**Getting Started with KSDK and USB Lab Guide**

**Introduction**

In this hands-on lab, we will use Kinetis Design Studio to build the KSDK and USB libraries, and then build and test an USB application example, and finally add some I2C driver code to read the on-board accelerometer

**Resources**

PC running Windows 7 with the following software:

• Kinetis Design Studio (KDS) v3.0

• Kinetis SDK 1.2

Hardware:

• FRDM-K22F

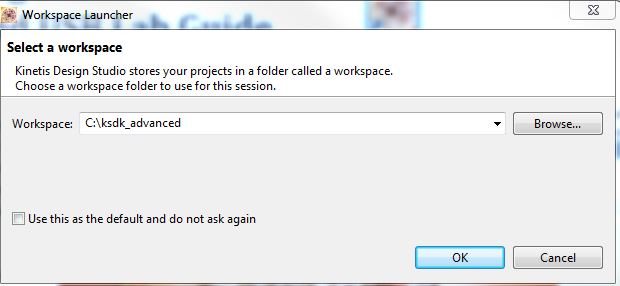
# Lab 1: Building and Running a KSDK USB Project with KDS

This lab will go through the steps to import a KSDK platform library and an example project, build it, and run it with the debugger.

* 1. Open KDS



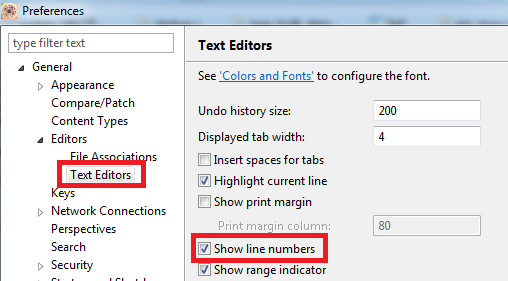
* 1. Set the workspace directory to **C:\ksdk\_advanced** (or your choice of other new directory location) and click on OK.

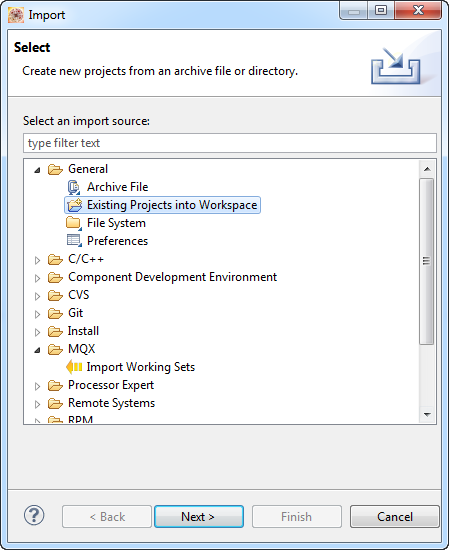


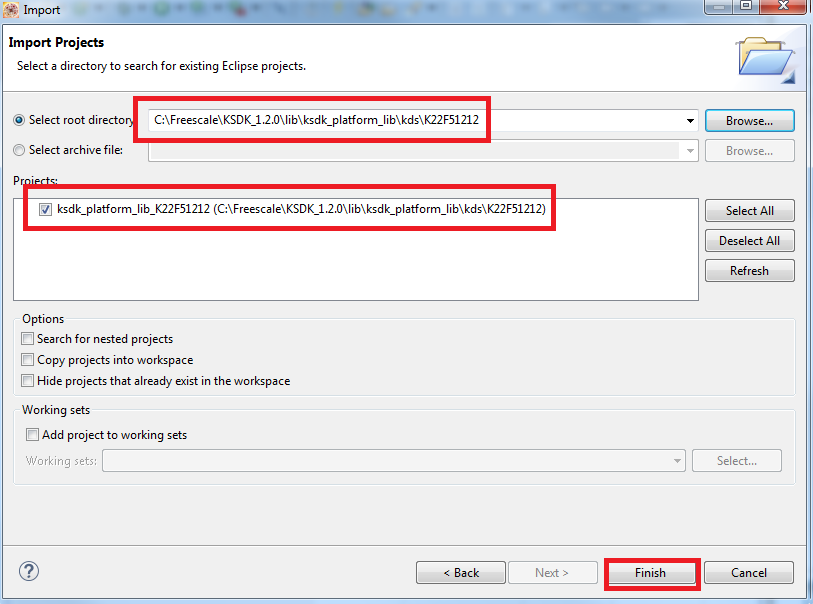
* 1. Click on the “Workbench” icon to go to the main Workbench screen. This only has to be done the first time a new workspace location is used.



* 1. Turn on line numbering in the KDS editor. Go to Window->Preferences. Then navigate down to General->Editors->Text Editors and check the box next to “Show Line Numbers”. It may already be checked if someone previously used the computer.



* 1. Next we will import the KSDK Platform library for the FRDM-K22F. This library contains the HAL and drivers provided by KSDK.
  2. Go to the KDS menu **File->Import**.
  3. In the General group, select **Existing Projects into Workspace**.  
     
  4. Browse to the root directory where the K22F platform library project is located: **C:\Freescale\KSDK\_1.2.0\lib\ksdk\_platform\_lib\kds\K22F51212**
  5. Select the project **ksdk\_platform\_lib\_K22F51212**. Click ‘Finish’.

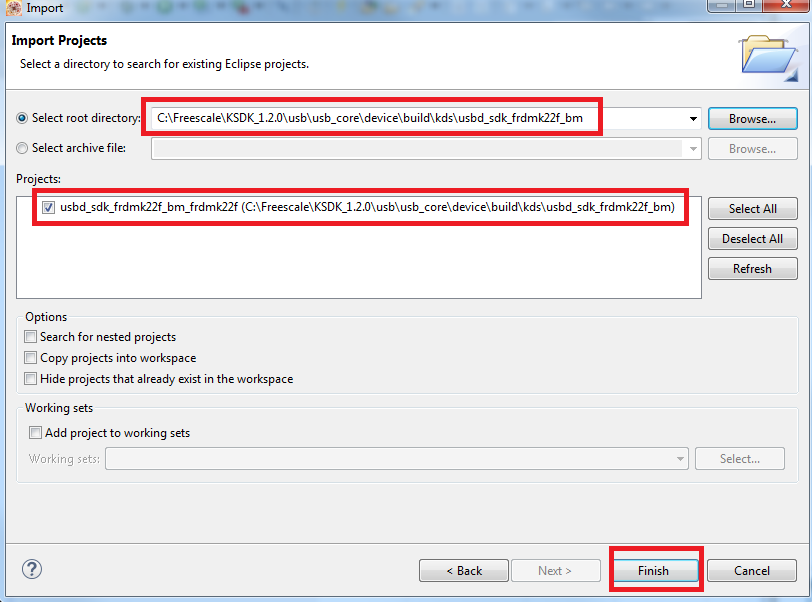


NOTE: Never check the box labeled “Copy projects into workspace” when importing KSDK or MQX projects. This will cause issues finding files when building the projects.

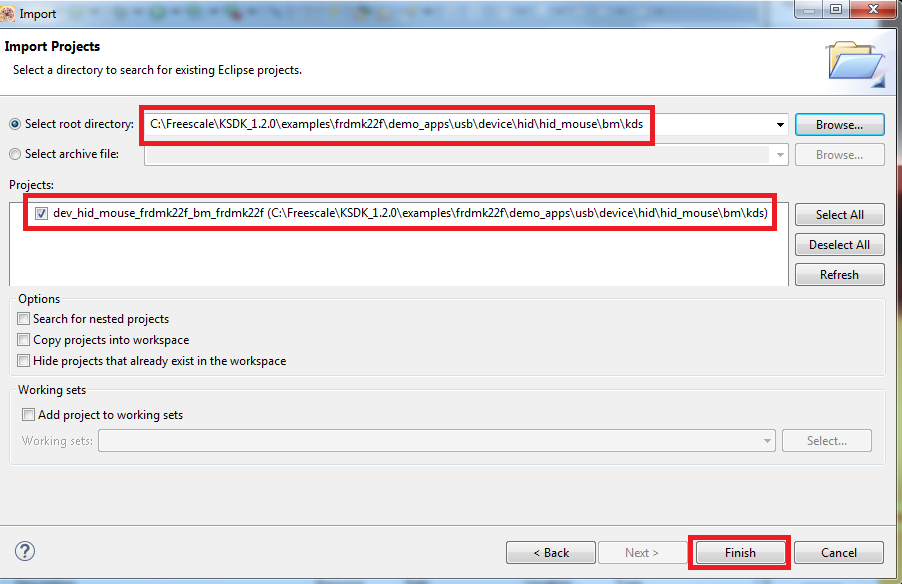
* 1. Follow the same steps to import the USB baremetal device library. It can be found at the following location:

**C:\Freescale\KSDK\_1.2.0\usb\usb\_core\ device\build\kds\usbd\_sdk\_frdmk22f\_bm**

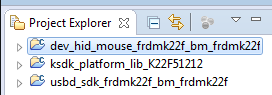
Note: Make sure you import the **frdm**k22f projects, and not the similarly named **twr**k22f120m projects.



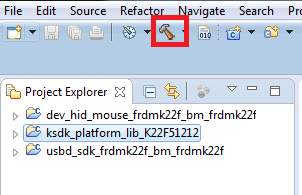
* 1. Finally import the device HID mouse example. Use the same steps as above, and the project can be found at: **C:\Freescale\KSDK\_1.2.0\examples\frdmk22f\demo\_apps\usb\device\hid\hid\_mouse\bm\kds**



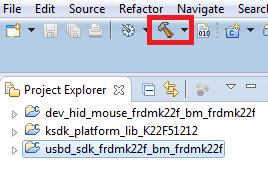
* 1. The following projects should now be imported in the workspace



* 1. Now we will build all of the projects. You must build the library project first before you can compile the USB example application.
  2. Click on “ksdk\_platform\_lib\_K22F51212” and hit the hammer icon to compile the KSDK platform library.

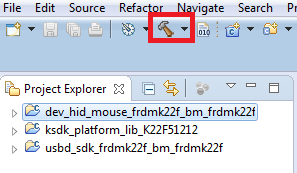


* 1. Then do the same on “usbd\_sdk\_frdmk22f\_bm\_frdmk22f” and hit the hammer icon to compile the USB device library for the K22F

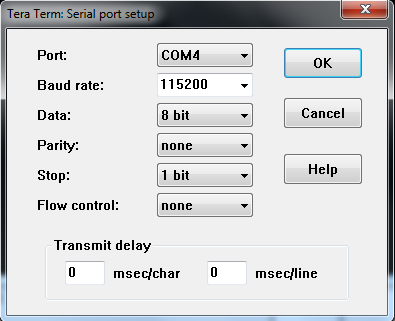


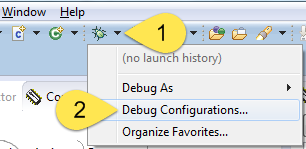
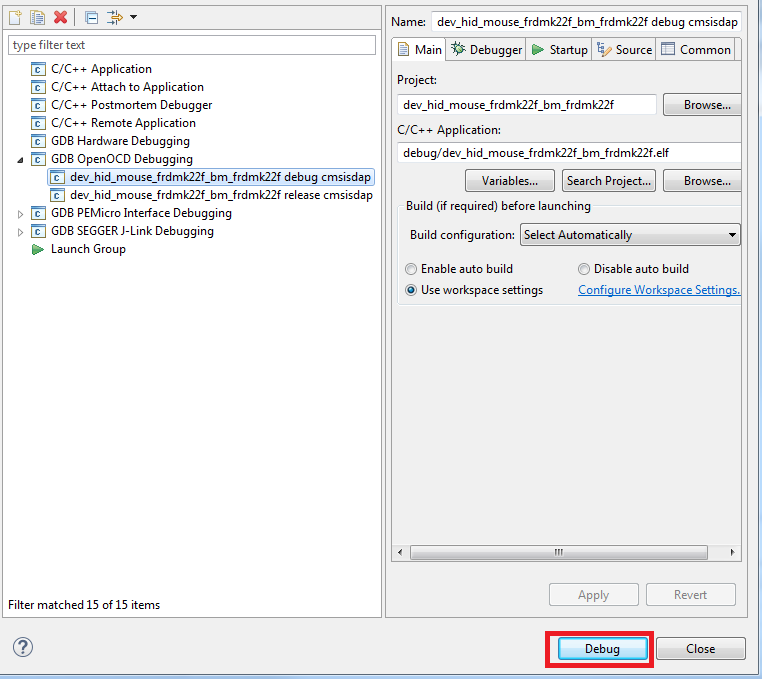
Note: If you click on the arrow next to the hammer icon, you will see a list of the build targets available for that project. By default, KSDK projects use the “Debug” target which means no optimization. There is also a “Release” target which is at full optimization. However for this lab we will use the default “Debug” target.

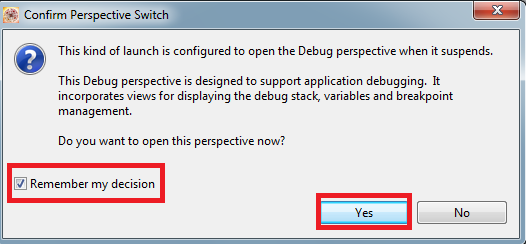
* 1. Now you can compile the HID Mouse example by clicking on the “dev\_hid\_mouse\_frdmk22f\_bm\_frdmk22f” project and then clicking on the hammer icon



* 1. Now plug the USB cable to the FRDM-K22F board if it is not already connected. Connect to the micro-USB connector labeled “SDA USB”. The board should have the CMSIS-DAP OpenSDA app loaded by default. Windows may need some time to load the driver.
  2. Find the COM port for the OpenSDA serial connection. Windows Device Manager will show the COM number under the Ports (COM & LPT) group. Here, the COM port is 4 but it will likely be different on your computer.   
     
  3. Connect a terminal program to this COM port using the TeraTerm program on the desktop. Use 115200 baud.

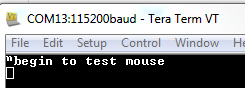


* 1. Back in KDS, open the Debug Configurations by clicking the **pull-down arrow** next to the green bug on the upper toolbar. Then select **Debug Configurations**.  
     
  2. In the left panel, select the configuration **dev\_hid\_mouse\_frdmk22f\_bm\_frdmk22f debug cmsisdap**, found under the under the “GDB OpenOCD Debugging” category, and then hit the **Debug** button.  
     
  3. Then you may also get the following message about switching to the Eclipse Debug view. Click on “Remember my decision” and then click on “Yes”. This will take you to the Debug view.

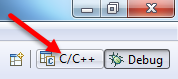


* 1. After the programming has completed and the debugger halts in main(), click the **Resume** button or press the **F8** key.



* 1. In the terminal, you will see that the USB stack is waiting to be connected to a host computer  
     

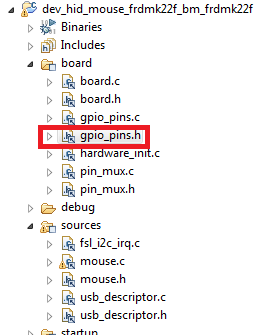
Note: If you do not see any output on the terminal, try closing Tera Term, then unplug the board. Re-plug in the board, re-open Tera Term, and then go through the debug process again. This is necessary because if the board is plugged in while Tera Term is already open, it will not properly connect to the virtual serial port.

* 1. Plug in the other USB cable into the micro-USB socket labeled “USB K22”. This is connection used by the K22 that will enumerate as a mouse.
  2. In Windows you should see the enumeration start to begin. After a short moment the mouse cursor will begin to move in a square.
  3. When done, unplug the HID Mouse USB cable
  4. Terminate the debugger by clicking on the red square  
     
  5. Switch back to the **C/C++ Perspective**.  
     

# Lab 2: Adding GPIO Functionality

In this lab, we’ll add code to use the GPIO driver to turn on and off an LED. The completed version of this lab can be found in the “Advanced KSDK” folder under **mouse-lab2.c**

* 1. First, open up the KSDK API Reference Manual, found at **C:\Freescale\KSDK\_1.2.0\doc\ Kinetis SDK v.1.2.0 API Reference Manual.pdf**
  2. Back inside of KDS, open up **gpio\_pins.h**. This is a shared file between all of the FRDM-K22F projects.

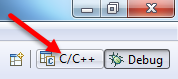


* 1. Inside of **gpio\_pins.h**, look at line 50 to see where the GPIO pins are being defined. Make note of kGpioLED1 as that is the LED we will be using for this lab. No changes should be done to this file.
  2. Next open up **gpio\_pins.c** and look at the structure for the LED pins starting at line 60 named **ledPins[]**. This is the structure that will be used to initialize the GPIO. Note how it makes use of kGpioLED1. No changes should be done to this file.
  3. Then open up mouse.c and scroll down to the main() function.
  4. At the start of main(), add the following line of code to initialize LED0 (green LED):

**GPIO\_DRV\_OutputPinInit(&ledPins[0]);**

* 1. Then add the following line right below it to run the LED on. The LED is active low, which is why we are writing a zero to the driver

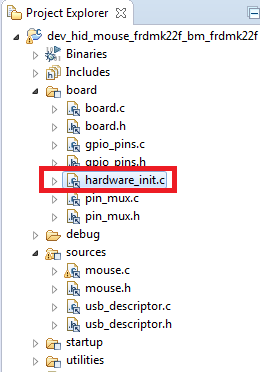
**GPIO\_DRV\_WritePinOutput(kGpioLED1, 0);**

* 1. Look in the API Reference Manual opened at the beginning of this lab and search for the **GPIO\_DRV\_WritePinOutput** API call to see the details about the arguments being passed into it.
  2. Now we want to see the LED blink on and off while the mouse moves.
  3. Find the mouse\_move() function. This function is called by the USB\_App\_Class\_Callback, which is ran when the host requests data. This callback function is where all the user code should be placed or called from, as the USB stack takes care of the rest.
  4. You will see there are 4 case statements that direct the mouse in different directions inside this move\_mouse() function. Make sure to look at the section of code currently enabled for Lab 1 and Lab 2.
  5. Inside both of the RIGHT and DOWN case statements, add a line of code to turn on the LED. The exact line and syntax is left as an exercise for the reader. However
  6. Inside both the LEFT and UP case statements, add a line to turn off the LED. The exact line and syntax is left as an exercise for the reader.
  7. Save the file, recompile, and debug as done in the previous lab above
  8. Now as the mouse cursor moves around the screen, the green LED will be turning on and off.
  9. When done, unplug the HID Mouse USB cable
  10. Terminate the debugger by clicking on the red square  
      
  11. Switch back to the **C/C++ Perspective**.  
      
  12. Now using the GPIO\_DRV chapter of the KSDK API Reference Manual, try using the other GPIO\_DRV APIs available to turn on, off, and toggle the LED without using the GPIO\_DRV\_WritePinOutput API function.
  13. You can also try using the different colors of LEDs. Remember the other pin names declared in gpio\_pins.h and gpio\_pins.c.

# Lab 3: Adding I2C Functionality

In this lab, we’ll add code to use the I2C driver to read accelerometer data from the on-board Freescale FXOS8700 accelerometer.

* 1. Because this lab will make use of a new hardware module, hardware\_init.c needs to be modified to do the pin muxing for the I2C module
  2. Open up the FRDM-K22F schematic, which is found inside the KSDK Advanced folder on the desktop, and determine which I2C module and pins are connected to the FXOS8700 accelerometer
  3. You should see that it is connected to I2C0 using pins PTB2 and PTB3
  4. Open up hardware\_init.c and find the hardware\_init() function. Remember that this file is specific to each application.



* 1. Inside that function, add the following line of code to initialize the I2C0 module:

**pin\_mux\_I2C(0);**

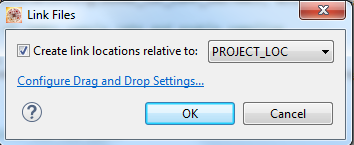
* 1. Open up pin\_mux.c
  2. Verify that the pin\_mux\_I2C() function (which is what configure\_i2c\_pins is mapped to) configures the correct pins for I2C0. No changes should be made since it is already setup correctly.
  3. Now open up mouse.c. This file has been modified from the default file that comes with KSDK to simplify this lab.
  4. Change the #define at the very top to enable the code for Lab 3 by setting it to 1:

**#define LAB3 1**

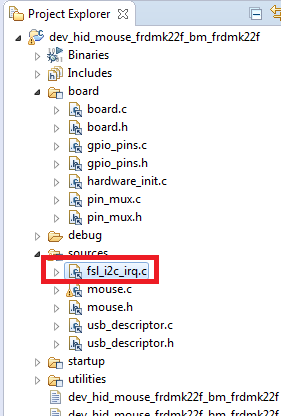
* 1. Inside of main() further down (starting at line 381)
  2. Look at the code that initializes the I2C module and writes values to the accelerometer.
  3. Then inside the while(1) loop in main, we want to read accelerometer data.
  4. Based on the I2C code you looked at earlier, and by looking at the KSDK API document, try to figure out what the code should be to read the 6 bytes of accelerometer data after the OUT\_X\_MSG\_REG register

Hint: Search in the PDF for I2C\_DRV\_MasterReceiveDataBlocking

* 1. The answer is to add the following line where the code is commented on line 434 under the comment about reading data: **I2C\_DRV\_MasterReceiveDataBlocking(BOARD\_I2C\_COMM\_INSTANCE, &slave, &reg, 1,receiveBuff, 6, 200);**
  2. Next, look at the move\_mouse() function to the code that is newly enabled with **#if LAB3**
  3. Add GPIO\_DRV calls to turn on the LED depending if the board is tilted, and off if it is not. Use the same GPIO API as before. Remember that the LEDs are active low.
  4. Finally, because we are using the I2C driver, we need to add the interrupt handler for it. This is conveniently already provided in a **fsl\_I2C\_irq.c** file provided by KSDK.
  5. Open up Windows Explorer and find this file at **C:\Freescale\KSDK\_1.2.0\platform\drivers\src\i2c\fsl\_irq.c**
  6. Drag and drop it into your KDS Project Explorer file structure.
  7. You’ll see a dialog box come up asking to add it as Project Relative, so click on OK:



* 1. Once done, it should look like the following:



* 1. Compile, run, and debug as earlier
  2. Now when the board is tilted, the mouse will move right or left, and the LED will turn on.
  3. If you have extra time, try adding movement based on if board is tilted up or down (found in receiveBuff[2]). Also initialize the other two LEDs to change colors based on the direction the board is tilted.