ECE3140 / CS3420 Embedded Systems
Lab 0: MCUXpresso Setup and Board Test

1 Introduction

This tutorial describes the basic setup of NXP's MCUXpresso to work with the FRDM-KL46Z which we will use throughout the class. You should be able to use this configuration for all the labs. If there are any setup differences, we will point them out in each lab's instructions. You can install and compile everything without having a board, but this document includes some additional instructions (not required to finish Lab 0) for which you need the board starting in Section 5 which describe how to run and debug programs.

MCUXpresso is based on Eclipse, which many of you might already know. More importantly, it runs on Windows, macOS, and Linux and offers debugging tools. We will only use relatively basic functionality in order to interact with the board.

1.1 ESD Warning!

It is winter. The air is dry, which makes static electricity a real concern when handling electronics. The board for this class is sensitive to Electro-static Sensitive Device (ESD), Wikipedia The board ships in a anti-static bag and you should store it in that bag. If you live in a carpeted area, try to ground yourself (e.g. touch your computer case) before handling. The board.

2 Installation

2.1 Install IDE

Download the latest version (11.9.0 as of this writing) of MCUXpresso from NXP. You can google it, or use the following this link this link to MCUXpresso. You'll need to make an account in order to get to the download page. Note: if you're a Linux user, the package should work on Debian-based systems, and most other distributions have support for .deb packages.

The installation process should also install all the necessary drivers needed to interact with the board, and depending on your security setting, you might get a few pop-up windows during the process confirming that you want to install drivers.

2.2 Install Board Packages

You're now almost ready to make a FRDM-KL46Z project. However, you need to download the specific board software development kit (SDK) before writing code. Through the top menu bar access:
"**Window**" → "**Show View**" → "**Other**" → "**MCUXpresso IDE**" → "**Install MCUXpresso SDKs**". Search for the board and install the FRDM-KL46Z SDK. If that worked, you can proceed to the next section. If you are not running the install with full permissions, this step can hang.

If the previous approach does not work for some reason you can install the SDK manually. You will have to visit the NXP SDK builder. Visit `https://mcuxpresso.nxp.com/en/select` and search for FRDM-KL46ZF. You don't need to check off boxes for any additional components at this time. The default SDK has all of the functionality we need. Once you have built and downloaded the FRDM-KL46Z SDK, navigate to the the view "**Installed SDKs**" as above, right click the window, and select the "Import Archive" option. Then navigate to the downloaded SDK, which should be a `.zip` file, and import it.

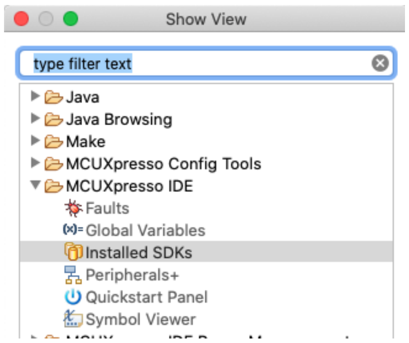
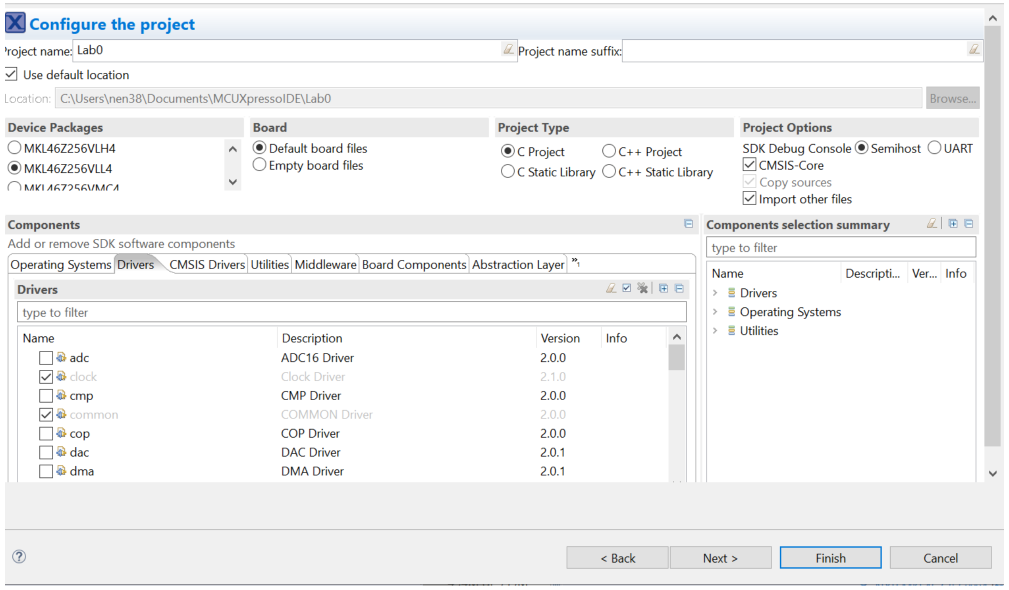


Figure 1: Finding the Installed SDKs view in "**Window**" → "**Show View**" → "**Other**". This is also how you would access "**Install MCUXpresso SDKs**" during the alternate install method.

3 Creating a project

Creating a basic project is straightforward. Create a new MCUXpresso C/C++ project. If you installed the FRDM-KL46Z SDK properly, you should be able to search for that board. Click on the FRDM-KL46Z board and click next. Give your project a name and change its directory if you want. Confirm that the "C project" radio button is filled. Semihost debug console is fine. Make sure CMSIS-Core is checked. Import other files. We will use the baremetal OS (that should be the default). You can leave the defaults for other drivers, utilities, and middleware. We won't use them. That page should look similar to this (depending on your OS):


Configure the project

Project name: Lab0 Project name suffix:

☑ Use default location

Location: C:\Users\nen38\Documents\MCUXpressoIDE\Lab0

Device Packages: ○ MKL46Z256VLH4 ● MKL46Z256VLL4 ○ MKL46Z256VMC4

Board: ● Default board files ○ Empty board files

Project Type: ● C Project ○ C++ Project ○ C Static Library ○ C++ Static Library

Project Options: SDK Debug Console ● Semihost ○ UART; ☑ CMSIS-Core; ☑ Copy sources; ☑ Import other files

Components — Add or remove SDK software components

Operating Systems | Drivers | CMSIS Drivers | Utilities | Middleware | Board Components | Abstraction Layer

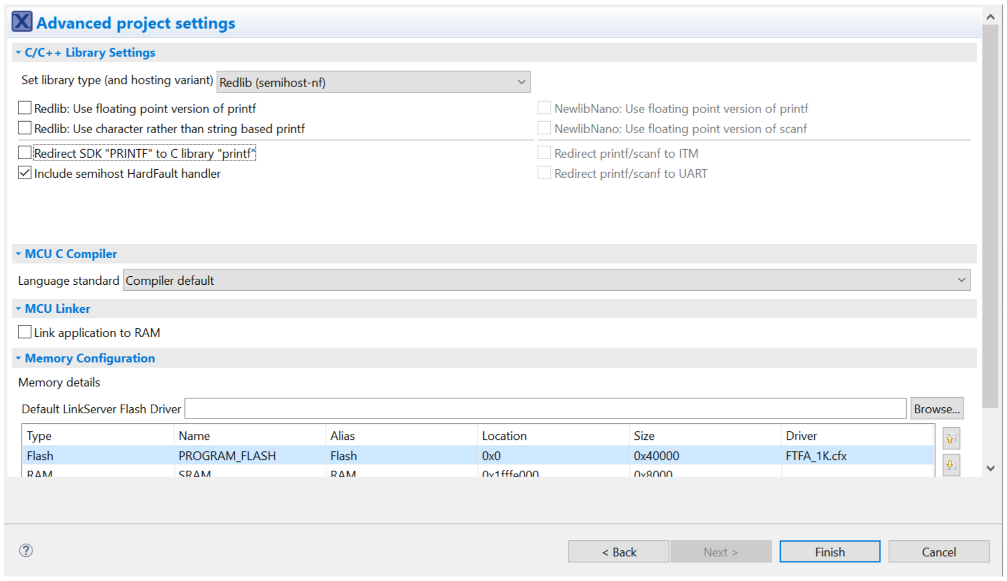
Drivers

	Name	Description	Version	Info
☐	adc	ADC16 Driver	2.0.0	
☑	clock	Clock Driver	2.1.0	
☐	cmp	CMP Driver	2.0.0	
☑	common	COMMON Driver	2.0.0	
☐	cop	COP Driver	2.0.0	
☐	dac	DAC Driver	2.0.1	
☐	dma	DMA Driver	2.0.1	

Components selection summary

Name	Descripti...	Ver...	Info
Drivers			
Operating Systems			
Utilities			

Hit next. Uncheck redirect SDK PRINTF to C library "printf". Hit finish.



Set library type (and hosting variant) Redlib (semihost-nf)

[ ] Redlib: Use floating point version of printf
[ ] Redlib: Use character rather than string based printf
[ ] Redirect SDK "PRINTF" to C library "printf"
[✓ Include semihost HardFault handler]
[ ] NewlibNano: Use floating point version of printf
[ ] NewlibNano: Use floating point version of scanf
[ ] Redirect printf/scanf to ITM
[ ] Redirect printf/scanf to UART

Language standard Compiler default

[ ] Link application to RAM

Default LinkServer Flash Driver

Type	Name	Alias	Location	Size	Driver
Flash	PROGRAM_FLASH	Flash	0x0	0x40000	FTFA_1K.cfx
RAM	SRAM	RAM	0x1fffe000	0x8000	

For now, start all of your MCUXpresso projects like this. You can add in more features as you need them. These settings are fine for the low-level stuff that we do in this class.

Now, find your new project in the Project Explorer. In the source folder, MCUXpresso generates three `.c` files in the source folder (mbt,semihost_hardfault,PROJECT_NAME). Delete them. Each lab will provide all the code you need.

Now, right click on the source folder and select "**Import**". Select "**General**" → "**Filesystem**". Find the directory with the `.c`, `.h`, and/or `s.` files for the lab. Import them. For this lab, you only need to import the assembly file `lab0.s`. Alternatively, you could just copy files directly to that project's source folder and refresh the project. Now, you can get to work. If you're having difficulty, make a Campuswire post with the MCUEXpresso tag.

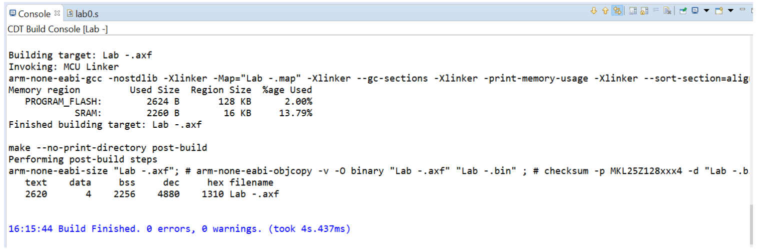
4 Working with Projects

4.1 GNU Assembly vs ARM Assembly

Note that there are a few different variants of ARM assembly, and we will use GNU the syntax. Keep this in mind if you are reading other resources and the syntax does not quite match what we are writing here. The instructions (opcodes) themselves are the same, but indentation and comments work differently. Within the GNU syntax there are also different versions, and the newest (unified) syntax is designed to work across many different processor types, however it is more picky and we will use a slightly older version that is easier to work with. (see comments in lab0.s)

4.2 Building

Once you have coded up your project, navigate to your project in the Project Explorer. Notice a hammer icon in the upper-left corner. Click that hammer icon to attempt to build your project. You should see something similar to the picture below:



```
Building target: Lab -.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -Xlinker -Map="Lab -.map" -Xlinker --gc-sections -Xlinker -print-memory-usage -Xlinker --sort-section=alig
Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:        2624 B       128 KB      2.00%
            SRAM:        2260 B        16 KB     13.79%
Finished building target: Lab -.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "Lab -.axf"; # arm-none-eabi-objcopy -v -O binary "Lab -.axf" "Lab -.bin" ; # checksum -p MKL25Z128xxx4 -d "Lab -.b
   text    data     bss     dec     hex filename
   2620       4    2256    4880    1310 Lab -.axf


16:15:44 Build Finished. 0 errors, 0 warnings. (took 4s.437ms)
```

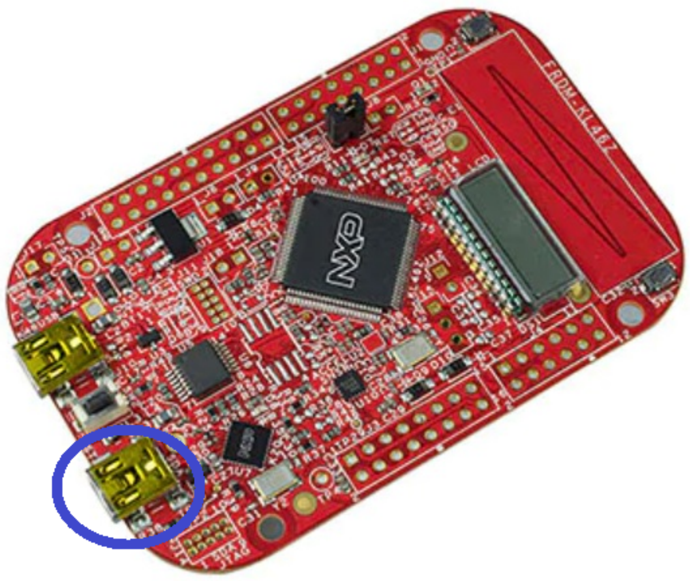
For example, the binary created for project as a size of 4880 bytes. This is how much space the program will take up on the flash memory. The reason this binary is so large compared to the small assembly file you wrote, is that the project automatically includes all the setup, startup procedures, and some drivers. Take a screenshot of the build window and upload it to CMS. Lab 0 is done! You should see how much memory your program used both in RAM and in FLASH.

5 Once You Have The Board

There are a number of things that you can do once you have the board, like uploading and debugging a program. We put them here so you can test that you setup works, although Lab 0 only asks you test the building your program.

5.1 Loading to the board

You can compile programs without having the board, but you will obviously need one in order to execute code. There are two USB-B mini ports on the FRDM-KL46Z. They are ***not*** interchangeable. For uploading code and debugging you should use the one that has `SDA` printed next to it.

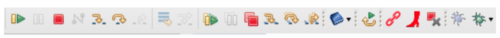


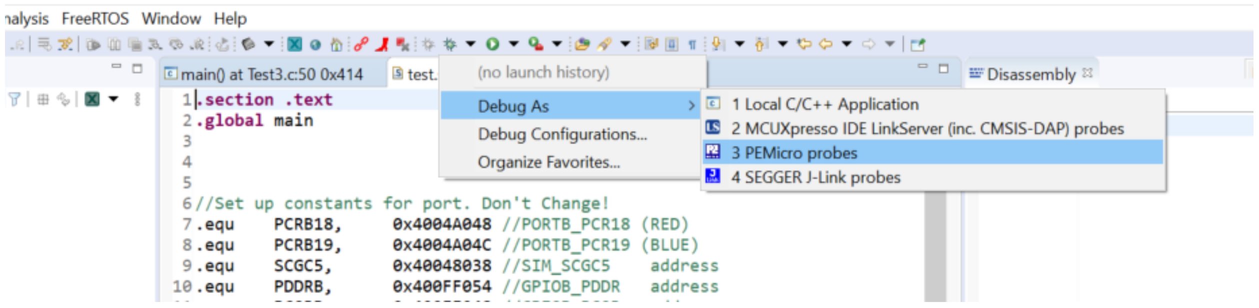
Click on your project folder in the Project Explorer. A chip icon should be filled-in. Click on that chip icon to open the GUI Flash Tool. MCUXpresso will look for your board and will display it as an OpenSDA probe by P&E Microcomputer Systems. If your board appears, click okay. You can confirm that the built binary file for your project is going to travel to the board. Click run. The board will receive your code and will try to run it immediately.

5.2 Debugging

Find the green bug icon in the toolbar. Click on the black triangle next to it and navigate to the following menu option if you are debugging this project for the first time. That option will create a debug configuration with sensible defaults.

When you start debugging, you'll see a toolbar that looks like this:




nalysis FreeRTOS Window Help
main() at Test3.c:50 0x414
test.
(no launch history)
Disassembly
Debug As
Debug Configurations...
Organize Favorites...
1 Local C/C++ Application
2 MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
3 PEMicro probes
4 SEGGER J-Link probes

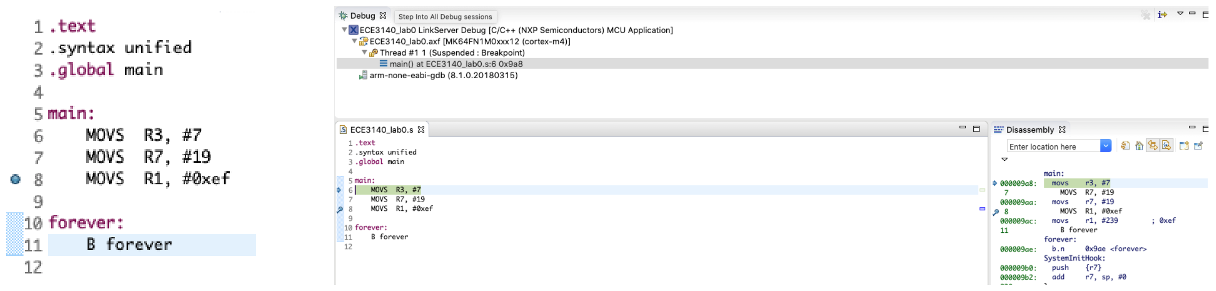
```
.section .text
.global main



//Set up constants for port. Don't Change!
.equ     PCRB18,     0x4004A048 //PORTB_PCR18 (RED)
.equ     PCRB19,     0x4004A04C //PORTB_PCR19 (BLUE)
.equ     SCGC5,      0x40048038 //SIM_SCGC5   address
.equ     PDDRB,      0x400FF054 //GPIOB_PDDR  address
```

Figure 2: Create a new debug configuration

Start execution from current breakpoint with . Pause with . Stop debugging and exit the debug session with .

5.2.1 Breakpoints

Set a breakpoint by double clicking in the area to the left of a line number in a file. Execution will pause immediately before executing that instruction or line of code. Stepping could go to the next C line executed or the next assembly instruction depending on the type of source. Press F5 or to step into something.



```
.text
.syntax unified
.global main

main:
    MOVS  R3, #7
    MOVS  R7, #19
    MOVS  R1, #0xef

forever:
    B forever

```

```
ECE3140_lab0 LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
  ECE3140_lab0.axf [MK64FN1M0xxx12 (cortex-m4)]
    Thread #1 1 (Suspended : Breakpoint)
      main() at ECE3140_lab0.s:6 0x9a8
  arm-none-eabi-gdb (8.1.0.20180315)
```

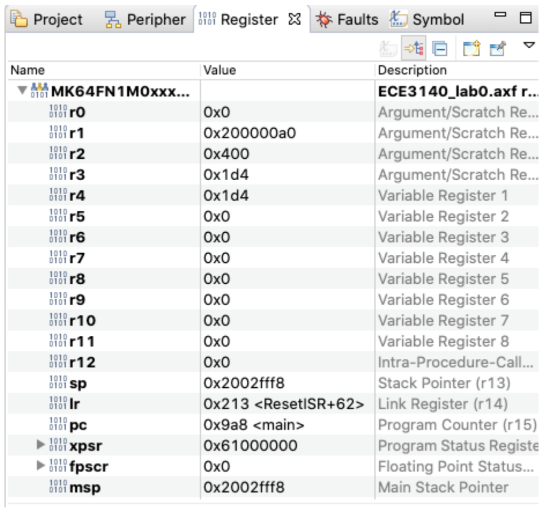
```
            main:
000009a8:     movs     r3, #7
7               MOVS  R7, #19
000009aa:     movs     r7, #19
8               MOVS  R1, #0xef
000009ac:     movs     r1, #239         ; 0xef
11              B forever
            forever:
000009ae:     b.n      0x9ae <forever>
            SystemInitHook:
000009b0:     push     {r7}
000009b2:     add      r7, sp, #0
```

Figure 3: (left) Setting a breakpoint in the source file. (right) Stepping through the code in a debug session. You can see both the source and the disassembled binary.

There are also options to step over things like loops, or to step return out of functions. Feel free to explore to find out what the rest of the icons do.

5.3 Viewing registers

On the left side of the window, there is a group of tabbed subwindows. Click on the register tab, and register information will appear if a debug session is active. You should be able to watch them change as you step through the code.



Name	Value	Description
MK64FN1M0xxx...		ECE3140_lab0.axf r...
r0	0x0	Argument/Scratch Re...
r1	0x200000a0	Argument/Scratch Re...
r2	0x400	Argument/Scratch Re...
r3	0x1d4	Argument/Scratch Re...
r4	0x1d4	Variable Register 1
r5	0x0	Variable Register 2
r6	0x0	Variable Register 3
r7	0x0	Variable Register 4
r8	0x0	Variable Register 5
r9	0x0	Variable Register 6
r10	0x0	Variable Register 7
r11	0x0	Variable Register 8
r12	0x0	Intra-Procedure-Call...
sp	0x2002fff8	Stack Pointer (r13)
lr	0x213 <ResetISR+62>	Link Register (r14)
pc	0x9a8 <main>	Program Counter (r15)
xpsr	0x61000000	Program Status Registe
fpscr	0x0	Floating Point Status...
msp	0x2002fff8	Main Stack Pointer

6 Running `lab0.s`

Include the provided `lab0.s` file in your projects sources and build it. Enter the debugging mode and watch the registers change as you step through the assembly instructions. The provided program is so small that you don't need to set any breakpoints. You can just step through the few instructions.

If you are curious or unclear about some of the assembly instructions we talk about in class, you can use the IDE to see what the assembly instructions do you your processor, by writing a program and then examining the registers and processor flags.

Happy hacking!