Freescale Semiconductor

Create a new FRDM-KL25Z project with KSDK 1.2.0 in KDS 3.0.0

By: Technical Information Center

Introduction

This document explains how to create a new KSDK 1.2.0 project in KDS 3.0.0. It is based in the document "Writing my first KSDK 1.2 Application in KDS 3.0 - Hello World and Toggle LED with GPIO Interrupt". This document is focused in the FRDM-KL25Z but can be used as a reference for other boards or processors.

Contents

1. Install the Eclipse update

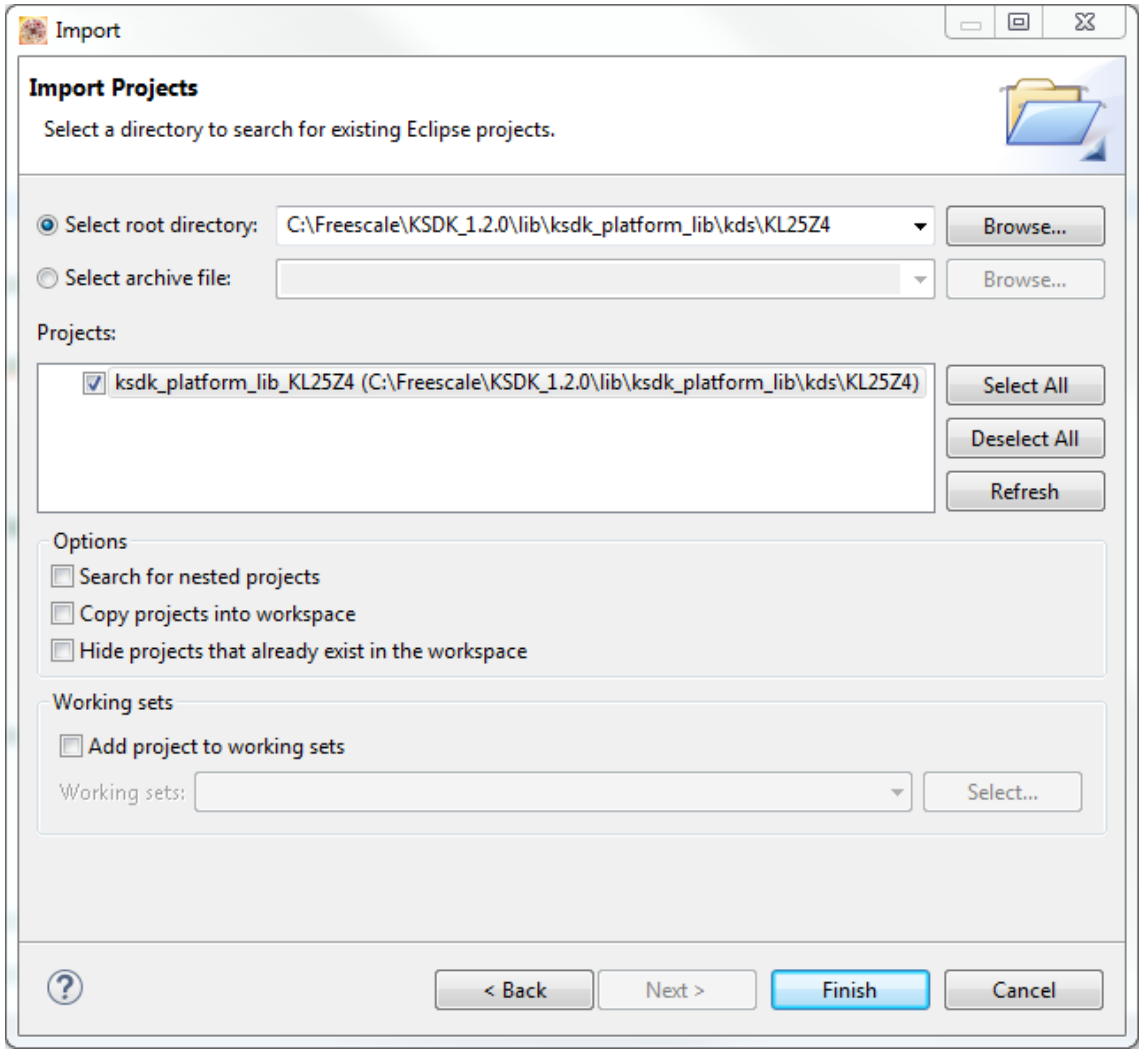
The first time that KSDK 1.2.0 is used with KDS 3.0.0 the eclipse update must be applied to generate KSDK compatible projects while creating a new project. The steps to install the Eclipse update are described in the section 5.2 Install Eclipse update in the document "Getting started with Kinetis SDK (KSDK) v.1.2.pdf", this document is located in the KSDK 1.2.0 installation folder (C:\Freescale\KSDK_1.2.0\doc).

2. Build the KSDK Platform Library (libksdk_platform.a)

Before to create the KSDK project the KSDK Platform Library must be built. This procedure must be done for each microcontroller to be used.

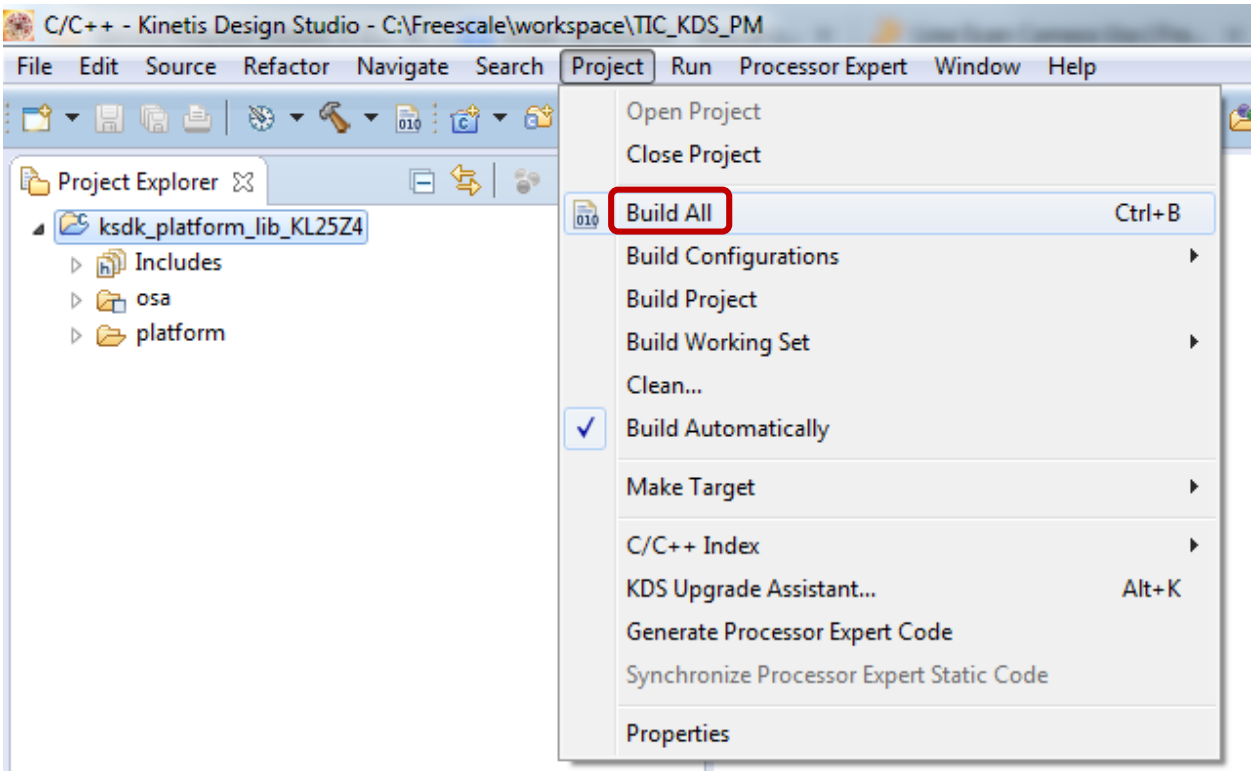
2.1 Import the library project

Import the project in the path C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds**Target_MCU**, in this case is chosen the project in the path **C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds\KL25Z4** because our target MCU is the KL25Z4.

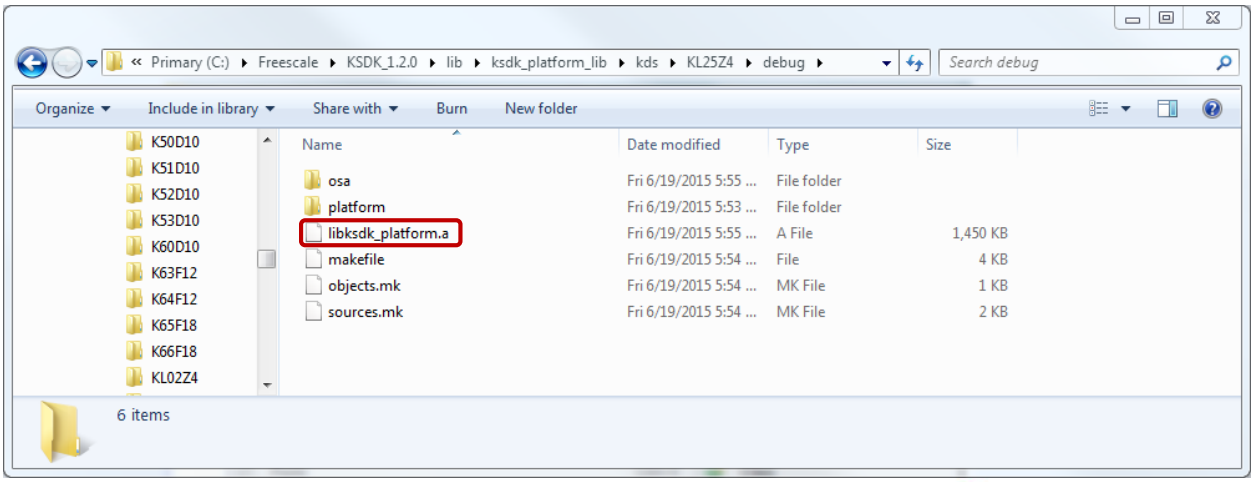


2.2 Compile KSDK Platform Library

Compile the project through the menu **Project > Build All** or by pressing CTRL+B to build all the open projects in the workspace.



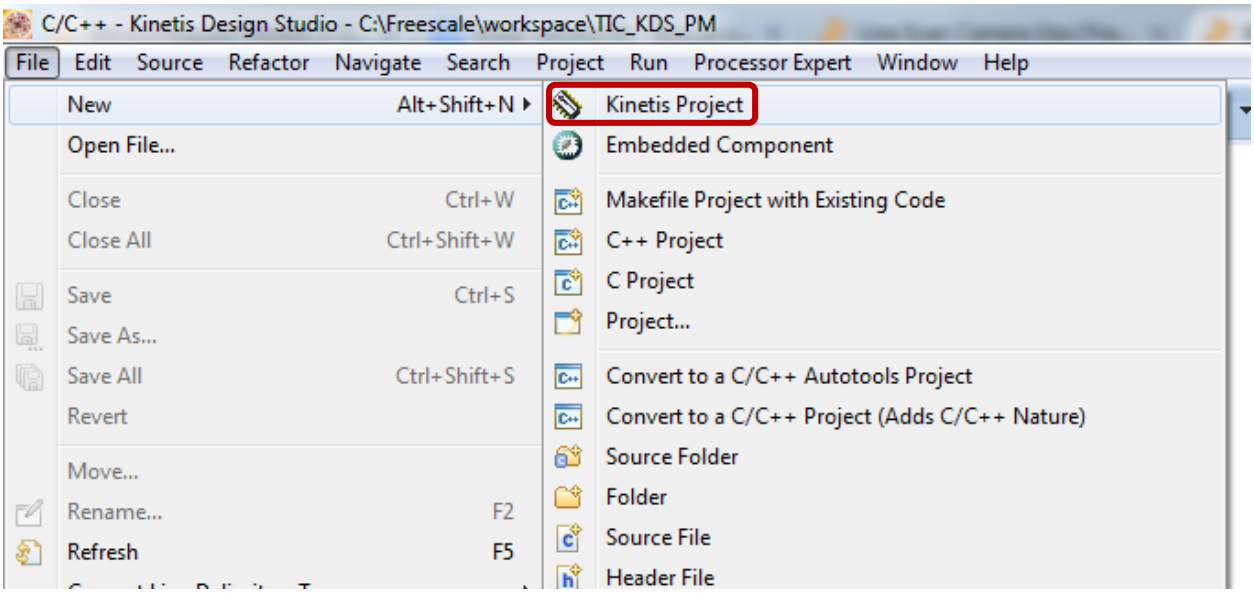
Now, the platform library (**libksdk_platform.a**) is built in the path C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds**Target_MCU**\debug, in this case **C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds\KL25Z4\debug**.



3. Create the KSDK 1.2.0 project

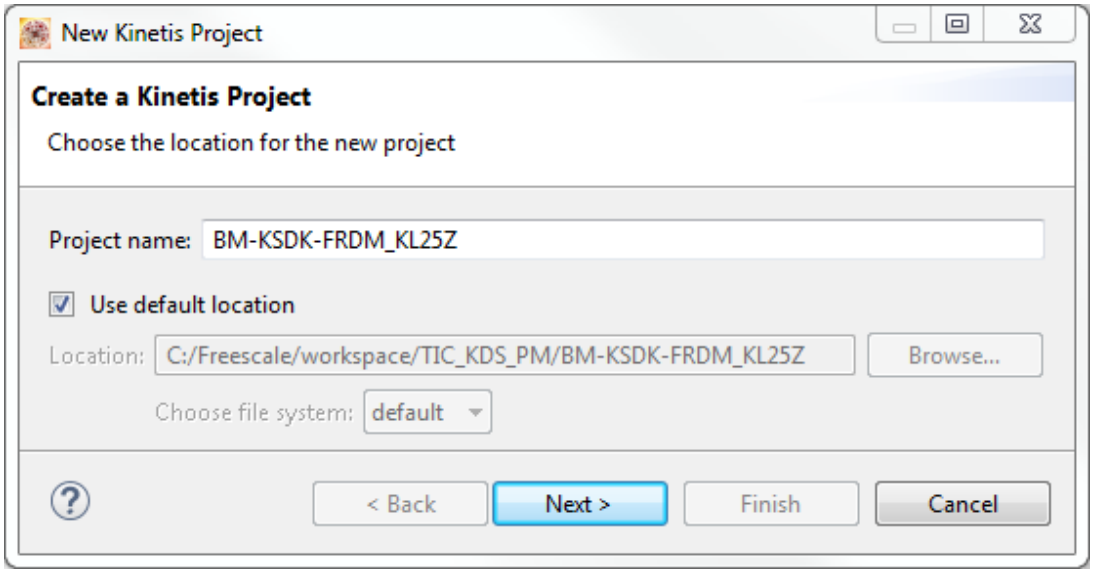
3.1 Create a new Kinetis Project

Create a new Kinetis project through the menu **File > New > Kinetis Project**.



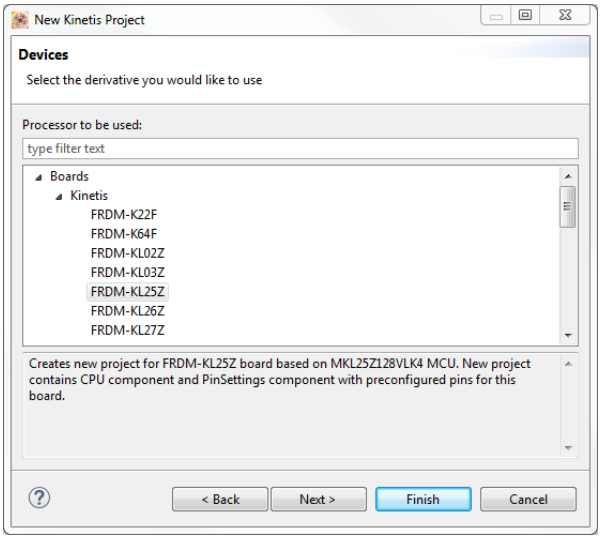
3.2 Choose the project's name and location

In this window is chosen the project's name and its location. For this example the default location (workspace) is chosen. Once the project's name and its location are defined click in the **Next >** button.



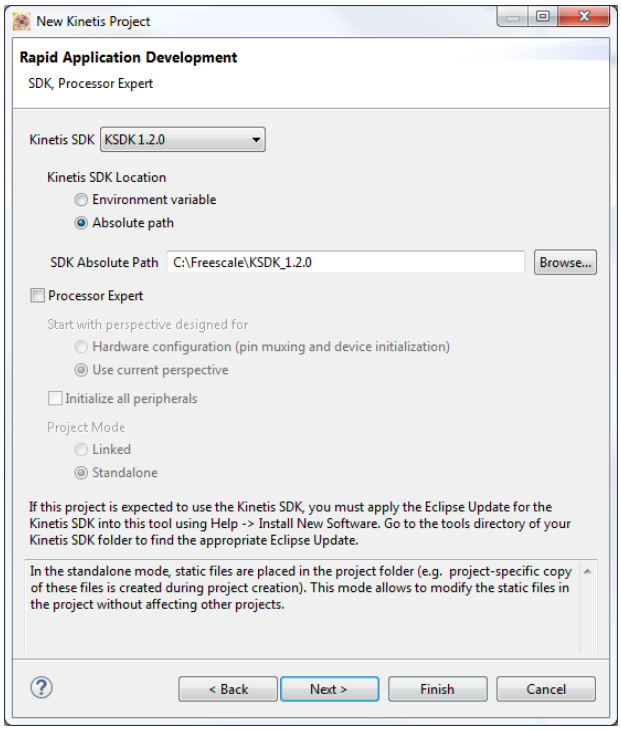
3.3 Select the target device

Select the target board or processor for the project. In this case has been chosen the FRDM-KL25Z which is based in the MKL25Z128VLK4 MCU. Once the device is selected click over the **Next >** button.



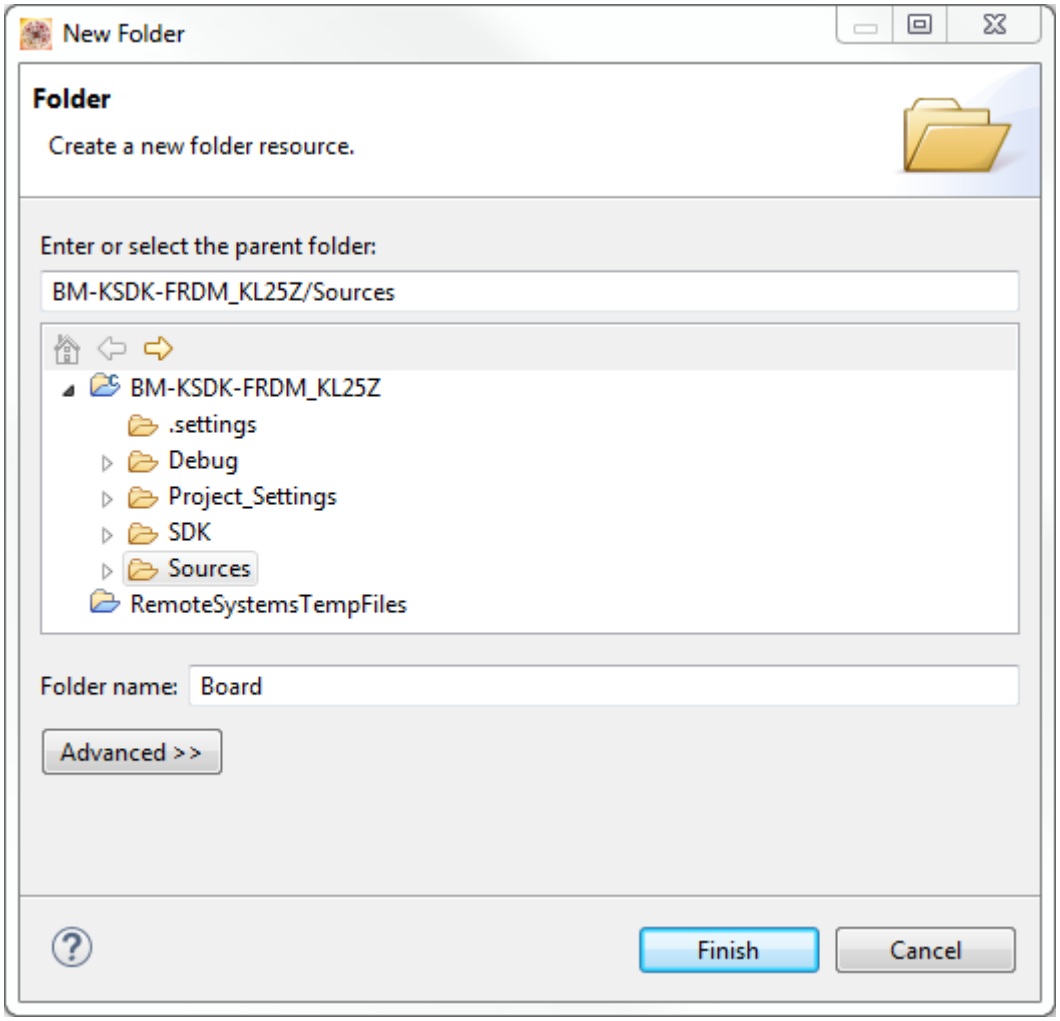
3.4 Enable KSDK in the project

In this window select KSDK 1.2.0 to be enabled and then click over the **Finish** button.



3.5 Create the Board and Utilities folders

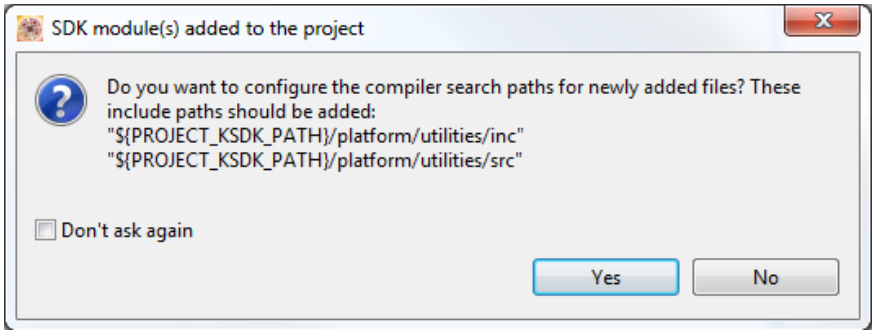
Once the project is created it is needed to add some configurations to let KSDK works. Click over the **Sources** folder and then select the option **New > Folder**. A new window is displayed where the folder's name and location must be chosen. The **Sources** folders must be selected and the folder's name must be **Board**, then click over the **Finish** button. This step has to be repeated to create the folder **Utilities** too.



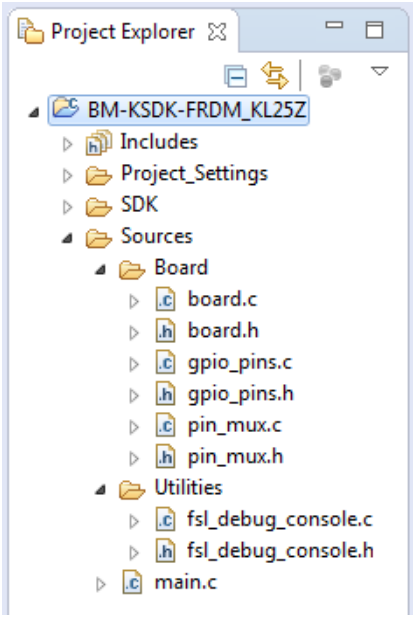
3.6 Copy the files to the Board and Utilities folders

Now is time to copy some files to the two new folders. The files **board.c**, **board.h**, **gpio_pins.c**, **gpio_pins.h**, **pin_mux.c** and **pin_mux.h** in the path C:\Freescale\KSDK_1.2.0\examples**Target_Board** (in this case **C:\Freescale\KSDK_1.2.0\examples\frdmkl25z**) must be copied inside the **Board** folder. The

file **fsl_debug_console.c** in the path C:\Freescale\KSDK_1.2.0\platform\utilities\src and the file **fsl_debug_console.h** in the path C:\Freescale\KSDK_1.2.0\platform\utilities\inc must be copied inside the **Utilities** folder. When the prompt below appears click over the **Yes** button to add the utilities paths in the project.



The resulting **Project explorer** tree must be the same as the image below.

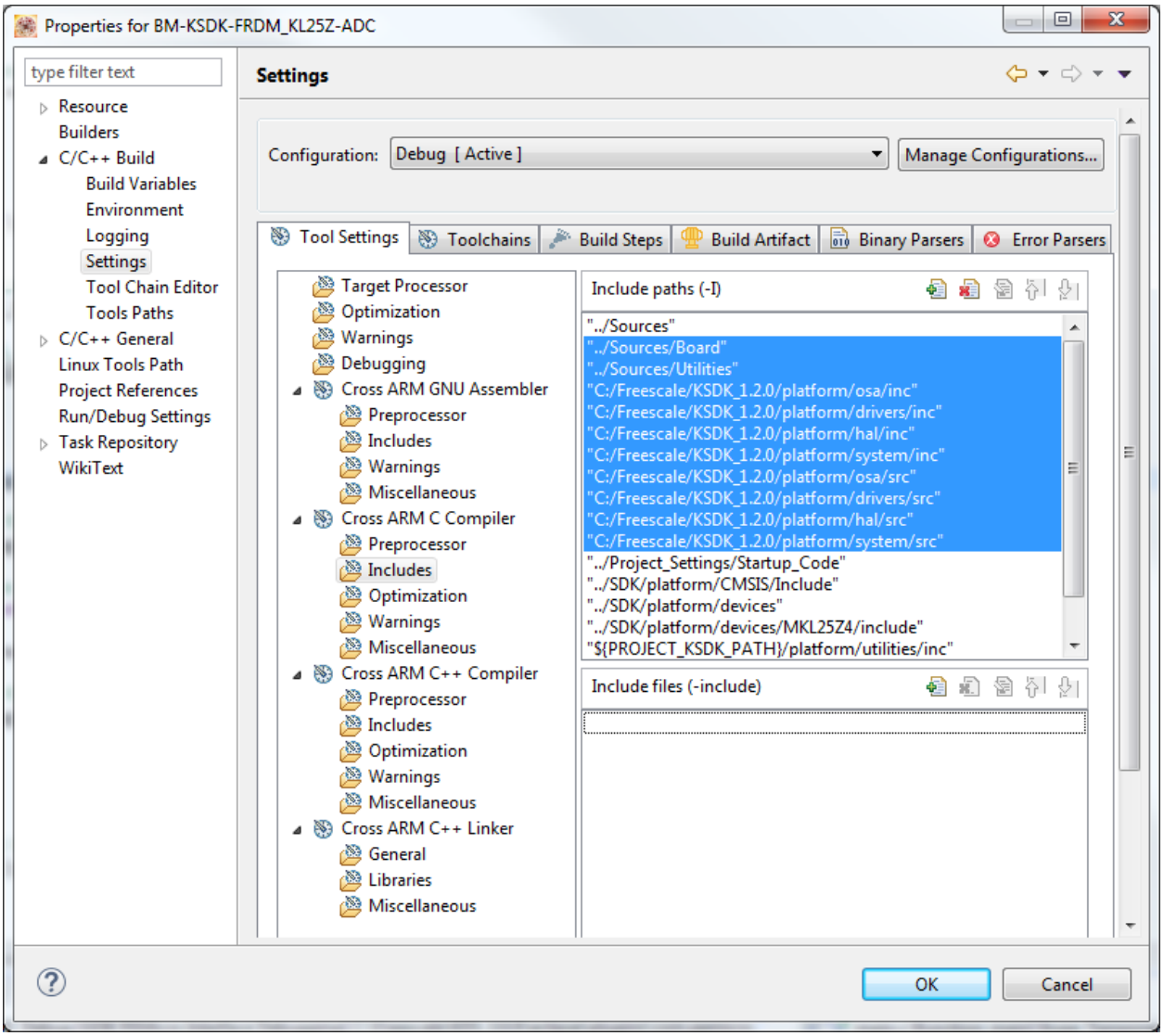


3.7 Add the KSDK paths

3.7.1 Cross ARM C Compiler Includes

It is time to add the KSDK paths in the project's configurations. To enter to the project properties right click over the project and then select the option **Properties** in the menu. Go to the option **C/C++ Build > Settings > Cross ARM C Compiler > Includes** and add the following paths in the section **Include paths (-I)**:

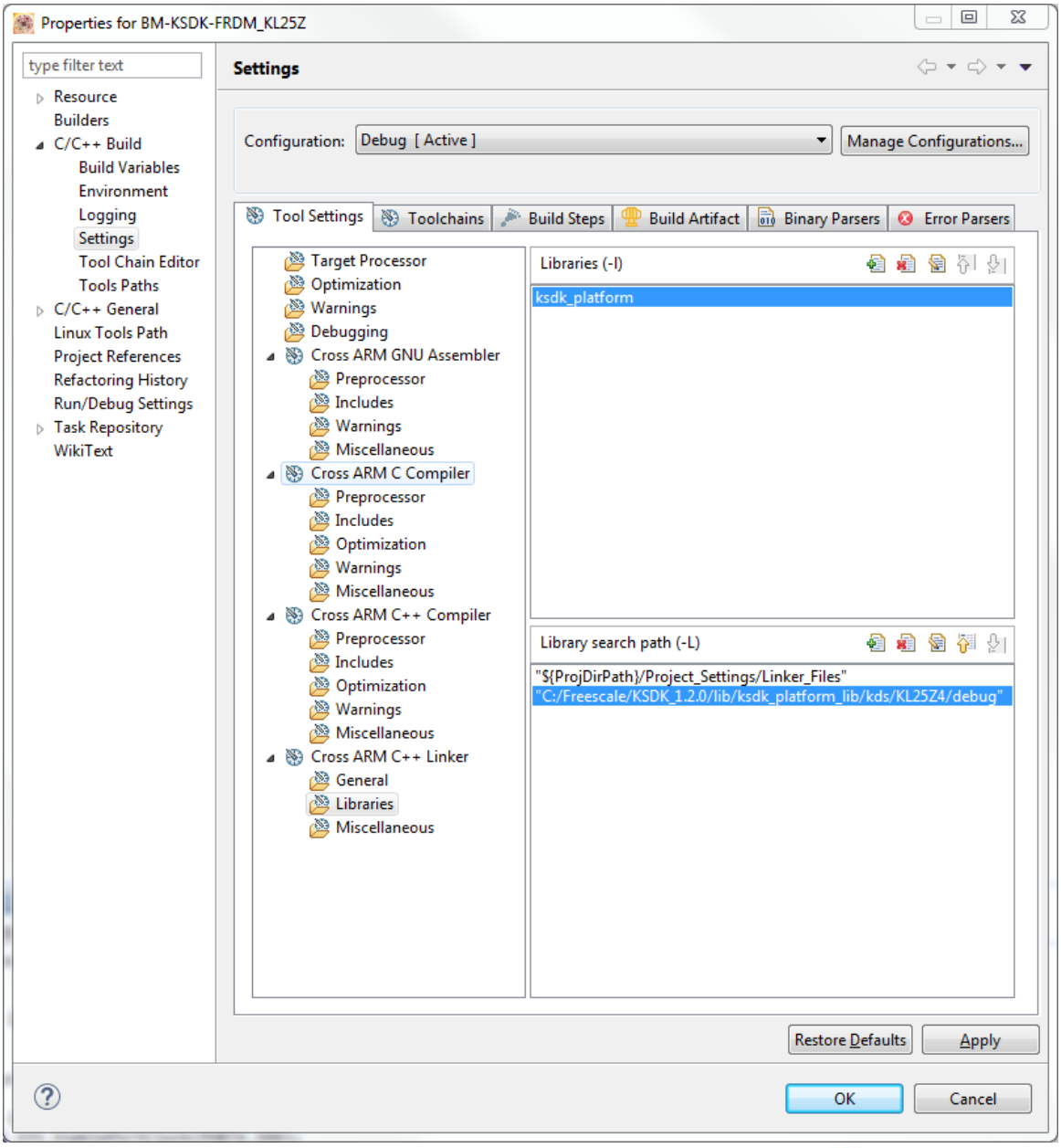
"../Sources/Board"
"../Sources/Utilities"
"C:/Freescale/KSDK_1.2.0/platform/osa/inc"
"C:/Freescale/KSDK_1.2.0/platform/osa/src"
"C:/Freescale/KSDK_1.2.0/platform/drivers/inc"
"C:/Freescale/KSDK_1.2.0/platform/drivers/src"
"C:/Freescale/KSDK_1.2.0/platform/hal/inc"
"C:/Freescale/KSDK_1.2.0/platform/hal/src"
"C:/Freescale/KSDK_1.2.0/platform/system/inc"
The resulting paths must be the same as the ones in the image below.



3.7.2 Cross ARM C++ Linker Libraries

Go to the option **C/C++ Build > Settings > Cross ARM C++ Linker > Libraries** and add **ksdk_platform** in the section **Libraries (-l)**. Then add the path C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds**Target_MCU**\debug, in this case **C:\Freescale\KSDK_1.2.0\lib\ksdk_platform_lib\kds\KL25Z4\debug**, in the section **Library search path (-L)**.

The resulting libraries must be the same as the ones in the image below.



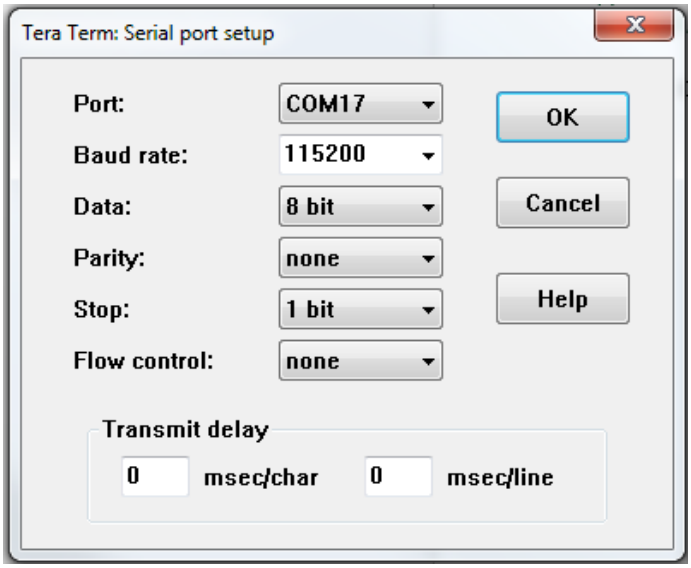
4. A simple Hello world! Application

Copy the code bellow and replace the one in the **main.c** file.

```
#include "fsl_device_registers.h"
#include "fsl_clock_manager.h"
#include "board.h"
#include <stdio.h>

int main(void)
{
	/* Enable clock gate for all the PORTs. */
	CLOCK_SYS_EnablePortClock(PORTA_IDX);
	CLOCK_SYS_EnablePortClock(PORTB_IDX);
	CLOCK_SYS_EnablePortClock(PORTC_IDX);
	CLOCK_SYS_EnablePortClock(PORTE_IDX);

	/* Configuration for the board clock to run with a core clock = 48MHz. */
	BOARD_ClockInit();

	/* Initialize the debug UART to allow the printf function works. */
	dbg_uart_init();

	/* Test the printf function. */
	printf("Hello world!\r\n");

	/* Initialize the switches and LEDs. The configuration is in the file
gpio_pins.c. */
	GPIO_DRV_Init(switchPins, ledPins);

	/* Test the GPIO pins. */
	LED1_ON;
	LED2_ON;
	LED3_ON;

	/* This for loop should be replaced. By default this loop allows a single
stepping. */
	for (;;)
	{}

	/* Never leave main */
	return 0;
}
////////////////////////////////////////////////////////////////////////////////
// EOF
////////////////////////////////////////////////////////////////////////////////
```

5. Run the example application

Compile and debug the project. Then configure a terminal like Tera Term with a baud rate of 115200 without flow control and run the application.



The terminal should show the **Hello world!** Message.