

Lab 6 Course Project

Timeline & Grading

- **Pre-proposal:** Friday, April 14th at 5:00 PM on CMS, **groups of 2**
 - **Meeting with a TA mentor:** April 17th – 19th
 - **Proposal:** Monday, April 24th at 11:59 PM on CMS
 - **Code and Project report:** Wednesday, May 17th at 4:30 PM on CMS
 - **Live demonstration:** Friday, May 19th
 - **Grading:** 11% of total grade
-

Section I: Overview

In the final course project, your group can take an idea that we explored in the course (but haven't yet done as a lab) and extend it. You are free to choose your project topic. However, your project needs to be implemented in C (may include some assembly) on the FRDM K64F board.

Note that you only have 2-3 weeks to work on the project implementation. So please choose the project scope carefully. It will also be a good idea to organize your project to have a baseline that you are confident that your group can finish in time with potential extensions that you would like to have if time allows.

The following list shows some of the top projects from the previous year as examples.

- Internet-synchronized digital clock
 - <https://cornell.box.com/s/kjstmjqbmfe8ams8grdjpmxlvj6rry>
 - Capacitive touch KEYboard
 - <https://cornell.box.com/s/ecl1vwkpk2f7g7xzmhjvdl56c6jn9u02>
 - PENbot: Drawing robot
 - <https://cornell.box.com/s/ti1qz6kflnt9566udwexbf3yzt9pspex>
 - Rhythm game
 - <https://cornell.box.com/s/flmi3ey52tf0kx07o4ddyry79zyxgmte>
 - VR zombie game
 - <https://cornell.box.com/s/hzm3azbsxb4s98in6cg4yizriim7kmsr>
 - Internet-connected weather display & alarm
 - <https://cornell.box.com/s/1n3cxkn3u8ar1b3znqmp45e74pm70kdh>
 - Digital multi-tool using NeoPixel LEDs
 - <https://cornell.box.com/s/3wgzr6kftmbrugxdnuug02d3hcxxh1r>
-

Section II: Project Proposal

Part 1: Project Mentor

For the course project, we will assign a TA mentor for each group. The TA project mentor will evaluate and provide feedback for your project proposal, and also serve as a contact point for questions that you may have while working on the project.

You are expected to schedule a short (15-min) meeting with your TA mentor to discuss the project idea. Before the meeting, you need to submit a short pre-proposal that describes your project idea; the pre-proposal will be a preliminary draft of the full proposal explained below.

Part 2: Proposal

After discussing your project idea with the mentor, your group needs to write and submit a full proposal. The proposal should be submitted in the PDF format, and use 11pt font and single line spacing. The proposal needs to explain your group's project idea in enough detail for us to evaluate if the difficulty is appropriate (1-2 pages should be enough).

All proposals should include the following:

- 1) **Title**
- 2) **Project group** – Names and NetID of the team members
- 3) **Overview** – Description of the project. What are you planning to do? Provide the necessary background information.
- 4) **System Description** – An explanation of how the system will work.
- 5) **Major technical tasks and testing** – How do you plan to implement your system and test it?
- 6) **Development timeline with milestones** – How do you plan to finish the project in time?
- 7) **Work distribution plan** – How do you plan to split the work on the project?

As part of your project, you may create an *embedded system* with expanded sensing and/or actuation by adding extra hardware. **However**, you're on your own in terms of getting it done; you are responsible for purchasing hardware components and implementing the extension. If your project requires an additional hardware component, the proposal should also include information on the hardware extension.

- 8) A complete **Bill of Materials (BoM)** – include specific part numbers ("ADXL345 3-Axis digital accelerometer", not "a motor") and a catalog numbers from the source (Digitkey, Mouser, etc.) with quantities and pricing.
- 9) A **System Schematic Diagram** – Check that parts will work with the FRDM board (e.g. correct voltage/current ranges, compatible communication protocols, etc.), and ensure that there are adequate pins for your system (not every pin is brought off the board).

Section III: Project Report Guideline

PDF format. 8 pages maximum, 11pt font and single line spacing.

This is a suggested outline for the Lab 6 report. Whatever format you decide to use, you should cover all the points below.

1. Introduction

Give brief background information about your project. Describe what your final project does. What does it achieve?

2. System diagram

Include a block diagram of your software system. Briefly describe how the different components interact. Explain and justify any major architectural choices. Also explain and justify the use of interrupts, scheduling, real-time, etc.

3. Hardware description (if applicable)

Provide a bill of materials for your project. Include a schematic showing how any external hardware is connected.

4. Detailed software description

Describe the main routines that your code uses. Describe the main data structures that are used throughout your design. Describe the protocol used to communicate with all the actuators and/or sensors.

5. Testing

Explain your testing strategy and how you convinced yourself your implementation is correct. Include a brief description of any test cases you wrote and what they are intended to test.

6. Results and challenges

Did you achieve what you proposed? What was the most complicated part of your design? Describe how your final implementation differs from your project proposal (if it does). Describe any extra credit work you completed. What would you do differently next time?

7. Work distribution

How did you carry out the project? Identify the major components that constituted the project. How did you collaborate? In your own words, explain how you divided the work, how you communicated with each other, and whether/how everyone on the team had an opportunity to play an active role in all the major tasks.

8. References, software reuse

You must cite the source of any code you do not personally write. It is ok and expected to build from small sections of example code (including what we gave you), you just need to acknowledge the source. Please list any manuals, data sheets and other documents that you used for your project.

Section IV: Live Demonstration

During finals week, you will schedule a time to demonstrate your project. You will have 5 minutes to demonstrate your work, followed by a few minutes of question and answer. *You should ensure your project works before your scheduled demo time as there will be no time to debug.*

A good demo will include the following elements:

1. Discussion of whether your group achieved what you set out to do in the proposal
 2. Why the project was complex enough to be a final project
 3. Show that the project works as described. The code should be the same as what was submitted on CMS
 4. The project works reliably, even if somebody with no prior background (e.g. TA) tries to make it work
 5. Both group members understand what is going on and most of the code was done by the group instead of copying it from other sources
 6. It is clear that both partners contributed to the work
-

Section V: Resources

Part 1: Accelerometer Tutorial

1) Install the full Kinetis SDK

The files that you installed in Lab 0 only included the bare minimum for running code. Keil provides a much richer set of drivers for interfacing with everything from I2C devices and SPI devices to USB, Ethernet, and the SD card, as well as many useful libraries (eg a TCP/IP stack, and a FAT filesystem for the SD card slot). We need to install those additional libraries.



1. From uVision, open the Pack Installer (highlighted above)
2. In the left pane, select All Devices » Freescale » K60 Series » MK64FN1M0xx12 (as in Lab 0). Now, in the right pane, install the following components (these will take some time):
 - Keil::Kinetis_SDK_DFP
 - Keil::MDK-Middleware
 - Keil::ARM_Compiler
3. Close the Pack Installer. If you see a dialog asking you reload packs, click yes.

2) Create a new project

When creating the project, do not select the device you normally do. Instead, choose Freescale » K Series » K6x Ethernet Crypto MCUs » MK64FN1M0VLL12.

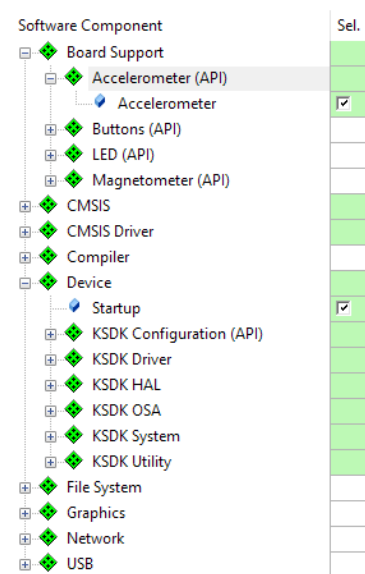
In the dialog that pops up next, you need to select two items:

- Board Support » Accelerometer (API) » Accelerometer
- Device » Startup

Once you've chosen those, click the Resolve button in the bottom left.

This will automatically select all of the other required libraries. You should now see something similar to the screenshot on the right.

Note: You may want to explore the options here, particularly under File System, Network, and USB. These contain extra libraries and drivers which may be useful to some projects.



3) Configure the linker

When the project is created, the compiler's linker (the program which decides how to place code and variables in memory) is not fully configured. Open the "Options for Target" dialog (the one with the debugger settings), and go to the Linker tab. Click the button with 3 dots next to the Scatter File field. Starting from the directory you created your project in, browse to RTE\Device\MK64FN1M0VLL12 and select MK64FN1M0xxx12_flash.

While the dialog is open, make sure you go to the Debug tab and select the CMSIS-DAP debugger. Unlike previous labs, however, you should not check the "Use MicroLIB" box or change the floating point hardware.

4) Configure the I/O peripherals

Table 5. Accelerometer and magnetometer signals connection

FXOS8700CQ	K64
SCL	PTE24/UART4_TX/I2C0_SCL/EWM_OUT_b
SDA	PTE25/UART4_RX/I2C0_SDA/EWM_IN
INT1	PTC6/SPI0_SOUT/PDB0_EXTRG/I2S0_RX_BCLK/FB_AD9/I2S0_MCLK/LLWU_P10
INT2	PTC13/UART4_CTS_b/FB_AD26

The libraries enabled for this project need to know what peripherals will be used, and how they should be configured. In the project browser, expand the Device tree, and open *RTE_Device.h* (it should be near the bottom). Click the Configuration Wizard tab at the bottom of the editor.

From the Configuration Wizard, expand the I2C0 entry. Check the box next to I2C0, then within I2C0 change I2C0_SDA to PTE25, and I2C0_SCL to PTE24. These settings were found in the

[FRDM-K64F User Manual](#) (the relevant table is shown above). The settings should look like this when you're done:

Option	Value
<input checked="" type="checkbox"/> ENET (10/100-Mbps Ethernet MAC) [Driver_ETH_MAC0]	<input type="checkbox"/>
<input checked="" type="checkbox"/> I2C0 (Inter-Integrated Circuit Interface 0) [Driver_I2C0]	<input checked="" type="checkbox"/>
I2C0_SDA	PTE25
I2C0_SCL	PTE24
DMA Rx	Disable
DMA Tx	Disable
<input type="checkbox"/> I2C1 (Inter-Integrated Circuit Interface 1) [Driver_I2C1]	<input type="checkbox"/>

5) Modify the hardware setup code

The default code does not enable the clock to PORTE, since the default settings don't use PORTE at all. Open *hardware_init.c* (it should be directly below *RTE_Device.h* in the project browser), and add `CLOCK_SYS_EnablePortClock(PORTE_IDX);` to *hardware_init()*. The code should look like the following when you're done:

```

35
36 void hardware_init(void) {
37
38     /* enable clock for PORTs */
39     CLOCK_SYS_EnablePortClock(PORTA_IDX);
40     CLOCK_SYS_EnablePortClock(PORTB_IDX);
41     CLOCK_SYS_EnablePortClock(PORTE_IDX);
42
43     /* Init board clock */
44     BOARD_ClockInit();
45     dbg_uart_init();
46 }
```

6) Write the main program

Create a new .c file (you can name it anything you want, main.c is a good choice), and put the following code in it:

```

1  #include <Board_Accelerometer.h>
2  #include <Board_Magnetometer.h>
3  #include <fsl_debug_console.h>
4  #include <board.h>
5
6  ACCELEROMETER_STATE state;
7  MAGNETOMETER_STATE mstate;
8
9  int main() {
10     hardware_init();
11     Accelerometer_Initialize();
12     Magnetometer_Initialize();
13
14     while (1) {
15         Accelerometer_GetState(&state);
16         Magnetometer_GetState(&mstate);
17         debug_printf("%5d %5d %5d %5d %5d %5d\r\n", state.x, state.y, state.z, mstate.x, mstate.y, mstate.z);
18     }
19 }
20
```

Board_Accelerometer.h and *Board_Magnetometer.h* contain the interface for the accelerometer/magnetometer chip. *fsl_debug_console.h* provides an interface for printing debug information to a computer over the UART, and *board.h* provides high level functions for the board.

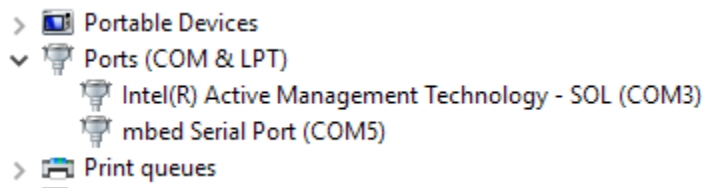
ACCELEROMETER_STATE and MAGNETOMETER_STATE are structures that the accelerometer driver uses to pass data back to your code. Each has an x, y, and z field, for the 3 axes that each sensor measures.

The `hardware_init` function configures some of the lower level parts of the board (sets up clocks, enables the GPIO ports, etc.). The I2C peripheral is automatically initialized by `Accelerometer_Initialize` and `Magnetometer_Initialize`.

In the loop, the code requests an update from the accelerometer and magnetometer using the `GetState` functions. The values are then sent to the computer using `debug_printf` (“\r\n” in the format string tell the computer to start a new line after the printf)

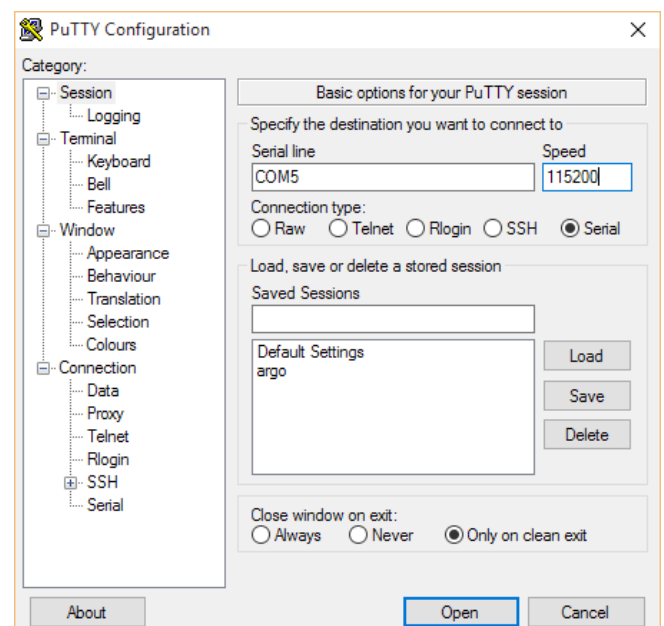
7) Test it

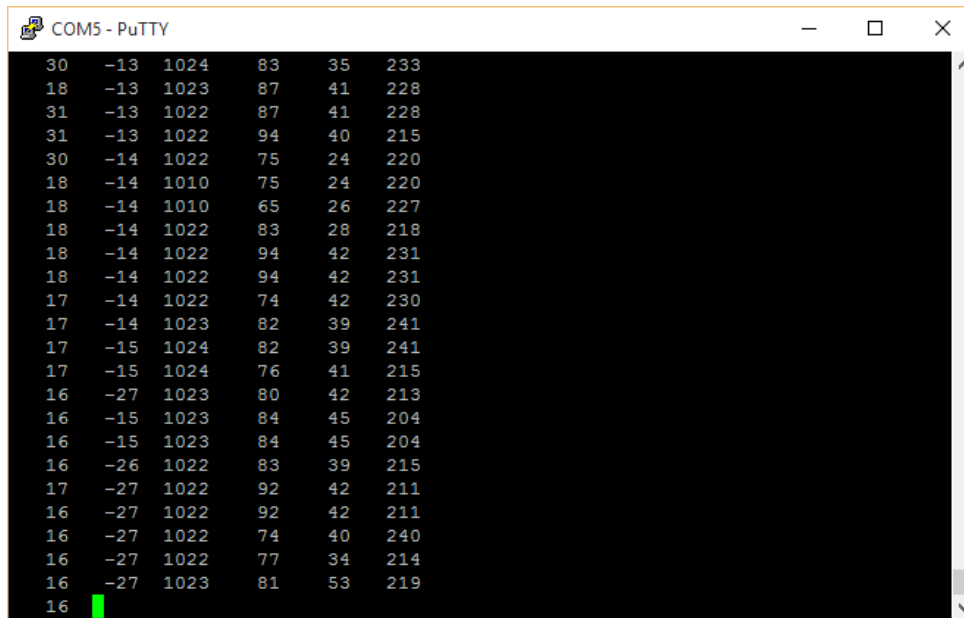
You can upload the program to your board as usual. To view the output, you’ll need to use a serial port monitor such as [PuTTY](#). You’ll also need to know the name of the serial port exposed by the board. To do this, open Device Manager (guide [here](#)), and expand the Ports section. You should see an “mbed Serial Port” with a port name, as shown below.



From PuTTY, select the Serial radio button, and put the port name you found in Device Manager in the Serial line box (COM5 in this example). Set Speed to 115200 (this is often called Baud Rate). Your setting should look something like what is shown to the right:

Click Open, and if everything worked correctly, you should see the accelerometer data in the console!





```
COM5 - PuTTY
30 -13 1024 83 35 233
18 -13 1023 87 41 228
31 -13 1022 87 41 228
31 -13 1022 94 40 215
30 -14 1022 75 24 220
18 -14 1010 75 24 220
18 -14 1010 65 26 227
18 -14 1022 83 28 218
18 -14 1022 94 42 231
18 -14 1022 94 42 231
17 -14 1022 74 42 230
17 -14 1023 82 39 241
17 -15 1024 82 39 241
17 -15 1024 76 41 215
16 -27 1023 80 42 213
16 -15 1023 84 45 204
16 -15 1023 84 45 204
16 -26 1022 83 39 215
17 -27 1022 92 42 211
16 -27 1022 92 42 211
16 -27 1022 74 40 240
16 -27 1022 77 34 214
16 -27 1023 81 53 219
16
```

Part 2: Additional Documentation

After doing step 1 in the tutorial, a number of other libraries that may be useful for your projects will be installed onto your computer. Here are some links to their documentation:

Kinetis SDK: [\[Click Here\]](#)

This is a low level library that provides a relatively simple abstraction over each of the peripherals on the microcontroller. It mostly just provides functions that make accessing registers in components such as the PIT simpler.

CMSIS-Driver: <http://www.keil.com/pack/doc/CMSIS/Driver/html/index.html>

This is a higher level abstraction for the communication interfaces (I2C, SPI, SD card, USB, Ethernet), designed to be a simpler interface that is shared across chips designed by different manufacturers. If your project involves interacting with another device over I2C or SPI, you probably want to use this library. The accelerometer library is built on top of this one.

MDK-Middleware: <http://www.keil.com/pack/doc/mw/General/html/index.html>

These are a series of high-level libraries built on top of CMSIS-Driver, which provide a very rich set of tools for your projects. Features include filesystems for the SD card, TCP/UDP stacks, HTTP/FTP/email servers/clients, and APIs for building USB devices, to name a few. The documentation also provides starter code examples for most of these features, which could be a good place to start for a lot of projects.