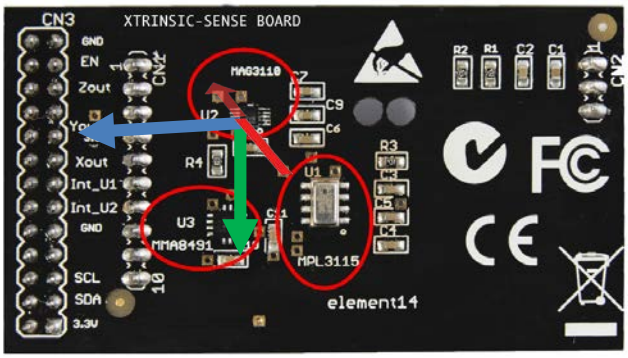


Figure 3. Xtrinsic Sense Board sensor orientation. Red arrow is Z-axis, pointing down into board (Courtesy Freescale and Element 14).

OPTIMIZATION PROCESS

Use the provided project 2 starter code (accelerometer I2C demonstration code with profiling support) as a starting point.

TOOLCHAIN TUNING

Tune the toolchain settings to reduce the execution cycle count. You should examine object code to ensure that the preferred operations are used, and that no extra operations exist.

- Select maximum speed optimization.
- Select the fast floating point math mode.
- Ensure single-precision floating point math operations are used, rather than double-precision.

POLYNOMIAL APPROXIMATIONS

Use a polynomial approximation for atan2.

- One approach is to use the two-argument atan2 approximation from Girones et al. [3].
- A different approach is to replace atan2 with an approximation of arctan [1] then convert from arctan to atan2 [2]. The following relationship of reciprocal arguments for arctan may be useful:
    - if x > 0, arctan(1/x) = π/2 – arctan(x)
    - if x < 0, arctan(1/x) = -π/2 – arctan(x)

SQUARE ROOT APPROXIMATION

Approximate the square root. The square root of x is the antilog of half of the log of x. So $\sqrt{x} = 2^{\frac{\log_2(x)}{2}}$. The IEEE single-precision floating point format is compatible with this operation; see the details online [4].

I2C COMMUNICATION OPTIMIZATION

Optimize the I^2C communication to reduce the execution cycles and time required.

- Refer to Chapter 38 of the KL25 Subfamily Reference Manual, which describes the IIC peripheral module in the MCU. Also consult the datasheet for the MMA8451Q inertial sensor and the schematic for the Freedom board.
- Run the I^2C communication links as fast as possible. Note that you can examine the SCL and SDA signals with an oscilloscope by probing the pull-up resistor connections.
- Evaluate how much time is spent waiting for I^2C transactions to occur. The existing code uses a macro (I2C_WAIT) to perform busy waiting until the IIC interrupt request flag is set. The functions i2c_read_setup, i2c_repeated_read, i2c_read_byte and i2c_write_byte call this macro. Replace this macro with a function so that the profiler shows this time explicitly.
- If the code spends a large amount of time performing busy waiting for the I^2C communication, consider using interrupts. You will need to restructure the code to perform this optimization, but it will free up time for other processing (or sleeping).

ECE 561

Write code to calculate compass direction using tilt compensation to improve its accuracy. Assume you have a three axis compass (magnetometer). Your goal is to optimize the time between receiving a READ command and calculating the tilt-compensated heading.

ST Micro Application Note AN3192 contains an excellent introduction to the concepts of electronic compasses, tilt compensation, and testing. However, equation 13 mixes units incorrectly, adding degrees to the radians returned by the arctan function. Freescale Application Note AN4248 is similar, targeting the Xtrinsic sensor board. These two application notes target three-axis compass modules. If you use the V2Xe module, which is only a two-axis module, refer to "Tilt-Induced-Error Compensation for 2-Axis Magnetic Compass with 2-Axis Accelerometer" by Li, Kang, Shu and, Yu.

OPTIMIZATION SUGGESTIONS

The following optimizations are suggested:

- Optimizations similar to those used for the inclination calculation above.
- Suggestions in AN3192 section 5 (Firmware implementation).
- Suggestions in AN4248 section 7 (Software implementation).
- Estimate inclination and/or heading with a filter (e.g. Kalman filter) based on periodic samples. Assume the system cannot rotate faster than 1440^o per second.

EXTRA CREDIT

Interface with a real compass module. Two types of compass are recommended, though you can use a different type if you prefer:

- PNI CompassPoint V2Xe 2 axis compass module using SPI communications. Limited quantities of these modules are available for students to borrow under a first-come, first-served basis.
- Xtrinsic Sensor Board three axis compass module. You can purchase one for about $15 (http://www.element14.com/community/docs/DOC-65084/l/xtrinsic-sensor-board#coverview).

Compass documentation is included in the project archive.

DELIVERABLES

- Report
  - Explanation of each step in optimization process
    - Text explaining optimization performed
    - Table showing top five functions in execution time profile
  - Development effort (time) tracking
    - Estimated person-hours required
    - Actual person-hours spent
- Source code.

GRADING

Your grade will depend on these factors:

- Functionality
- Accuracy of readings
  - The maximum pitch or roll error allowed for full credit is 1^{o}, or 0.01745 radians
- Quality of report
- Execution time targets:
  - **Inclination measurement and calculation**: 200 µs. This includes the time to read the accelerometer and then calculate the roll and pitch. You will receive full credit for the execution time portion of the project if you can achieve this target time.
  - **Magnetometer calculations to determine the tilt-compensated heading:** 80 µs. This does **not** include the time for communication with the magnetometer, but instead starts with the raw values which have been received from the magnetometer.

REFERENCES

[1] Rajan, S.; Sichun Wang; Inkol, R.; Joyal, A., "Efficient approximations for the arctangent function," *Signal Processing Magazine, IEEE* , vol.23, no.3, pp.108,111, May 2006
doi: 10.1109/MSP.2006.1628884
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1628884&isnumber=34166

[2] Wikipedia entry for atan2, http://en.wikipedia.org/wiki/Atan2

[3] Girones, X.; Julia, C.; Puig, D., "Full Quadrant Approximations for the Arctangent Function [Tips and Tricks]," *Signal Processing Magazine, IEEE* , vol.30, no.1, pp.130,135, Jan. 2013 doi: 10.1109/MSP.2012.2219677
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6375931&isnumber=6375903

[4] Wikipedia entry for square root approximation based on IEEE representation, http://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Approximations_that_depend_on_IEEE_representation