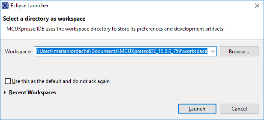
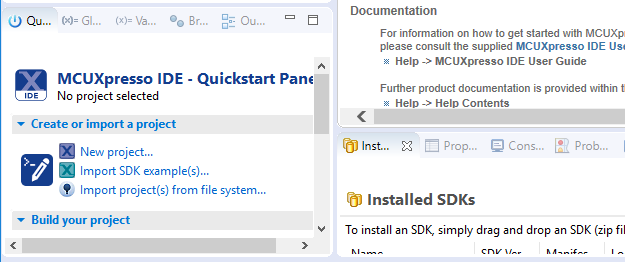
**MICROCONTROLLERS—LAB EXERCISE #1**

**INTRODUCTION TO THE SOFTWARE AND HARDWARE ENVIRONMENT**

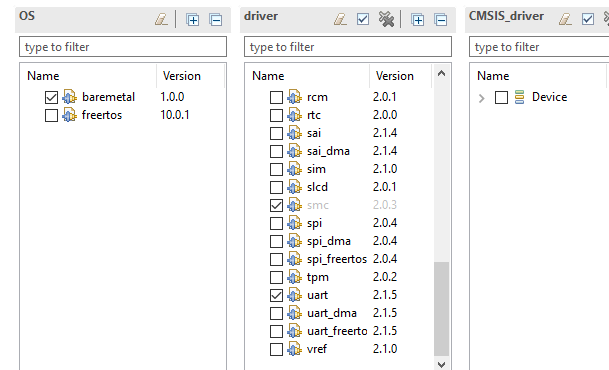
*Lead me in thy truth, and teach me: for thou* art *the God of my salvation; on thee do I wait all the day. (Psalms 25:5)*

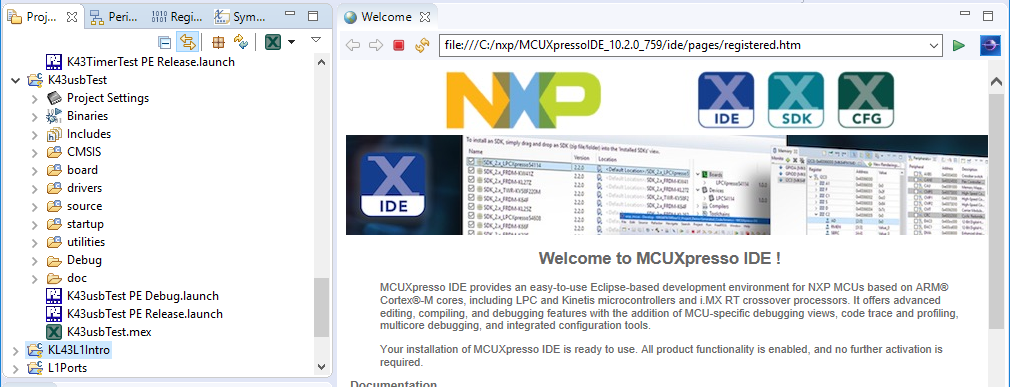
The goal of this assignment is to introduce MCUXpresso and the hardware that will be used in this lab. As you go through this lab assignment, do not worry if you find many things that you do not know. The material should become clear later in the semester.

1. Open MCUXpresso. A dialog box such as the following should appear.
2. If you already have a workspace, select it from the drop-down list or from *Recent Workspaces*.
3. If you do not have a workspace, select *Browse* and then the folder that will be your workspace.
4. If you work on a lab computer, do NOT select the option *Use this as default*.
5. Click *Launch*.
6. MCUXpresso is based on the Eclipse IDE (Integrated Development Environment). The IDE provides several *perspectives*. Two perspectives that will be used in this class are the *Develop* perspective and the *Debug* perspective.
7. Click the *open perspective* button to see available perspectives.
8. Select the *Debug* perspective.
9. Click the *open perspective* button again and select the *Develop* perspective.
10. Locate the *Quickstart Panel* tab on the lower left side of the window and select *New Project …*.
11. At this point the software will show all installed boards.
12. Select your board (FRDM-KL43Z).



1. Once the dialog window is updated, select *Next*.
2. Choose an appropriate name for your project (such as Lab1).
3. Add the slcd driver to your project by clicking its checkbox. (This is the driver of LCD on your board.)



1. Select *Finish*.
2. At this point your project name should appear in the *Project Explorer* tab located in the upper left side of the window. Click the > sign to see its folders and files. 
3. The project contains the files that will be used to build your application. These files are grouped in folders, according to their function.
4. Locate the file bearing the name of your project that is in the *source* folder.
5. Open that file (double-click it).
6. The default code of your file is sufficient for a simple test your board.
7. Note that the board has two USB connectors. One is reserved for debugging (the SDA USB port) and the other is for general purpose (available to the programmer for any use).
8. Identify the SDA USB jack.



1. Connect the SDA jack to the computer.
2. Note that the board is powered via the USB.
3. At this point, the board should run the last installed program (such as the factory preinstalled program).
4. Select *Build* in the *Quickstart Panel* tab. This will build the project without uploading it on the board.
5. Select *Debug* in the *Quickstart Panel* tab and click OK when the software reports finding the Open SDA probe. This will build (if necessary) the project and upload it on the microcontroller.
6. In case you get an error message stating that no probe is found, double-check that the USB cable is connected to the SDA jack.
7. If the software did not switch automatically to the *Debug* perspective, switch to the *Debug* perspective.
8. Note the following buttons:

Disconnect IDE from MCU board



Run

Pause (Suspend)

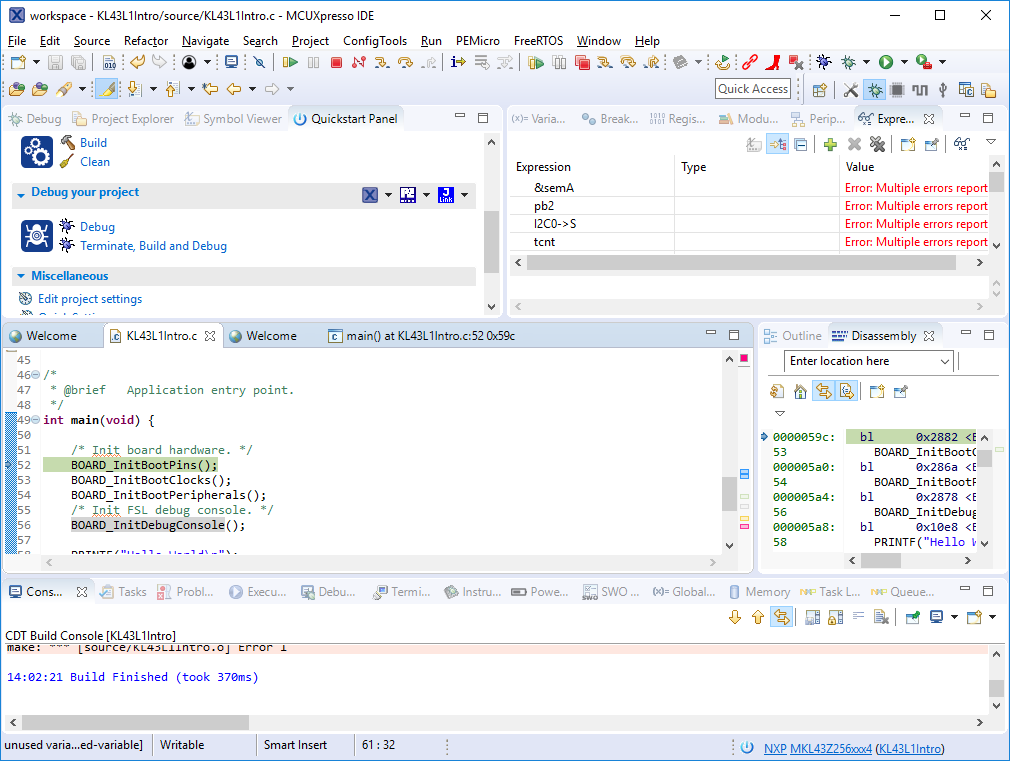
Terminate

Restart

1. The program should print “Hello World” in the console at the bottom of the screen and then enter an infinite loop.
2. Run the program and verify that it does what is supposed to.
3. Terminate the program.

**Errors and Warnings.**

1. Modify the program by replacing the i++ with j++. Do not change anything else!
2. Build the program. This should result in one error and one warning.
3. The window listing the source code could look like this:



Purple indicates errors. Yellow indicates warnings.

Click on purple region to highlight the line where the error is.

Click on yellow region to highlight the line that causes the warning.

1. Locate the lines that cause the error and the warning.
2. Place the mouse pointer over the exclamation mark to read what the error is or what the warning is.
3. Indicate what the error is: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Indicate what the warning is: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. Correct the program and verify that when you build it there are no errors and warnings.

**Locating Definitions within the Project.**

1. The project consists mainly of header files (they have the extension h) and definition files (they have the extension C).
2. Header files define data types and constants and declare predefined functions that are available to the programmer.
3. Some of the available functions are available in source form, and we can read their definition. In the case of other functions only a machine-readable form is available; their source code is not available.
4. Switch to the *Develop* perspective.
5. Locate the *Project Explorer* tab on the left-hand side.
6. Explore the folders of your project and locate the header file MKL43Z4.h.
7. The file MKL43Z4.h is in the folder \_\_\_\_\_\_\_\_\_.
8. The file MKL43Z4.h will be very useful in our class, as it contains the definitions required for writing a *readable* microcontroller program.
9. The IDE has search features allowing to find definitions and declarations.
10. Select *Search*, then *C/C++…* (or press Ctrl-H) and search for the definition of the function BOARD\_InitBootPins.
11. The C file containing it should appear at the bottom of the window in the *Search* tab.
12. The function is in the file \_\_\_\_\_\_\_\_\_\_
13. Double-click the file name to find where the function is.
14. Fill in the definition of the function:

**void** **BOARD\_InitBootPins**(**void**)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

**The Logic Analyzer**

1. The logic analyzer is an important instrument that can be used for testing and debugging programs.
2. Erase the main function of your program and replace it with the following (you may copy and paste the code).

**#define** ARSIZE 5 // the size of the following array

**const** uint32\_t data[] = {0, 1, 2, 3, 7};

**int** **main**(**void**) {

BOARD\_InitBootPins();

BOARD\_InitBootClocks();

BOARD\_InitBootPeripherals();

BOARD\_InitDebugConsole();

SIM->SCGC5 |= SIM\_SCGC5\_PORTB\_MASK; // enable port B clock

**int** i = 0b101;

**volatile** **int** j;

j = i + 1;

**for**(i = 0; i < 3; i ++) {

// enable pin i of port B

PORTB->PCR[i] &= ~PORT\_PCR\_MUX\_MASK;

PORTB->PCR[i] |= PORT\_PCR\_MUX(1);

}

GPIOB->PDDR |= 0xf; // PB3, PB2, PB1, PB0 should be outputs

GPIOE->PDDR |= 0b11111100u; // 7, 6, ..., 2: output bits

GPIOE->PSOR |= 0b10110100u; // set bits 7, 5, 4, and 2

GPIOE->PCOR |= 0b01001000u; // clear bits 6 and 3

j |= (GPIOC->PDIR & 0b101010u); // set bits of j if PDIR bits are set

j &= (GPIOC->PDIR | ~0b101010u); // clear bits of j if PDIR bits are clear

printf("Entering the infinite loop ...\n");

**for**(i = 0;;) {

// update the pins of port B

GPIOB->PSOR |= data[i] & 0xf;

GPIOB->PCOR |= (~data[i]) & 0xf;

j = FGPIOB->PDOR & 0xf;

// update the variable i

i = i + 1; // update i

i = i % ARSIZE; // reset i when it reaches the size of the data array

}

**return** 0;

}

1. The program outputs continuously the three-bit numbers 0, 1, 2, 3, 7, 0, 1, 2, 3, 7, 0, 1, 2, 3, 7, … on the pins PB0, PB1, PB2, and PB3. (Each pin outputs one of the four bits.)
2. Build the program.
3. Verify that there are no errors.
4. Run the program.
5. While the program is running, verify with the logic analyzer that you obtain the correct number sequence on the pins PB0, PB1, PB2, and PB3.

* Locate GND and the PB pins on the board (they are marked on the back of the board, in the *Quick Start Guide*, and in the figure below.
* Using jumper wires, connect the GND wire of the logic analyzer to GND and four logic channels to the four PB pins.
* Press the LA button of the oscilloscope to turn on the logic analyzer.
* Press the TRIGGER MENU button and select the Edge mode, rising edge, Auto sweep, and a channel source for which the image is stable (such as the channel on which you have the bit 2).

1. Demonstrate your work to the instructor.