Document Number: XSFL_Prod_UG
Rev 1.2, 1/2015

Freescale Sensor Fusion for Kinetis Product Development Kit User Guide

Contents

1 Introduction

Sensor fusion is a process by which data from several different sensors are “fused” to compute something more than could be determined by any one sensor alone. An example is computing the orientation of a device in three dimensional space. That data is then used to alter the perspective presented by a 3D GUI or game.

The Freescale Sensor Fusion Library for Kinetis MCUs (also referred to as *Fusion Library* or *development kit*) provides advanced functions for computation of device orientation, linear acceleration, gyro offset and magnetic interference based on the outputs of Freescale inertial and magnetic sensors.

The development kit now includes:

- Full source code for the sensor fusion libraries
- Prepackaged implementations targeted at specific Freedom development platforms.
- Support for CodeWarrior and Kinetis Design Studio
- Full documentation including user manual and fusion data sheet

The fusion library is now supplied under a liberal BSD open source license, which allows the user to employ this software with Freescale MCUs and sensors, *or those of our competitors*. Support for issues relating to the default distribution running on Freescale reference hardware is available via standard Freescale support channels. Support for

non-standard platforms and applications is available at https://community.freescale.com/community/sensors/sensorfusion or can be contracted via Freescale's Software Services team.

This document is part of the documentation for Freescale Sensor Fusion Library for Kinetis MCUs (compatible with Freedom Development Platform, FRDM-KL25Z, FRDM-KL26Z, FRDM-K20D50M, and FRDM-K64F) software. Its use and distribution are controlled by the license agreement in the Software Licensing.

1.1 Software Licensing

1.2 Software Features

The development kit comes with an installer which installs, to the development PC, and CodeWarrior projects targeted to each of the supported Freescale Freedom Development Platforms (used in conjunction with the FRDM-FXS-MULTI family of sensor boards) listed below. In addition to CodeWarrior, Freescale will be supporting Kinetis Design Studio for this release.

- FRDM-KL25Z
- FRDM-KL26Z
- FRDM-K20D50M
- FRDM-KL46Z and
- FRDM-K64F Freedom Development Platforms

This product provides access to source code for all functions. The software features include:

- Fusion and magnetic calibration algorithms are pre-compiled as a linkable library
- Programmable sensor sample rate (Default= 200 Hz)
- Programmable sensor fusion rate (Default = 200 Hz)
- Supported frames of reference include NED, Android and Windows 8.
- No procedural interface to learn.
    - Write sensor readings into one set of global structures
    - Read fusion results from a different set of global structures
- Ability to compile and link any combination of four standard algorithms

  - Accelerometer only (tilt)
  - Accelerometer plus magnetometer
  - Accelerometer plus gyroscope
  - Accelerometer plus magnetometer and gyroscope
  - 2D eCompass plus gyroscope only
- Freescale's award-winning magnetic compensation software, which provides geomagnetic field strength, hard and soft iron corrections and quality-of-fit indication
- Supported by Freescale Code Warrior, Kinetis Design Studio and Processor Expert tools
- Directly compatible with the Freescale Sensor Fusion Toolbox for Android and Windows (***Fusion Toolbox***). Board-specific template programs include predefined Bluetooth/UART interfaces compatible with the Fusion Toolbox.

1.3 Supporting Documentation

- Freescale Sensor Fusion Library for Kinetis Data Sheet
- www.freescale.com/sensorfusion
- MCU on Eclipse (blog)
- Orientation Representations: Part 1
- Orientation Representations: Part 2
- Hard and soft iron magnetic compensation explained
- CodeWarrior Integrated Development Environment Software at www.freescale.com/codewarrior
- Kinetis Design Studio software at www.freescale.com/kds
- Euler Angles at en.wikipedia.org/wiki/Euler_Angles
- Introduction to Random Signals and Applied Kalman Filtering, 3rd edition, by Robert Grover Brown and Patrick Y.C. Hwang, John Wiley & Sons, 1997
- Quaternions and Rotation Sequences, Jack B. Kuipers, Princeton University Press,1999
- Freescale Freedom development platform home page at Freescale Freedom Development Platform
- OpenSDA User's Guide, Freescale Semiconductor, Rev 0.93, 2012-09-18
- PE micro Open SDA Support page at www.pemicro.com/opensda

1.4 Requirements

1.4.1 Kinetis Platform

The Sensor Fusion Library currently runs on the following two hardware platforms:

- KL25Z Freedom Development Platform Figure 1 based on the Kinetis KL2 family MKL25Z128VLK4 MCU
  - 48 MHz ARM M0+ core
  - 128 KB of flash memory
  - 16 KB of RAM.

Figure 1. KL25Z Freedom Development Platform (FRDM-KL25Z)

- KL26Z Freedom Development Platform, Figure 2, based on the Kinetis KL2 family MKL26Z128VLH4 MCU
  - 48 MHz ARM M0+ core
  - a full-speed USB controller
  - 128 KB of flash memory
  - 16 KB of RAM.

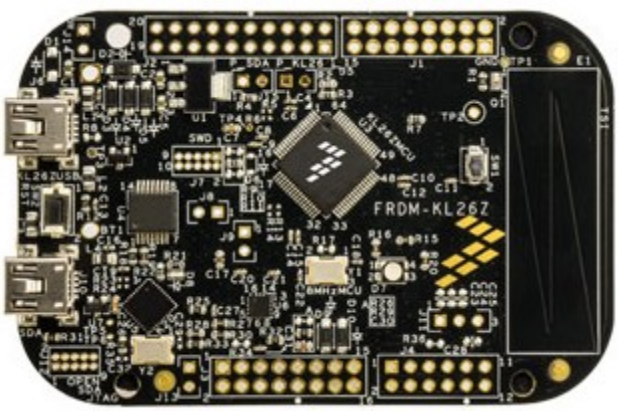


Figure 2. KL26Z Freedom Development Platform (FRDM-KL26Z)

- KL46Z Freedom Development Platform, Figure 3, based on the Kinetis KL2 family MKL26Z128VLH4 MCU
  - 48 MHz ARM M0+ core
  - a full-speed USB controller
  - 128 KB of flash memory
  - 16 KB of RAM.

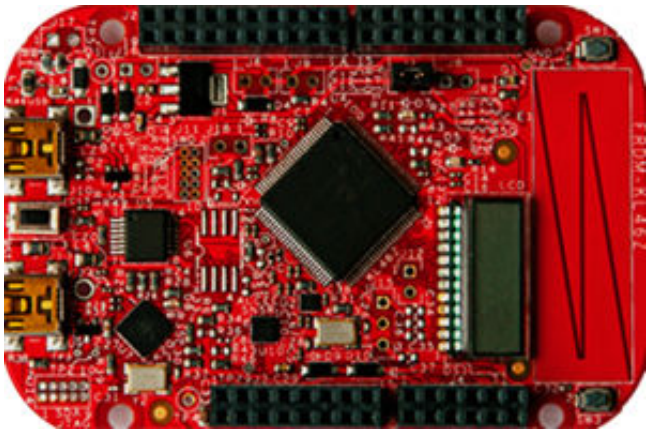


Figure 3. KL46Z Freedom Development Platform (FRDM-KL46Z)

- K20D50M Freedom Development Platform, Figure 4, based on the Kinetis K20 family MK20DX128VLH5 MCU
  - 50 MHz ARM M4 core
  - 128 KB of flash memory
  - 32 KB of FlexNVM
  - 16 KB of RAM.

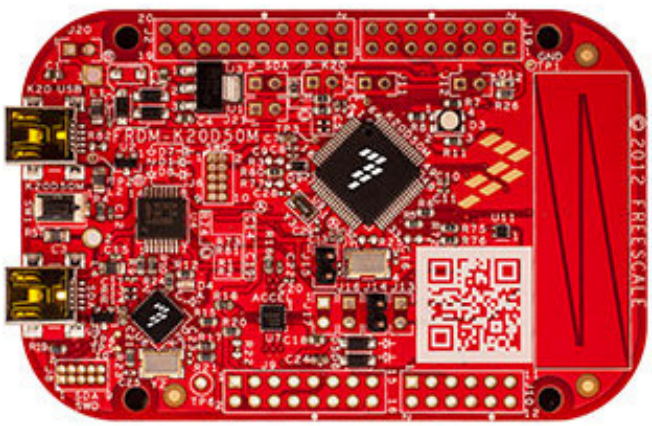


Figure 4. K20D50M Freedom Development Platform (FRDM-K20D50M)

- K64F Freedom Development Platform, Figure 5, based on the Kinetis K60 family MK64FN1M0VLL12 MCU
  - 120 MHz ARM M4 core with FPU
  - 1 MB of flash memory
  - 256 KB of RAM.

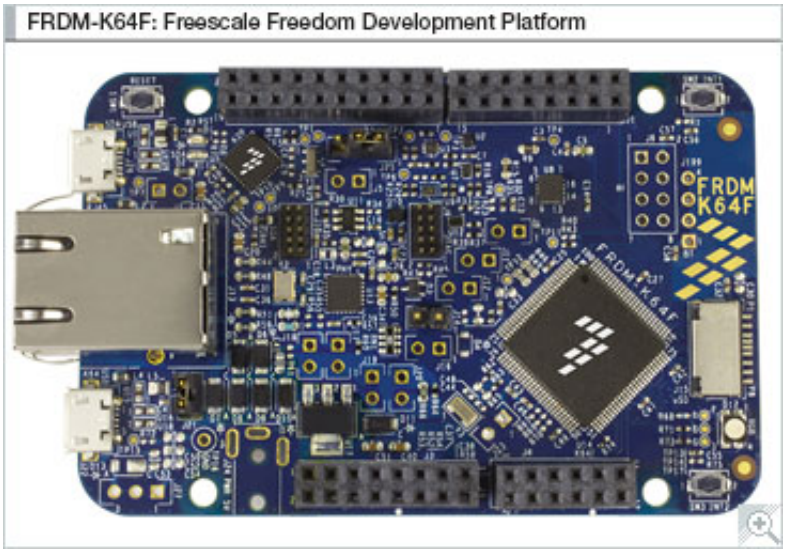


Figure 5. K64F Freedom Development Platform (FRDM-K64F)

Any of these platforms can be programmed via the OpenSDA serial port on the board. A USB mini connector cable is required to program the board.

1.4.2 Freedom Sensor Development Platform

Freescale has released several Freedom Sensor Development platforms compatible with the Freedom Development Kinetis Platforms discussed in Kinetis Platform. These are:

- FRDM-FXS-9-AXIS, which contains an FXOS8700CQ 6-axis accelerometer/magnetometer combination sensor and an FXAS21000 gyroscope
- FRDM-FXS-MULTI, which contains all features of the FRDM-FXS-9-AXIS plus additional sensors not yet supported
- FRDM-FXS-MULTI-B, which contains all features of the FRDM-FXS-MULTI plus Bluetooth Module and battery

All three are built from the same PCB design and differ only in the number of parts on the platform. The Fusion Library works with any of the three boards, but the FRDM-FXS-MULTI-B is the only one that supports Bluetooth communications to the Freescale Sensor Fusion Toolbox for Android.

Key components of the FRDM-FXS-MULTI-B are identified in Figure 6.

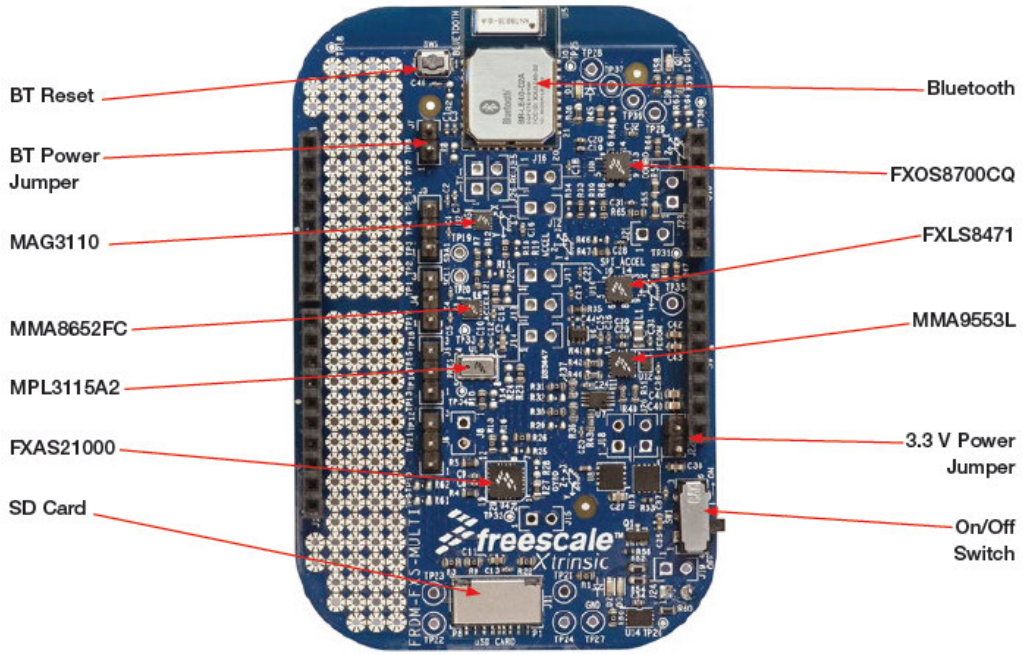


Figure 6. Freedom Development Platform Multi-B with key components

NOTE

The user must remove the BT Power Jumper when using OpenSDA Communications Device Class [CDC] (UART/USB) to communicate with a PC.

The FRDM-FXS-MULTI-B Sensor Development Platform connects to the FRDM-KL25Z Freedom Development Platform, see Figure 7.

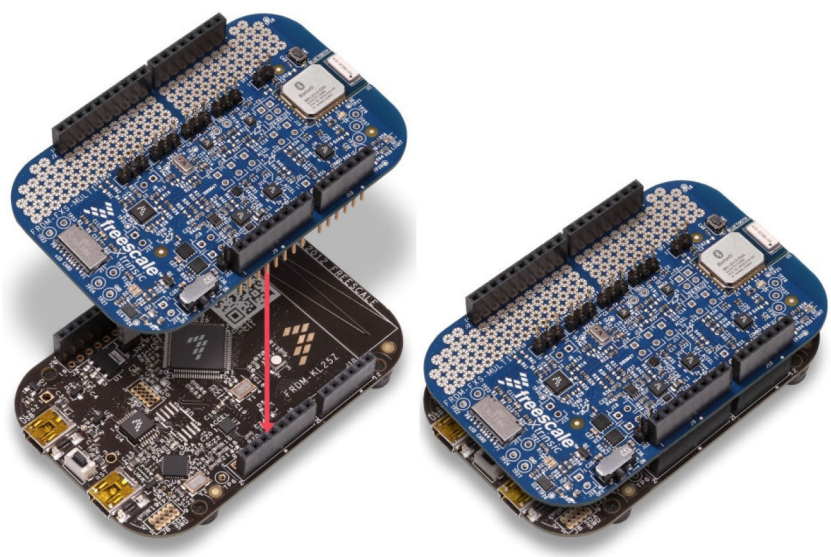


Figure 7. Freedom Development Platforms, FRDM-FXS-MULTI-B mated to FRDM-KL25Z

NOTE

When mating the two boards, check to ensure that sensor board, when configured with a battery mounted on the back, is free of any obstacles presented by the base, Freescale Freedom Development board.

1.4.3 Peripherals Used by the Kit

The product development software uses the virtual peripherals in Table 1. These are initialized via Processor Expert.

Table 1. Virtual peripherals mapping to MCU resources per board combination

Virtual Peripheral	Processor Expert Function	FRDM-KL25Z	FRDM-KL26Z	FRDM-KL46Z	FRDM-K20D50M	FRDM-K64F	Description
N/A	CPU	MKL25Z128VLK4 Cortex M0 + MCU	MKL26Z128VLH4 Cortex M0 + MCU	MKL46Z256VMC4 Cortex M0 + MCU	MK20DX128VLH5 Cortex M4	MK64FN1M0VLL12 Cortex M4 with FPU	Microcontroller
LED_RED	BitIO_LDD	PTB18	PTE29	PTE29	PTC3	PTB22	Red LED illuminated only during the period when a magnetic calibration is in progress.
LED_GREEN	BitIO_LDD	PTB19	PTE31	PTDS	PTD4	PTE26	Green LED flickers when the sensor fusion algorithms are executing.
LED_BLUE	BitIO_LDD	PTD1	PTD5	PTE31	PTA2	PTB21	Currently unused.
FTM	TimerUnit_LDD	LPTMR0	LPTMR0	LPTMR0	LPTMR0	LPTMR0	Timer to drive the sensor read process at a typical rate of 200Hz.
UART[1]	Serial_LDD	UART0 on PTA2:1	UART0 on PTA2:1	UART0 on PTA2:1	UART1 on PTE1:0	UART3 on PTC17:16	Serial port used to communicate with the Bluetooth module transmitting orientation and sensor data to the Android device.
I2C	I2C_LDD	I2C1 on PTC2:1	I2C1 on PTC2:1	I2C1 on PTC2:1	I2C0 on PTB1:0	I2C1 on PTC11:10	The I2C master bus communicating with the sensors.
Test_Pin_KF_Time	BitIO_LDD	PTC10	PTC10	Test_pin_KF_Time-PTC10	PTC10	PTC7	Output line used for debug purposes.
Test_Pin_MagCal_Time	BitIO_LDD	PTC11	PTC11	Pin_Test_MagCal_Time	PTC1	PTC5	Output line used for debug purposes.

1. See Table 1 for UART portability issues on the K20D50M and K64F platforms.

1.4.4 CodeWarrior

CodeWarrior 10.6 or above must be used to build these applications. CodeWarrior can be downloaded from Freescale at CW-MCU10 CodeWarrior for MCUs.

1.4.5 Graphical User Interface

A device running either Android 3.0 or Windows 7.0 or higher is required to run the Freescale Sensor Fusion Toolbox. Details are available at www.freescale.com/sensorfusion.

2 Quick Start Guides

2.1 Quick Start Guide for CodeWarrior

The development kit contains predefined CodeWarrior project(s) for use with supported Freescale Freedom Development Platforms. Kit functionality includes sensor drivers generated via Processor Expert, magnetic calibration, sensor fusion and data streaming interface. User code can be easily added to *user_tasks.c* within any of the following predefined functions:

- `void UserStartup(void);`
- `void UserHighFrequencyTaskInit(void); //runs once, the first time through the 200 Hz task`
- `void UserHighFrequencyTaskRun(void); //runs each time the 200 Hz task runs`
- `void UserMediumFrequencyTaskInit(void); //runs once, the first time through the 25 Hz task`
- `void UserMediumFrequencyTaskRun(void); // runs each time the 25 Hz task runs`

Sensor and fusion results are simply read from predefined structures.

The development kit provides access to drivers, hardware abstraction layers, communications interfaces, and more. Although this guide provides a convenient place to start, the user is not limited to only the configurability that is described. Details of complete configurability can be found in later sections of this guide.

2.1.1 Installing the Development Kit

This procedure explains how to obtain and install the Sensor Fusion Library Development Kit.

1. Download the firmware installer from www.freescale.com/sensorfusion. Under the section **Design Resources**, click on the link titled **Freescale Sensor Fusion Library for Kinetis MCUs**. Accept the licensing agreement and the installer should begin downloading automatically, or else click on the **Direct Link** to download it. The installer provides all necessary instructions and prerequisites.
2. If not already done, Freescale recommends updating the OpenSDA boot loader for the Freedom Development Platform to the most recent release for that board. There are two generations of OpenSDA boot loaders in existence. Each is targeted at specific boards. Version 1.0 (also known as **P & E Micro**) boot loaders may be found at www.pemicro/opensda. Freescale recommends these loaders for the KL25Z, KL26Z, KL46Z and K20D50M boards, see Figure 8 for example of the KL-25Z.

For K64F, Freescale recommends the Segger solution, www.segger.com/opensda.html.

The process to upgrade the K64F Version 2 OpenSDA (MBED) boot loader is described at mbed.org/handbook/Firmware-FRDM-K64F.

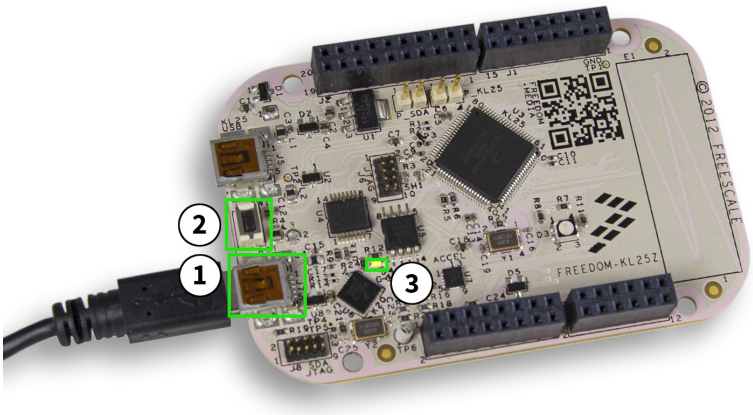


Figure 8. Freedom Development Platform KL-25Z

Procedure

The following procedure outlines the process for updating the OpenSDA Version 1.0 boot loader. The Version 2.0 process is similar, but uses .bin files. Consult the links above for the latest instructions. See Figure 8.

a. Hold down the Reset button (located in box #2 above) and then power the board by connecting a powered USB cord to the USB port (located in box #1 above). The LED will blink (located in box #3 above), indicating that the board is now in boot loader mode. Release the Reset button at this point.
b. Open Windows Explorer and locate the BOOTLOADER thumb drive.
c. If necessary, extract the DEBUG-APP-Vxxx.SDA file from the Pemicro_OpenSDA_Debug_MSD_Update_Apps_yyyy_mm_dd.zip file.
d. Drag the DEBUG-APP-Vxxx.SDA file onto the thumb drive (any application or firmware file can be drag-and-dropped onto the device at this point).
e. Unplug the USB cable from the board then plug it back in. The Freedom Development Platform is now updated to the latest firmware or application.

3. Start the installer and then follow the instructions in the installation wizard.

NOTE

The user can choose to install CodeWarrior, Kinetis Design Studio, or both sets of template projects.

2.1.2 Getting Started with the Firmware or Development with CodeWarrior

1. If not already accomplished, install and test CodeWarrior 10.6 on your computer. Download the tool by following the link provided in CodeWarrior.
2. After successfully running the installer, the file structure in the workspace should be as shown below:

```
approximations.c
approximations.h
build.h
drivers.c
drivers.h
Events.c
```

```
Events.h
files.txt
fusion.c
fusion.h
include_all.h
magnetic.c
magnetic.h
main.c
matrix.c
matrix.h
mqx_tasks.c
mqx_tasks.h
orientation.c
orientation.h
tasks.c
tasks.h
user_tasks.c
user_tasks.h
```

NOTE

The contents of the Sources directory is identical regardless of which board is targeted. Hardware specific customizations are done in **Processor Expert**. That code will go into the Generated_Code and MQX™ LITE directories.

3. Open **CodeWarrior** and in the Workspace Launcher dialog, see Figure 9, choose the workspace where the product development project is located.

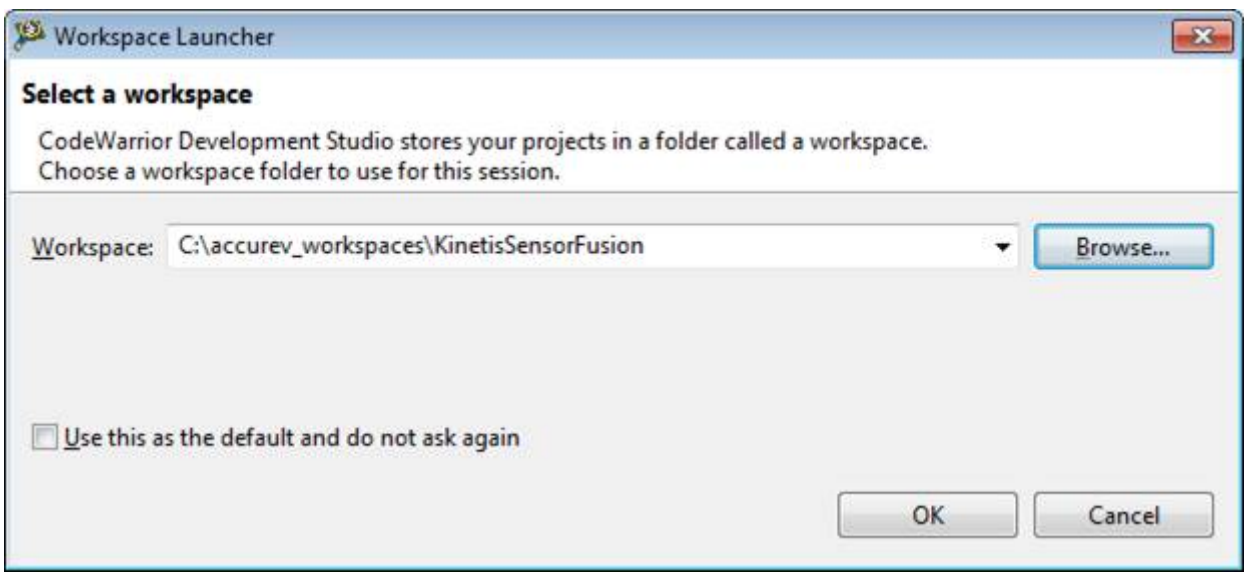


Figure 9. Workspace launcher select window

4. Import the project using File-> **Import** option, see Figure 10 select **General->Existing Projects into Workspace**

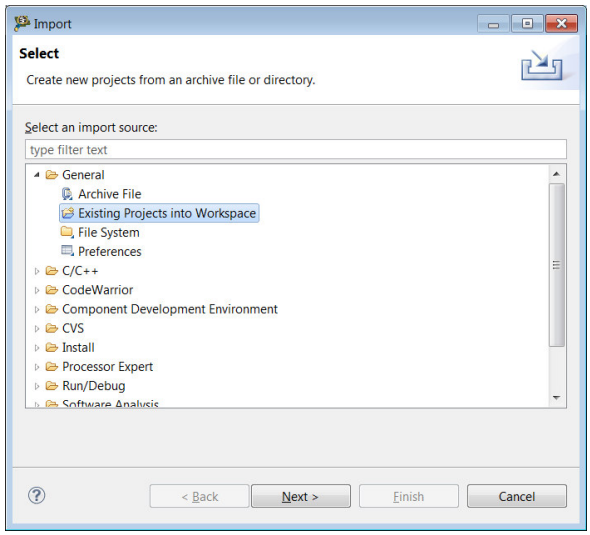


Figure 10. Import dialog window

5. Click **Next**. In the Import Window that appears, navigate to the folder containing the CodeWarrior project, and select it. See Figure 11.

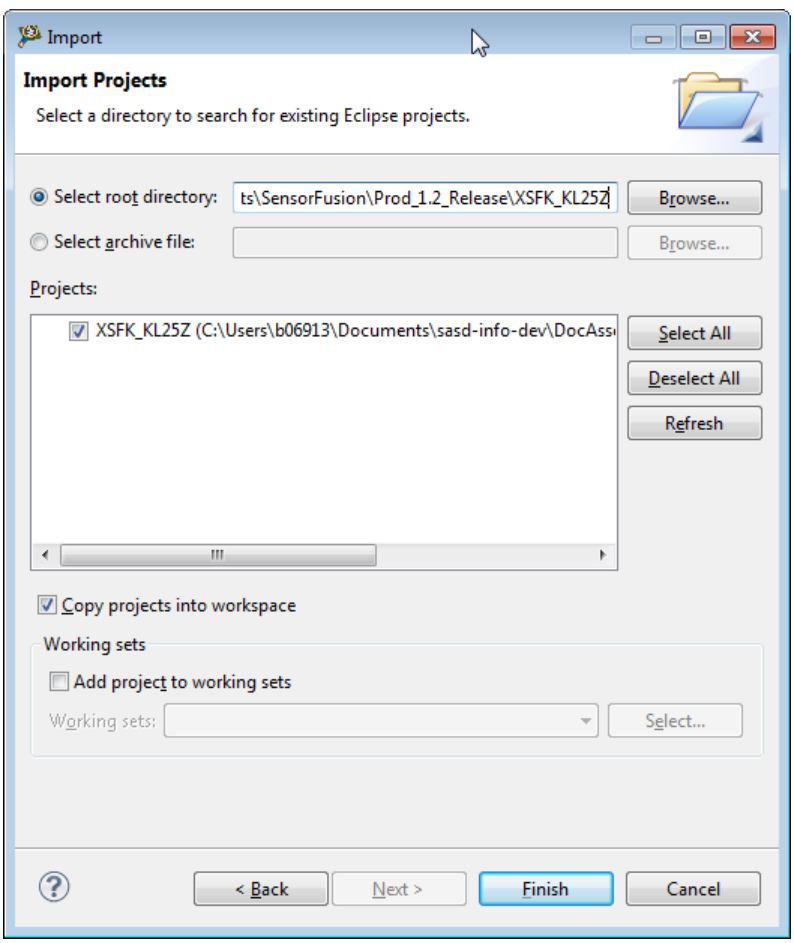


Figure 11. Import Projects window

6. By default, all Freedom Development Platforms projects are configured to be used with the FRDM-FXS-MULTI-B sensor board. When using FRDM-K20D50M or FRDM-K64F with the FRDM-FXS-MULTI board, the Processor Expert UART configuration for the project should be changed. The UART assignment for different combinations of Freedom Development Platform type and sensor board is shown in Table 2. You need to make this change whenever you use the UART/USB wired interface, even with the MULTI-B board.

Table 2. MCU UART selection

MULTI board	FRDM-KL25Z	FRDM-KL26Z	FRDM-KL46Z	FRDM-K20D50M	FRDM-K64F
FRDM-FXS-MULTI	UART0 PTA2:1	UART0 PTA2:1	PTA2:1	UART0 PTB17:16	UART0 PTB17:16
FRDM-FXS-MULTI-B (default)	UART0 PTA2:1	UART0 PTA2:1	PTA2:1	UART1 PTE1:0	UART3 PTC17:16

a. To change the UART assignment in Processor Expert, in the **CodeWarrior Projects** tab, select the appropriate project name for the board you are using. See Figure 12.

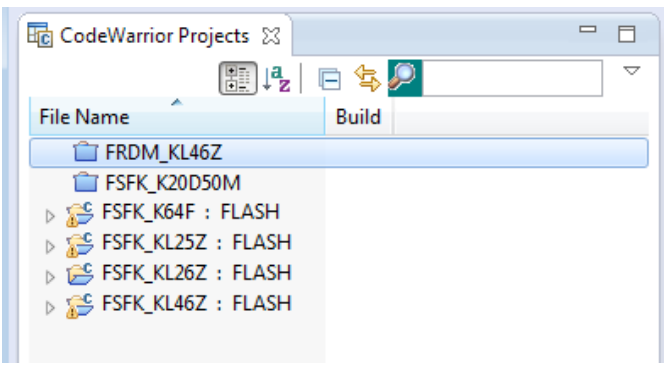


Figure 12. CodeWarrior Projects tab

b. With the project selected, in the **Processor Expert Components** panel, select **UART:Serial_LDD** entry as shown in Figure 13 :

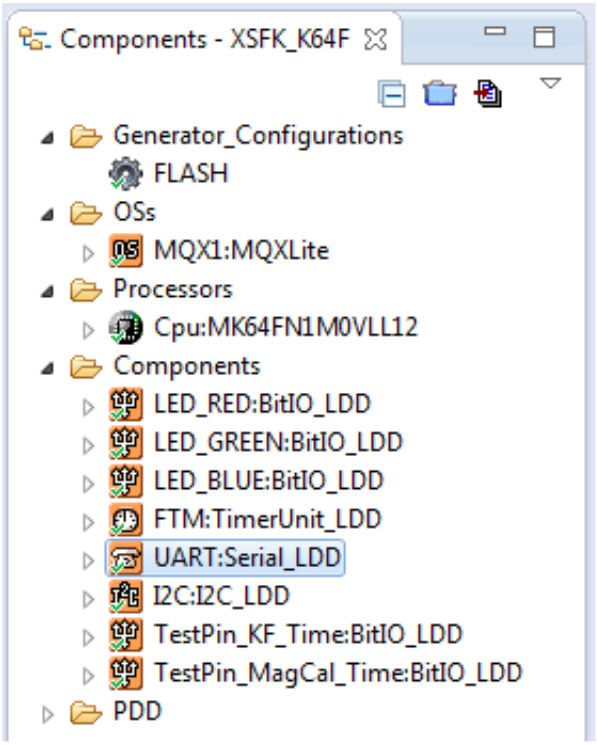
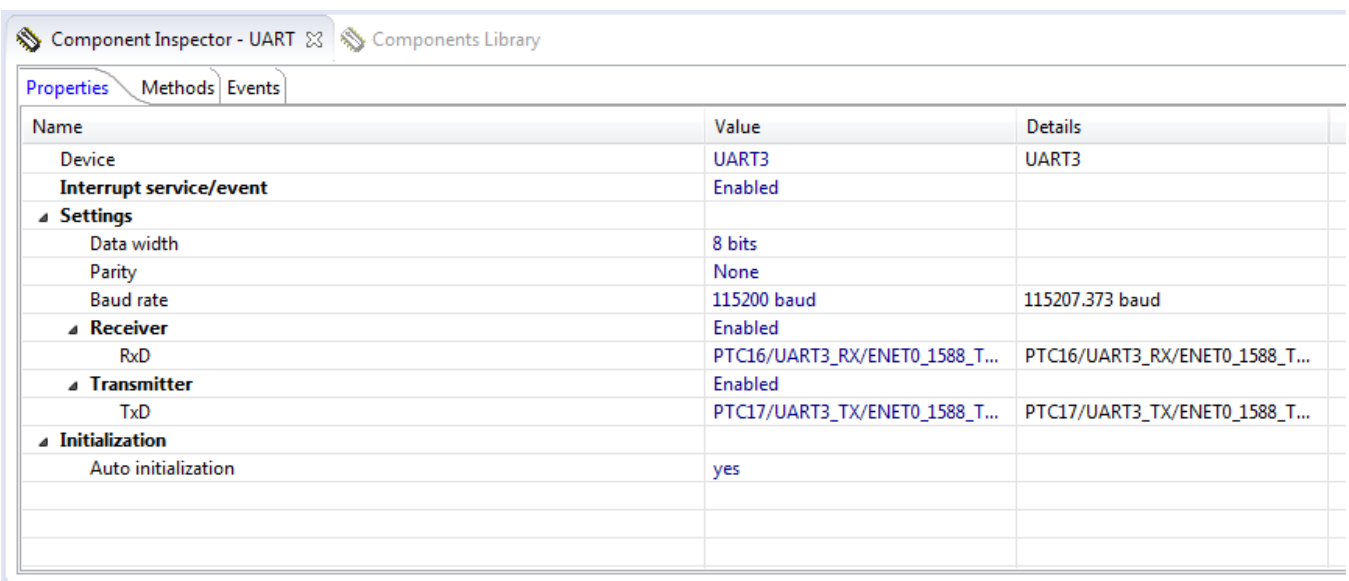


Figure 13. Processor Expert Components for XSFLK_64F panel

c. Under the **Properties** tab, of the **Component Inspector - UART**, select the appropriate UART and its correct Tx and Rx pins for the board you are using according to Table 2. See Figure 14.



Name	Value	Details
Device	UART3	UART3
Interrupt service/event	Enabled	
Settings		
Data width	8 bits	
Parity	None	
Baud rate	115200 baud	115207.373 baud
Receiver	Enabled	
RxD	PTC16/UART3_RX/ENET0_1588_T...	PTC16/UART3_RX/ENET0_1588_T...
Transmitter	Enabled	
TxD	PTC17/UART3_TX/ENET0_1588_T...	PTC17/UART3_TX/ENET0_1588_T...
Initialization		
Auto initialization	yes	

Figure 14. Component Inspector - UART panel

d. In the Components panel, click on the **Generate Processor Expert Code** icon. Make sure the appropriate project name in the CodeWarrior Projects View is selected and then select from the menu **Project->Build Project** to build the project.

7. Follow the instructions in OpenSDA User's Guide listed in Supporting Documentation, to ensure that the Freedom Development Platform is configured for code download from CodeWarrior.
8. Select **Run->Run Configurations** to bring up the Run Configurations dialog, shown in Figure 15.

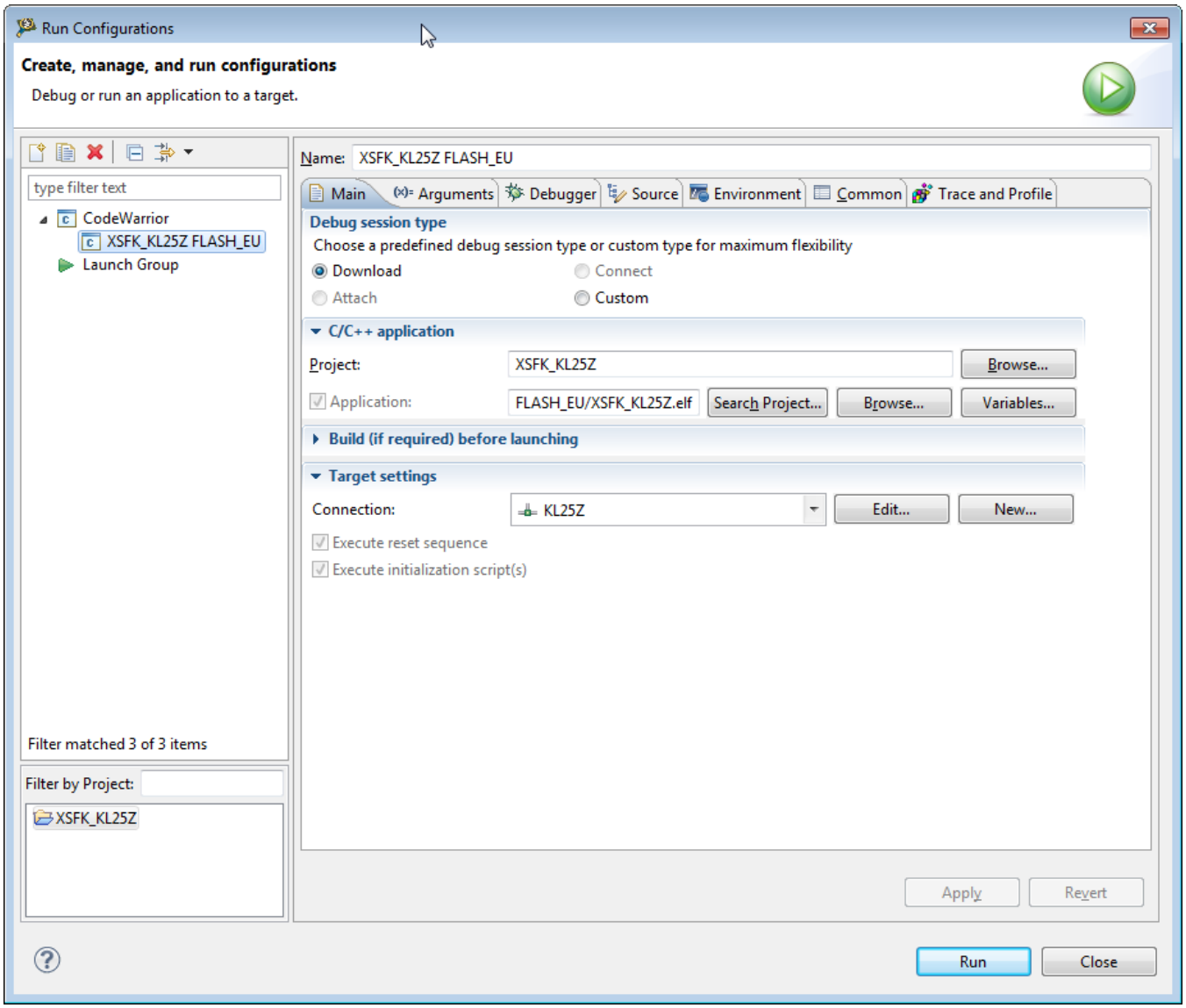


Figure 15. Run Configurations window

9. Click the **Run** button. If the user wishes to use a different configuration, follow instructions in Creating a Run Configuration for OpenSDA 1.0 boards.
10. Proceed to Android Quick Start Guide.

2.1.3 Getting Started with the Firmware or Development with Kinetis Design Studio

The Freescale Sensor Fusion Library also supports the use of the Kinetis Design Studio. Information on KDS can be found at www.freescale.com/kds.

The installation and Getting Started with KDS process is nearly identical to that for CodeWarrior except that some of the screens shown in Getting Started with the Firmware or Development with CodeWarrior will have the Kinetis Design Studio title displayed instead of CodeWarrior as was previously shown.

2.1.4 Limitations

Sensors supported using the development kit out-of-the box include FXOS8700CQ 6-axis accelerometer/magnetometer combination sensor and the FXAS21000 gyroscope. Kalman filters are tuned for specific sensors. Alternate sensor combinations are available, and more can easily be added by making the appropriate additions to drivers.c/.h and build.h.

2.2 Android Quick Start Guide

1. Once running, the Tri-Color LED on the Freedom Development Platform flashes green at a rapid rate. Every once in a while, a red flash will appear when the magnetic calibration routines are running.

   Freedom/sensor shield configurations also include a blue LED next to the BlueRadios module on the shield board. Table 1. For production boards, this LED is OFF when the board is NOT connected to a Bluetooth link. It is ON when it IS connected via Bluetooth to another device.

2. If not already done, download the **Sensor Fusion Toolbox** onto an Android 4.0 or higher device (tablets are desired). This app can be found by searching Google Play, at http://play.google.com for the term *sensor fusion*. Start the app on the Android device once it has been installed. Once the Sensor Fusion Toolbox is open, the software will display as shown in this example, see Figure 16.

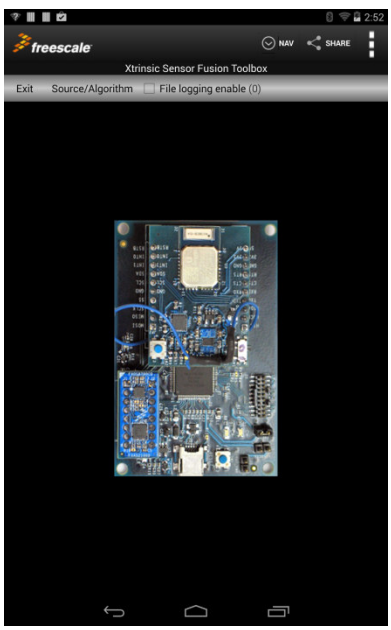


Figure 16. Sensor Fusion Toolbox

3. Open the **Preferences Screen** , see Figure 17 and check the **Automatically enable Bluetooth on entry** checkbox. Click **Save and Exit** option and then exit the application.

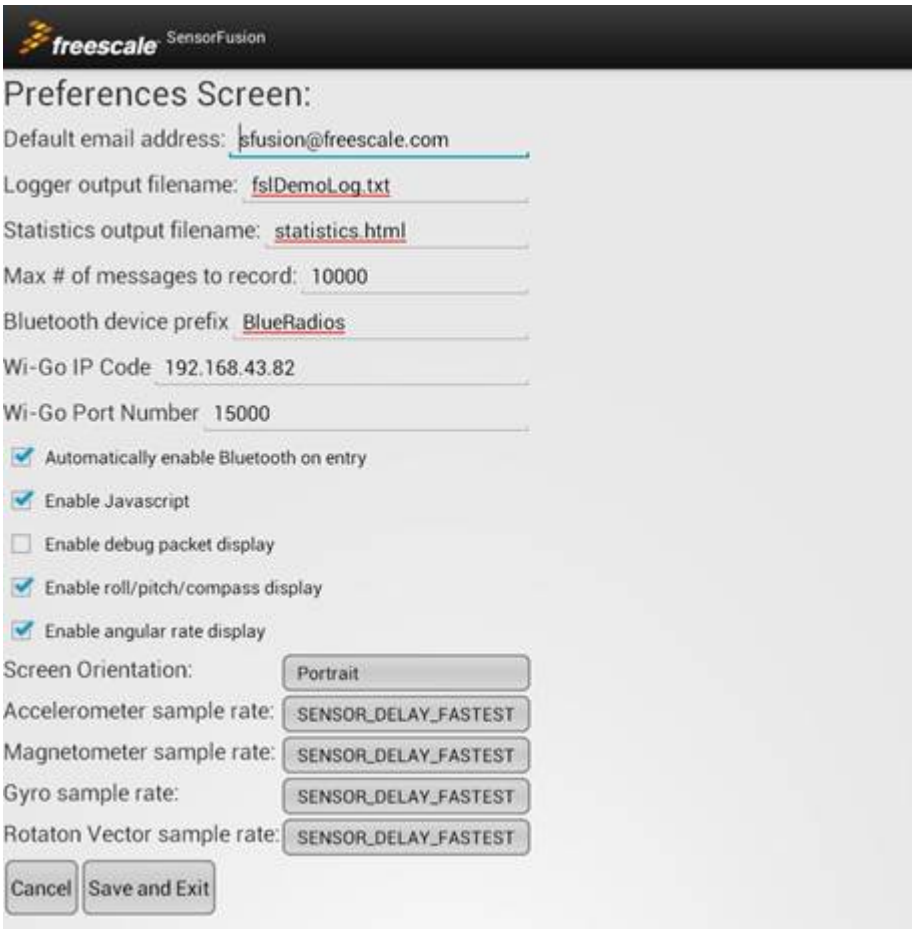


Figure 17. Sensor Fusion preferences window

4. If not already done, pair the development board Bluetooth interface with the Android device. The exact process is different for each model of Android device, but usually this is done in the **Settings->Bluetooth** screen. Search for active devices and look for one that starts with BlueRadios, followed by a 6-digit hex code Pair to that device. It is recommended to pair to a single sensor fusion Freedom Development Platform at a time.
5. Restart the Android app.
6. In the app menu, select **Remote 9-axis** for the Source/Algorithm.
7. Rotate the development board and observe the effects in the Android app display.

2.3 Windows Quick Start Guide

The Sensor Fusion Toolbox supports the Windows operating system, as well as Android OS. The procedure below describes getting started with Windows.

1. This tool requires an OpenSDA CDC Serial Port device driver for proper operation. Consult the OpenSDA documentation from a debug vendor such as (P&E Micro, Segger or MBED) for details.
2. Run the Windows installer (Refer to www.freescale.com/sensorfusion).
   a. Welcome Screen, click **Next**
   b. Choose installation folder, click **Next**
   c. Confirm Installation, click **Next**
   d. Installation is complete, click **Close**
3. Start the Fusion Toolbox
4. File->Flash-><board combination>-><sensor combination>->Android (ENU) Coordinates
5. Attach Freedom/Sensor board stack-up via USB, click **OK** on Instructions dialog box. [1]

6. Select your Freedom board in the resulting Save As dialog box, click **Save**
7. Reset your Freedom Development Platform by unplugging and re-plugging from/to your PC, click **OK**
8. Click the **Auto Detect** button
9. Start using the software.

3 Project Details

This section discusses implementation details for the template programs for Freescale Kinetis microcontrollers. These leverage the MQX™ LITE RTOS and device drivers generated via the **Processor Expert** code generation tool.

3.1 Tasks Structure

Table 3 lists the top level Sensor Fusion software routines.

Table 3. Top Level Sensor Fusion Routines

Sensor Fusion Routines	Description
`RdSensData_Init()`	Performs sensor initialization (one time task)
`RdSensData_Run()`	Performs sensor sampling (high frequency task)
`Fusion_Init()`	Initializes the sensor fusion data structures (one time task)
`Fusion_Run()`	Performs sensor fusion (medium frequency task)
`MagCal_Run()`	Performances magnetic calibration in the background

NOTE

For users who are planning to employ a different RTOS, the functions in this table make good interface points for the new RTOS.

These functions are called from the MQX task definitions, and provide a very high level interface to the fusion library.

3.2 Project Overview

The Sensor Fusion Library is encapsulated using functions in *tasks.c*. see Figure 18 and Figure 19

`RdSensData_Init()` in *tasks.c* calls

- `FXOS8700_Init()`
- `FXAS21000_Init()`

`RdSensDataRun()` in tasks.c calls

- `FXOS8700_ReadData()`
- `FXAS21000_ReadData()`

In both cases, the low level functions can be thought of as the “sensor drivers”. The `Init` functions perform any one-time initialization required by the sensors. The Read functions are responsible for reading sensor values and then populating those to global structures for use by the fusion routines.

1. For a MULTI-B board, be sure to either remove jumper J7 , when using a wired connection, OR pair the device with your PC's Bluetooth adapter. Step 1, above requirement applies only to the wired interface.

Full source code is provided for all layers of software, allowing the user to customize as necessary. Inspect the source code in *drivers.c* to determine if other sensor drivers are available in the release.

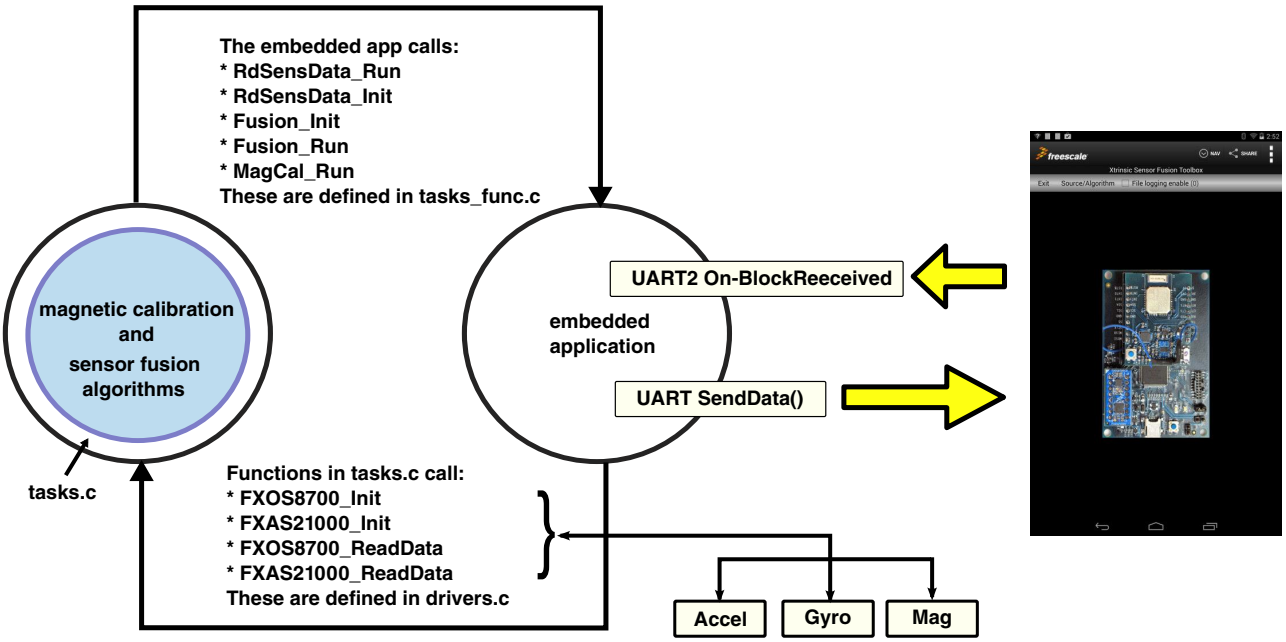


Figure 18. Sensor Fusion Library processing functions

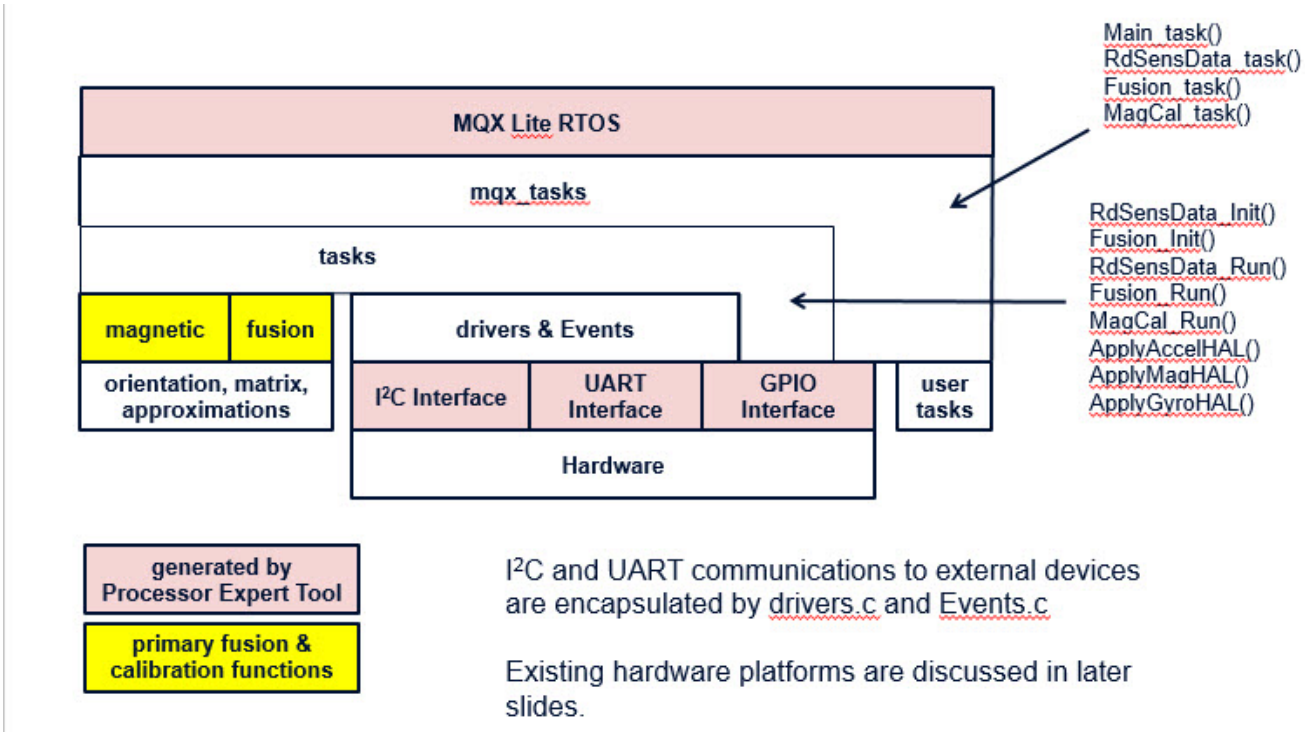


Figure 19. High level architecture

3.3 Global Data Structures

Fusion inputs and outputs are stored in global data structures, which are only allocated if the selected build requires them. Build options used to control the allocation of the global data structures are the following and are contained in *build.h*:

```
#define COMPUTE_3DOF_G_BASIC     // 3-axis accel tilt
#define COMPUTE_6DOF_GB_BASIC    // 6-axis accel + mag eCompass
#define COMPUTE_6DOF_GY_KALMAN   // 6axis accel + gyro Kalman algorithm
#define COMPUTE_9DOF_GBY_KALMAN  // 9-axis Kalman algorithm
```

The library is precompiled with all four options above enabled. This allows ready comparison of the different options in real time. Disabling an option is as easy as commenting out the appropriate #define in *build.h*.

Global data structures are allocated as follows in source file *tasks.c* in response to the build options above. Key structure pointers are shown in the table below.

Table 4. Global Structures

Pointer Function	Structure Name	Structure Type	Defined in Include File
Accelerometer	`thisAccel`	`AccelSensor`	*build.h*
Magnetometer	`thisMag`	`MagSensor`	
Gyroscope	`thisGyro`	`GyroSensor`	
3-axis results	`thisSV_3DOF_G_BASIC`	`SV_3DOF_G_BASIC`	*tasks.h*
eCompass results	`thisSV_6DOF_GB_BASIC`	`SB_6DOF_GB_BASIC`	
accel+gyro results	`thisSV_6DOF_GY_KALMAN`	`SV_6DOF_GY_KALMAN`	
9-axis results	`thisSV_9DOF_GBY_KALMAN`	`SV_9DOF_GBY_KALMAN`	

3.3.1 Writing Sensor Values into Global Structures

The product development version of the library should update global input structures via `ReadAccelMagnData()` and `ReadGyroData()`. Both of these are called via `RdSensData_Run()`, which is itself called from the high frequency MQX task. `RdSensData_Run()` is also responsible for integrating sensor values at an oversample rate defined by OVERSAMPLE_RATIO. See *tasks.c* for details.

3.4 Reading Sensor Values

Note: X, Y and Z are conveniently defined as 0, 1 and 2 for readability in the library. The user should only read averaged/ integrated sensor values from global structures via the sensor fusion (25 Hz) task.

3.4.1 Accelerometer

Accelerometer values are stored as `int16`. Multiply by `thisAccel->fgPerCount` to convert to gravities.

```
float fAcc[3]; // accel sensor output x, y, z
fAcc[X] = (float) thisAccel.iGp[X]*thisAccel.fgPerCount;
fAcc[Y] = (float) thisAccel.iGp[Y]*thisAccel.fgPerCount;
fAcc[Z] = (float) thisAccel.iGp[Z]*thisAccel.fgPerCount;
```

3.4.2 Gyroscope

Gyroscope values are stored as `int16`. Multiply by `thisGyro->fDegPerSecPerCount` to convert to degrees/second.

```
float fAngularRate[3]; // averaged gyro sensor output
fAngularRate[X] = thisGyro.iYp[X]*thisGyro.fDegPerSecPerCount;
fAngularRate[Y] = thisGyro.iYp[Y]*thisGyro.fDegPerSecPerCount;
fAngularRate[Z] = thisGyro.iYp[Z]*thisGyro.fDegPerSecPerCount;
```

3.4.3 Magnetometer

Magnetometer values are stored as int16. Multiply by `thisMag->fCountsPeruT` to convert to µTs. Both raw and calibrated magnetometer outputs are available. Scaling is the same for both.

```
float fBp[3]; // mag - raw average over OVERSAMPLE_RATIO measurements
float fBc[3]; // mag - iBp after calibration
fBp[X] = thisMag.iBp[X]*thisMag.fCountsPeruT;
fBp[Y] = thisMag.iBp[Y]*thisMag.fCountsPeruT;
fBp[Z] = thisMag.iBp[Z]*thisMag.fCountsPeruT;
fBc[X] = thisMag.iBc[X]*thisMag.fCountsPeruT;
fBc[Y] = thisMag.iBc[Y]*thisMag.fCountsPeruT;
fBc[Z] = thisMag.iBc[Z]*thisMag.fCountsPeruT;
```

3.5 Reading Fusion Results

Global Data Structure variables

Table 5 specifies where to find various data items within the global structures maintained by the Fusion Library.

Each of the various fusion options have similar output structures. Use one of the following structure pointers then index according to the following table.

```
thisSV_3DOF_G_BASIC   // OR
thisSV_6DOF_GB_BASIC  // OR
thisSV_6DOF_GY_KALMAN // OR
thisSV_9DOF_GBY_KALMAN
```

Structures within *tasks.h* may be re-arranged in future versions of the library. Index directly to the data you need using the pointer names in Table 5.

Table 5 Fusion Algorithm Options correspond to the following:

- G = SV_3DOF_G_BASIC (accel only)
- GB = SV_6DOF_GB_BASIC (accel + mag eCompass)
- GY = accel + gyro Kalman
- GBY = 9-axis Kalman

Table 5 variable names follow a strict naming convention.

- Core variable names
  - m = magnetic, b = gyro offset, a = acceleration, q = quaternion
  - angle names= Phi, The, Psi, Rho, Chi and Delta
- f prefix = floating point variable
- LP = low pass filtered

- Pl suffix = post apriori estimate from Kalman filter
- Gl suffix = global frame
- Se = sensor frame

Table 5. Location of individual variables within the global structures

Description	data type	Fusion Algorithm Options			
		G (accel)	GB (eCompass)	GY (accel + gyro)	GBY (9-axis)
roll in degrees	float	fLPPhi	fLPPhi	fPhiPl	fPhiPl
pitch in degrees	float	fLPThe	fLPThe	fThePl	fThePl
yaw in degrees	float	fLPPsi	fLPPsi	fPsiPl	fPsiPl
compass heading in degrees	float	fLPRho	fLPRho	fRhoPl	fRhoPl
tilt angle in degrees	float	fLPChi	fLPChi	fChiPl	fChiPl
magnetic inclination angle in degrees	float	N/A	fDeltafLPDelta	N/A	fDeltaPl
geomagnetic vector (µT, global frame)	float	N/A	N/A	N/A	fmGl[3]
gyro offset in degrees/sec	float	N/A	N/A	fbPL[3]	fbPL[3]
linear acceleration in the sensor frame in gravities	float	N/A	N/A	faSePl[3]	faSePl[3]
linear acceleration in the global frame in gravities	float	N/A	N/A	N/A	faGlPl[3] fLPaGlPl[3]
quaternion (unitless)	fquaternion	fqfLPq	fqfLPq	fqPl	fqPl
angular velocity in dps	float	fOmega[3][1]	fOmega[3]	fOmega[3][2]	fOmega[3][2]
orientation matrix (unitless)	float	fR[3][3] fLPR[3][3]	fR[3][3] fLPR[3][3]	fRPl[3][3]	fRPl[3][3]
rotation vector	float	fLPRVec[3]	fLPRVec[3]	fLPRVec[3]	fLPRVec[3]
time interval in seconds	float	fDeltat	fDeltat	fDeltat	fDeltat

1. Yaw is not supported for 3-axis accelerometer.
2. Physical gyro angular rate corrected to subtract computed offset.

Example: 3.5.1 Reading Quaternion Values

The Freescale library uses the convention that q0 = cos(α/2) and $[q1, q2, q3]^T = u \times \sin(\alpha/2)$. Values are stored as float, and can be used directly.

```
struct fquaternion fq;              // quaternion
float q0, q1, q2, q3;

//fq = thisSV_3DOF_G_BASIC.fLPq;    // OR
//fq = thisSV_6DOF_GB_BASIC.fLPq;   // OR
//fq = thisSV_6DOF_GY_KALMAN.fqPl; // OR
fq = thisSV_9DOF_GBY_KALMAN.fqPl;

q0 = fq.q0;
q1 = fq.q1;
q2 = fq.q2;
q3 = fq.q3;
```

Example: 3.5.2 Reading Euler Angles

Euler angles are stored as float with units of degrees.

Using 3-axis model:

```
float roll  = thisSV_3DOF_G_BASIC.fLPPhi;
float pitch =  thisSV_3DOF_G_BASIC.fLPThe;
float yaw = thisSV_3DOF_G_BASIC.fLPPsi;
```

Using 6-axis accel + mag (eCompass) model:

```
float roll = thisSV_6DOF_GB_BASIC.fLPPhi;
float pitch = thisSV_6DOF_GB_BASIC.fLPThe;
float yaw = thisSV_6DOF_GB_BASIC.fLPPsi;
```

Using 6-axis accel + gyro Kalman filter model:

```
float roll = thisSV_6DOF_GY_KALMAN.fPhiPl;
float pitch = thisSV_6DOF_GY_KALMAN.fThePl;
float yaw = thisSV_6DOF_GY_KALMAN.fPsiPl;
```

Using 9-axis Kalman filter model:

```
float roll = thisSV_9DOF_GBY_KALMAN.fPhiPl;
float pitch = thisSV_9DOF_GBY_KALMAN.fThePl;
float yaw = thisSV_9DOF_GBY_KALMAN.fPsiPl;
```

4 Software Tasks

Kinetis template programs for Sensor Fusion Library are partitioned into three software tasks.

Table 6. Software task frequencies and priorities

Task	Frequency	Priority
Sensor sampling, integration and decimation task	Typically runs at 200 Hz and typically decimates the sensor data to lower frequency for the sensor fusion.	This task has the highest priority.
6-axis and 9-axis fusion task	Typically executes at 25 Hz (assuming 200 Hz sensor sampling and 8x OVERSAMPLING)	This task has middle priority.
Magnetic calibration task[1]	Typically executes once per minute.	This task has the lowest priority.

1. The10-element version of the Freescale magnetic calibration library is utilized.

4.1 Task Scheduling

The **Sampling** task is scheduled by the RTOS timer (typically at 200 Hz). The execution of the **Fusion** task is locked to the **Sampling** task by being released to execute after a pre-determined number of iterations of the **Sampling** task. The execution of the **Calibration** task is controlled by the **Fusion** task and is released to execute after a pre-determined number of iterations of the **Fusion** task.

Software Tasks

The accelerometer, magnetometer, and gyro sensors operate asynchronously to the RTOS. Typically, the sensors are configured to run freely at a nominal output data rate of 200 Hz and are polled by the **Sampling** task at 200 Hz.

Figure 20 illustrates the basic structure of the three real-time tasks used by the fusion library and the hierarchical scheduling of the software tasks.

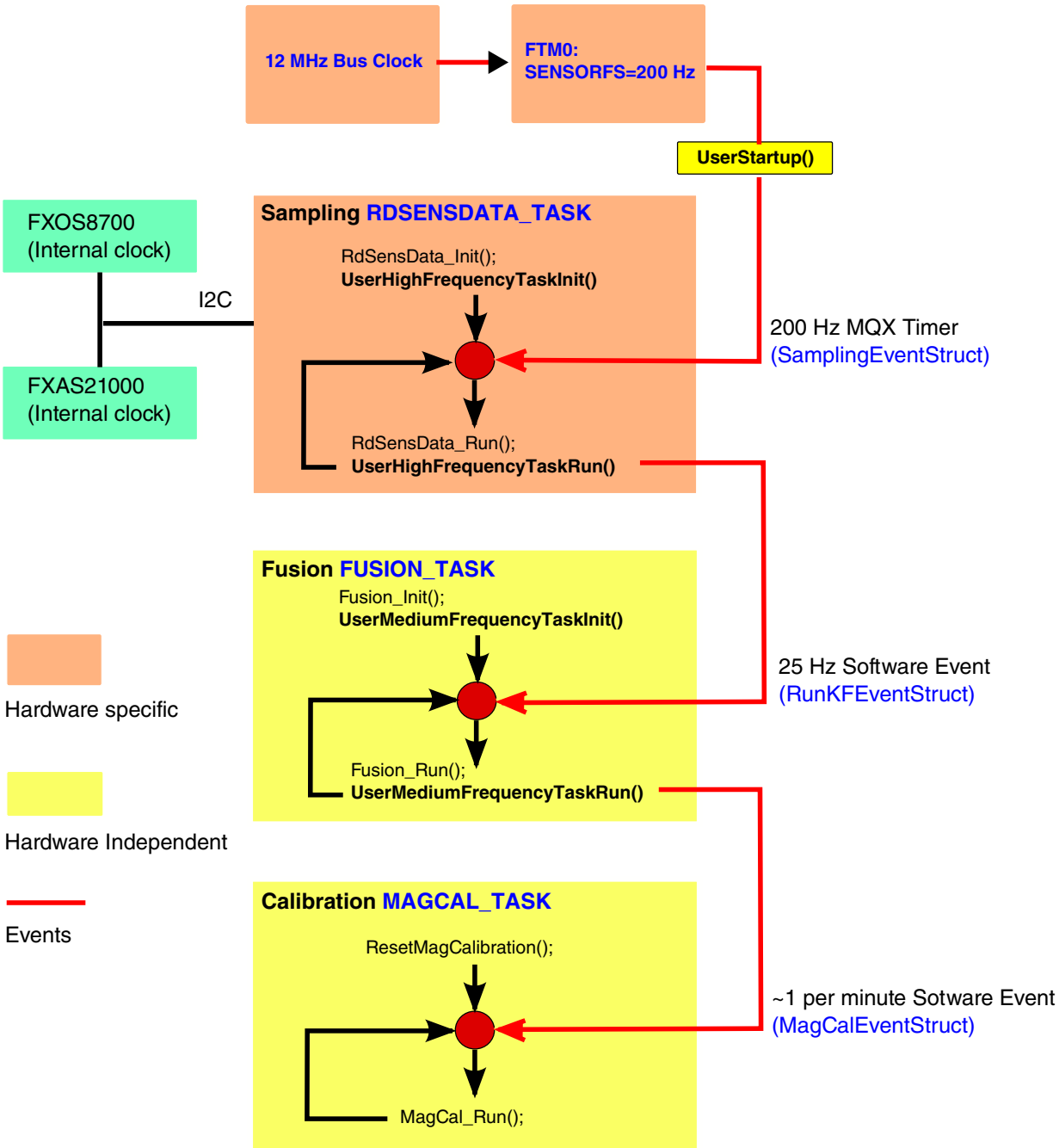


Figure 20. Software Task Scheduling Diagram

The five software function names [UserStartup(), UserHighFrequencyTaskInit(), UserHighFrequencyTaskRun(), UserMediumFrequencyTaskInit() and UserMediumFrequencyTaskRun()] are shown in Figure 20. These functions may be altered by the developer to create additional functionality as described in user_tasks.c.

4.2 Timing Diagram

Figure 21 shows the relative timing and priorities of the sensors and the three software tasks.

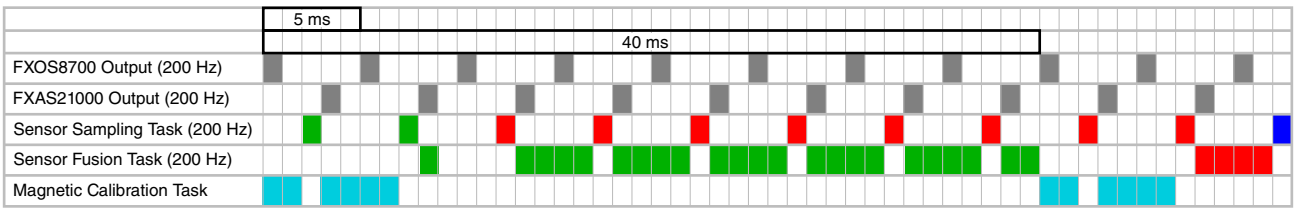


Figure 21. Software Task Timing Diagram

The first two rows show the internal timing of the FXOS8700 and FXAS21000 sensors sampling and latching data at a typical rate of 200 Hz. These two sensors operate asynchronously to each other and asynchronously to the three software tasks executing on the uC. There are, therefore, offsets between the latching of the FXOS8700 output data, the latching of the FXAS21000 output data and their reading by the Sensor Sampling Task. These offsets will slowly drift, resulting in a variable timing error of up to 2.5 ms (at 200 Hz) between the FXOS8700 and FXAS21000 sensor data and a variable latency of up to 2.5 ms before this data is read by the uC sensor **Sampling** Task.

The third row shows the I^2C reads of the 200 Hz sensor **Sampling** task with groups of eight measurements coded in the same color on the assumption that the OVERSAMPLE_RATIO = 8.

The fourth row shows the 25 Hz sensor **Fusion** task processing the groups of eight measurements from the 200 Hz sensor **Sampling** Task. The sensor **Fusion** task starts when the eighth measurement is available from the Sensor **Sampling** Task and continues,with interruptions by the higher priority Sensor **Sampling** Task,until complete.

The magnetic **Calibration** Task has lower priority and executes only when neither the sensor **Sampling** Task nor the sensor **Fusion** Task are executing. In the event that all tasks are complete for a given 200 Hz interval, the software may drop into an idle task, which may, in turn, place the MCU into a temporary STOP mode to save power.

Not shown in Figure 21 are serial communications from the embedded MCU and an external controller and/or GUI. In the case of the supplied board-specific templates, this is comprised of UART communication to an external Bluetooth module. Entry into STOP mode must be deferred until any UART communications are complete (the UART requires that clocks be active for transmission).

Instead, the device will enter WAIT mode until transmission is complete.

4.3 user_tasks.c

The CodeWarrior and KDS template projects successfully build in their as-shipped configurations. However, additional functionality may be supplemented by adding code in the file *user_tasks.c* (see code listing below) wherever `//PUT YOUR CODE HERE` appears. After adding the code, the updated application can be built and downloaded to the development board.

If using the product development version, the file *user_tasks.c* can be considered redundant and may be removed by the developer if desired. Be sure to remove all calls to affected functions before deleting the file.

```
*
* (C) Freescale Semiconductor, Inc.
* user_tasks.c
*
*  Created on: Sep 13, 2013
*      Author: Michael Stanley
*/
```

```
#include "Cpu.h"
#include "Events.h"
#include "mqx_tasks.h"
#include "UART2.h"
#include "include_all.h"

void UserStartup(void)
{
    // The following UART function call initializes Bluetooth communications used by the
    // Freescale Sensor Fusion Toolbox.  If the developer is not using the toolbox,
    // these can be removed.
    //
    // initialize BlueRadios Bluetooth module
    BlueRadios_Init(UART2_DeviceData);

    // put code here to be executed at the end of the RTOS startup sequence.
    //
    // PUT YOUR CODE HERE
    //

    return;
}

void UserHighFrequencyTaskInit(void)
{
    // User code to be executed ONE TIME the first time the high frequency task is run.
    //
    // PUT YOUR CODE HERE
    //
    return;
}

void UserMediumFrequencyTaskInit(void)
{
    // User code to be executed ONE TIME the first time the medium frequency task is run
    //
    // PUT YOUR CODE HERE
    //
    return;
}


void UserHighFrequencyTaskRun(void)
{
    // The default frequency at which this code runs is 200Hz.
    // This code runs after sensors are sampled.
    // In general, try to keep "high intensity" code out of UserHighFrequencyTaskRun.
    // The high frequency task also has highest priority.
    //
    // PUT YOUR CODE HERE
    //
    return;
}

void UserMediumFrequencyTaskRun(void)
{
    // This code runs after the Kalman filter loop
    // The default frequency at which this code runs is 25Hz.

    // The following UART function constructs and sends Bluetooth packets used by the
    // Freescale Sensor Fusion Toolbox.  If the developer is not using the toolbox,
    // it can be removed.
    // transmit orientation over the radio link
    CreateAndSendBluetoothPacketsViaUART(UART2_DeviceData);

    //
    // PUT YOUR CODE HERE
    //
```

```
    return;
}
```

4.4 UART Communications

`UART_Send_Data` is responsible for sending Bluetooth packets to the Fusion Toolbox. Variable array `sUARTOutputBuf[]` contains information to be transmitted to the Android device. The code snippet below, which may be found in drivers.c, illustrates how a DEBUG packet (packet type 0x02) can easily be constructed.

```
// packet start byte
  sUARTOutputBuf[iIndex++] = 0x7E;

// [1]: packet type 2 byte
  tmpuint8 = 0x02;
  sBufAppendItem(sUARTOutputBuf, &iIndex, &tmpuint8, 1);

// [2]: packet number byte
  sBufAppendItem(sUARTOutputBuf, &iIndex, &(globals.iBluetoothPacketNumber), 1);
  globals.iBluetoothPacketNumber++;

// [4-3] software version number
  tmpint16 = THISBUILD;
  sBufAppendItem(sUARTOutputBuf, &iIndex, (uint8*)&tmpint16, 2);

// [5 in practice but can be variable]: add the tail byte for the debug packet type 2
  sUARTOutputBuf[iIndex++] = 0x7E;
```

NOTE

UART/Bluetooth communications are included for ease of use, but they are NOT considered part of the Fusion Library. This portion of the library should be treated as example code which may be changed in future versions of the library. In this section it is presumed that the user is adapting one of the Freescale Fusion Library template projects to a similar Kinetis-based board, also using MQX™ LITE. Ports to other (possibly non-Freescale) MCUs will require the user to write their own sensor drivers and supply their own RTOS layers. Regardless of the MCU/RTOS used, Freescale recommends the user adopt the task structure and outlined in Tasks Structure. The functions discussed in that section provide the cleanest point at which to interface to different hardware/RTOS architectures.

5 Adding Support for a New PCB

The simplifying assumption in this section is that you are adapting one of the template projects to a similar Kinetis-based board, also using MQX™ LITE. Ports to other, possibly non-Freescale, MCUs will require you to write your own sensor drivers and supply your own RTOS layers. Regardless of the MCU/RTOS used, Freescale recommends the user adopt the task structure and outlined in Tasks Structure. The functions discussed in that section provide the cleanest point at which to interface to different hardware/RTOS architectures. This section provides a brief explanation of how to add support in the software for a new PCB.

The fusion library has been implemented on top of basic functions created via Processor Expert. Consult Peripherals Used by the Kit to determine if your PCB design affects any software assumptions. If so, it is best to make those adjustments directly within Processor Expert and then regenerate support functions using Processor Expert.

Table 7 lists files not generated by Processor Expert, that may need to be modified to implement the sensor fusion algorithms on different hardware, with a different RTOS or with different sensors.

Table 7. Source files which may need to be modified

Files	Description
Events.c *Events.h*	Callback functions for hardware events
drivers *drivers.h*	Initialization of hardware timers and I2C drivers for FXOS8700 and FXAS21000 sensors.
mqx_tasks.c *mqx_tasks.h*	Creates and runs the Sampling, Fusion and Calibration tasks which call functions in tasks.c. If the user employs a different RTOS, the Main-task() in mqx_tasks.c will need to be replaced with a function appropriate to their RTOS.
tasks.c *tasks.h*	Contains the functions executed by the three software tasks defined in mqx_tasks.c. It also contains Apply HAL(), which is responsible for applying the PCB-specific hardware abstraction layer.
build.h	Build options consolidated into a single file.
main.c	Initializes and executes MQX.

5.1 File-by-File Changes

Events.c

This file contains event handlers for:

- non-maskable interrupt (currently empty)
- 200 Hz timer
- UART communications

UART communications include function `UART2_OnBlockReceived()`.This implements decode of incoming Bluetooth packets sent by the Freescale Sensor Fusion Toolbox for Android. The user can replace that code with whatever communications is appropriate for their application.

drivers.c

This file is highly project-dependent. It contains:

1. Timer utility functions
2. Sensor initialization[2] and sampling functions , such as sensor drivers
3. UART utility functions
4. `CreateAndSendBluetoothPacketViaUART()`, which is responsible for creating output packets for transmission via Bluetooth.

If the user's project utilizes the same default peripherals and sensors defined in earlier sections, they will probably not need to make any changes to steps 1-3 above. If you use a different communications mechanism or packet structure, item 4 above will need to be replaced.

mqx_tasks.c

This file contains:

- `Main_task()`: Basic initialization of events, timers and peripherals
- `RdSensData_task()`: High frequency sensor sampling
- `Fusion_task()`: Top level control loop for sensor fusion
- `MagCal_task()`: Low frequency task for magnetic calibration

2. If the user is employing non-Kinetis MCUs, this file will probably need to be completely reworked.

If the user's project needs additional MQX-Lite tasks in this file, these tasks will need to be added to this file and `Main_task()` modified to initialize them at startup. Several GPIOs have been pre-allocated for LED control within *mqx_tasks.c*. The user will need to remove or modify those references if those resources are not available in the project.

tasks.c

Each sensor PCB will have a different layout of the accelerometer, magnetometer and gyro sensors, which will require alignment in the Hardware Abstraction Layer (HAL) software.

The function `ApplyHAL()` in the source file *tasks.c* contains the HAL sensor alignment code for supported PCBs and the three supported coordinate systems. The existing software, shown below for one permutation, should be used as a template for the alignment in the new PCB. Detailed instructions on this subject will be presented in an upcoming application note.

```
// BOARD_FRDM_KL25Z and NED
#if (THIS_BOARD_ID == BOARD_FRDM_KL25Z) && (THISCOORDSYSTEM == NED)
    int16 itmp16;
    // accelerometer NED HAL
    itmp16 = thisAccel.iGpFast[X];
    thisAccel.iGpFast[X] = thisAccel.iGpFast[Y];
    thisAccel.iGpFast[Y] = itmp16;
    // magnetometer NED HAL
    itmp16 = thisMag.iBpFast[X];
    thisMag.iBpFast[X] = -thisMag.iBpFast[Y];
    thisMag.iBpFast[Y] = -itmp16;
    thisMag.iBpFast[Z] = -thisMag.iBpFast[Z];
    // gyro NED HAL
    itmp16 = thisGyro.iYpFast[irow][X];
    thisGyro.iYpFast[irow][X] = -thisGyro.iYpFast[irow][Y];
    thisGyro.iYpFast[irow][Y] = -itmp16;
    thisGyro.iYpFast[irow][Z] = -thisGyro.iYpFast[irow][Z];
#endif
```

build.h (Build Parameters)

The file *build.h* contains build options and defines data structures applicable to all source files.

Adding a new PCB

The identifier for the new PCB should be added at the end of the list of sensor PCBs currently supported with a new unique identifier. The constant THIS_BOARD_ID should then be set to the identifier of the new sensor PCB. In the default code below, three boards currently supported are listed and the software will build for the BOARD-FRDM-KL25Z sensor PCB. The board IDs defined here are used by the Freescale Sensor Fusion Toolbox, both Windows and Android, to identify the board images used in the rotating device view.

```
// PCB HAL options
#define BOARD_WIN8_REV05     0     // Reserved for Freescale use

#define BOARD_WIN8_REV05     0            // with shield
#define BOARD_FRDM_KL25Z     1            // with shield
#define BOARD_FRDM_K20D50M   2            // with shield
#define BOARD_FXLC95000CL    3
#define BOARD_FRDM_KL26Z     4            // with shield
#define BOARD_FRDM_K64F      5            // with shield
#define BOARD_FRDM_KL16Z     6            // with shield
#define BOARD_FRDM_KL46Z     7            // with shield
#define BOARD_FRDM_KL46Z_STANDALONE     8    // without shield

// enter new PCBs here with incrementing values
// C Compiler Preprocessor define in the CodeWarrior project will choose which board to use
#ifdef REV05
#define THIS_BOARD_ID        BOARD_WIN8_REV05
#endif
#ifdef KL25Z
#define THIS_BOARD_ID           BOARD_FRDM_KL25Z
#endif
```

```
#ifdef K20D50M
#define THIS_BOARD_ID        BOARD_FRDM_K20D50M
#endif
#ifdef FXLC95000CL
#define THIS_BOARD_ID        BOARD_FRDM_FXLC95000CL
#endif
#ifdef KL26Z
#define THIS_BOARD_ID        BOARD_FRDM_KL26Z
#endif
#ifdef K64F
#define THIS_BOARD_ID        BOARD_FRDM_K64F
#endif
#ifdef KL16Z
#define THIS_BOARD_ID        BOARD_FRDM_KL16Z
#endif
#ifdef KL46Z
#define THIS_BOARD_ID        BOARD_FRDM_KL46Z
#endif
#ifdef KL46Z_STANDALONE
#define THIS_BOARD_ID        BOARD_FRDM_KL46Z_STANDALONE
#endif
```

Coordinate System

The coordinate system to be used is set by THISCOORDSYSTEM with valid values being NED, ANDROID and WIN8. The quaternion representing the orientation transmitted to the Freescale Sensor Fusion Toolbox for Android is the same for all three coordinate systems, but the Euler angles and sensor values are different.

```
// coordinate system for the build (3 permutations)
#define NED 0                       // identifier for NED angle output
#define ANDROID 1                   // identifier for Android angle output
#define WIN8 2                      // identifier for Windows 8 angle output
#define THISCOORDSYSTEM ANDROID     // the coordinate system to be used
```

Orientation Models

The software is capable of running the 3-axis (accelerometer only), 6-axis (accelerometer and magnetometer eCompass) and 9-axis (accelerometer, magnetometer and gyro sensors) in parallel. The default setting computes the 6-axis and 9-axis orientation because the Bluetooth interface transmits the 6-axis and 9-axis orientation, but not the 3-axis orientation quaternion, to the Android tablet for display. For a pure 9-axis application, define only `COMPUTE_9DOF_GBY_KALMAN` in the list below.

```
// degrees of freedom algorithms to be executed
#define COMPUTE_1DOF_P_BASIC // 1DOF pressure (altitude) and temperature
#define COMPUTE_3DOF_G_BASIC // 3DOF accelerometer tilt (basic algorithm)
#define COMPUTE_3DOF_B_BASIC // 3DOF magnetometer compass (basic vehicle algorithm)
#define COMPUTE_3DOF_Y_BASIC // 3DOF gyro integration (basic algorithm)
#define COMPUTE_6DOF_GB_BASIC // 6DOF accelerometer and magnetometer eCompass (basic
algorithm)
#define COMPUTE_6DOF_GY_KALMAN // 6DOF accelerometer and gyro (Kalman algorithm)
#define COMPUTE_9DOF_GBY_KALMAN // 9DOF accelerometer, magnetometer and gyro (Kalman
algorithm)
```

Timing and Filters

SENSORFS defines the execution rate of the sensor sampling task in Hz. The sensor readings are averaged and decimated to a rate OVERSAMPLE_RATIO lower before passing to the Kalman filter. The default values of 200 and 8 respectively result in the sensors being sampled at 200 Hz and the Kalman filter executing at 25 Hz.

```
// sampling rate and kalman filter timing
#define SENSORFS 200         // integer frequency (Hz) of sensor sampling process
#define OVERSAMPLE_RATIO 8   // integer 3-axis, 6-axis, 9-axis run at
                                  // SENSORFS / OVERSAMPLE_RATIO Hz
```

main.c

This file contains the application's `main()` function which performs microcontroller initialization and starts the MQX Lite operating system. This function should be replaced with equivalent processor initialization and operating startup code if either is changed.

6 Bluetooth Packet Structure

The tables presented in this section are the same as those included with the Freescale Sensor Fusion Toolbox for Android in-app help.

Prior to version 2013.07.18 of the Freescale Sensor Fusion Toolbox for Android application, communication between the Android device and external board was strictly one way, from board to Android device. Version 2013.07.18 and above add the ability for the application to send commands to the development board. This is done at application start-up for flags to enable the following:

- debug mode
- roll/pitch/compass display on the Device view
- virtual compass display on the Device view

The debug mode is now required to be always on, since this packet transmits the firmware version number that is displayed by the Toolbox. Additionally, a packet may be sent to change quaternion type whenever the Source/Algorithm selector in the Sensor Fusion Toolbox is changed.

6.1 Development board to Fusion Toolbox

The development board to Android protocol is a streaming data protocol (using RFCOMM). Packets are delimited by inserting a special byte (0x7E) between packets. This means that we must provide a means for transmitting 0x7E within the packet payload. This is done on the transmission side by making the following substitutions:

- Replace 0x7E by 0x7D5E (1 byte payload becomes 2 transmitted)
- Replace 0x7D by 0x7D5D (1 byte payload becomes 2 transmitted)

The Fusion Toolbox does the inverse mapping as the data stream is received. Partial packets are discarded. The options menu has a Toggle Hex Display option available for developers coding their own board interface.

On the embedded app side, the 0x7E and 0x7D encoding is performed automatically by function `sBufAppendItem()`, which is used to create outgoing packet structures. The developer only needs to explicitly add starting and ending 0x7E delimiters.

6.1.1 Packet Type 1

This is the primary data packet format utilized by the Fusion Toolbox.

Table 8. Current version of packet type 1

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 01	None
2	Packet #	This number increments by 1 for each sample. Rolls over at 0xFF to 0x00.
6:3	Timestamp	1 LSB = 1.00 microseconds (Previously 1.33. Updated 2013.07.18)
8:7	ACC X	1 LSB = 122.07 µg

Table continues on the next page...

Table 8. Current version of packet type 1 (continued)

Byte #	Function	Units
10:9	ACC Y	1 LSB = 122.07 µg
12:11	ACC Z	1 LSB = 122.07 µg
14:13	MAGX	1 LSB = 0.1 µT
16:15	MAGY	1 LSB = 0.1 µT
18:17	MAGZ	1 LSB = 0.1 µT
20:19	GYRO X	1 LSB = 872.66 µrad/s (0.05 dps)
22:21	GYRO Y	1 LSB=872.66 µrad/s (0.05 dps)
24:23	GYRO Z	1 LSB=872.66 µrad/s (0.05 dps)
26:25	quaternion q0	1 LSB=1/30,000 (unitless)
28:27	quaternion q1	1 LSB=1/30,000 (unitless)
30:29	quaternion q2	1 LSB=1/30,000 (unitless)
32:31	quaternion q3	1 LSB=1/30,000 (unitless)
33	Flags	Bit field with the following bit definitions: Bit 0 = valid gyro data Bit 1 = gyro is virtual Bit 2 = 6-axis quaternion is valid Bit 3 = 9-axis quaternion is valid Bits 5:4 = 00 = Data is NED Frame of Reference Bits 5:4 = 01 = Data is Android Frame of Reference Bits 5:4 = 10 = Data is Windows Frame of Reference Bits 5:4 = 11 = RESERVED Bits 7:6 = RESERVED
34	Board ID	Numeric field with the following possible values: 0 = Freescale Windows8 Sensor Platform Rev 0.5 1 = FRDM-KL25Z with Freescale Sensor Shield 2 = FRDM-K20D50M with Freescale Sensor Shield 3 = FXLC95000CL with Freescale Sensor Shield 4 = FRDM-KL26Z with Freescale Sensor Shield All others reserved Contact sfusion@freescale.com if the user is interested in having another board added to this list.
35	Packet END = 0x7E	None

Table 8 represents the packet format adopted by the Fusion Toolbox on 1 August 2013. The assumption inherent in this format is that the application will instruct the development board with regard to desired algorithm.

6.1.2 Packet Type 2: Debug

The development board may send 1 to n 16-bit words to the app for display on the Device view. This view is enabled via a checkbox in the Preferences screen.

Table 9. Debug Packet Format

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 0x02	Bytes 6:5 now carry the sysTick count/20
2	Packet Number	None
4:3	Software Version Number	None
...	...	...
2n + 1:2n	Debug Wordn	Last of *n* debug words to transmit
2n + 2	Packet END = 0x7E	None

6.1.3 Packet Type 3: Angular Rate

This packet type is used to send angular rate values to the Android app. This view is enabled via a checkbox in the Preferences screen.

Table 10. Packet Type 3: Angular Rate

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 0x03	None
2	Packet #	This number increments by 1 for each sample. Rolls over at 0xFF to 0x00.
6:3	Timestamp	1 LSB = 1.00 microseconds (Previously 1.33. Updated 2013.07.18)
8:7	X	1 LSB = 872.66 micro-radians/sec (0.05 dps)
10:9	Y	1 LSB = 872.66 micro-radians/sec (0.05 dps)
12:11	Z	1 LSB = 872.66 micro-radians/sec (0.05 dps)
13	Packet END = 0x7E	None

6.1.4 Packet Type 4: Euler Angles

This packet type is used to send roll/pitch/compass heading information to the Android app. This view is enabled via a checkbox in the Preferences screen.

Table 11. Packet type 4 - Roll/Pitch/Compass

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 0x04	None
2	Packet #	This number increments by 1 for each sample. Rolls over at 0xFF to 0x00.
6:3	Timestamp	1 LSB = 1.00 microseconds (Previously 1.33. Updated 2013.07.18)
8:7	X	1 LSB = 0.1degree
10:9	Y	1 LSB = 0.1degree

Table continues on the next page...

Table 11. Packet type 4 - Roll/Pitch/Compass (continued)

Byte #	Function	Units
12:11	Z	1 LSB = 0.1degree
13	Packet END = 0x7E	None

6.1.5 Packet Type 5: Altitude and Temperature

This packet type is used to send altitude and temperature information to the Android app. This view is enabled via a checkbox in the Preferences screen.

Table 12. Packet type 5 - Altitude and Temperature

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 0x05	None
2	Packet #	This number increments by 1 for each sample. Rolls over at 0xFF to 0x00.
6:3	Timestamp	1 LSB = 1.00 microseconds
10:7	Altitude	1 LSB = 0.001 metres
12:11	Temperature	1 LSB = 0.01 degree
13	Packet END = 0x7E	Packet type 6 is reserved for transmitting magnetic buffer information. This is used by the Windows version of the Toolbox, but not by the Android version. This is so application specific that Freescale is not publishing it. Packet type 7 is reserved for transmitting kalman filter information. Use the same notations.

6.2 Toolbox to Freedom Development Platform

Up until version 2013.07.18 of the Fusion Toolbox, communication was strictly one way: from development platform to Android device. Versions released after that date include limited ability to configure the embedded board from Android. The command protocol below is preliminary, and is subject to change.

Possible 4-byte commands are shown below. Substitute a space for each “#” to enforce the 4-byte width.

- ALT+ = Altitude/Temperature packet on
- ALT- = Altitude/Temperature packet off (default) [3]
- DB+# = debug packet on (default) (transmitted via "Options Menu->Enable debug")
- DB-# = debug packet off (transmitted via "Options Menu->Disable debug")
- Q3## = transmit 3-axis quaternion in standard packet (transmitted when "Remote accel" is selected on the Source/Algorithms spinner)
- Q6MA = transmit 6-axis mag/accel quaternion in standard packet (transmitted when "Remote mag/accel" is selected on the Source/Algorithms spinner)
- Q6AG = transmit 6-axis accel/gyro quaternion in standard packet (transmitted when "Remote accel/gyro" is selected on the Source/Algorithms spinner)

3. The debug packet must be enabled for firmware version number and sysTick counts to be properly displayed by the Toolbox.

- Q9## = transmit 9-axis quaternion in standard packet (default) (transmitted when "Remote 9axis" is selected on the Source/Algorithms spinner)
- RPC+ = Roll/Pitch/Compass packet on
- RPC- = Roll/Pitch/Compass packet off (default)
- RST# = Sensor Fusion soft reset (resets all sensor fusion data structures)
- VG+# = Virtual gyro packet on
- VG-# = Virtual gyro packet off (default)

Note that the commands above can request that the embedded board perform computations in a specific way. Confirm that the proper operation has taken place by checking the Flags field in Packet type 1. See the flags row of Table 8

7 Odds and Ends

7.1 ANSI C

The Sensor Fusion Library software is written according to the ANSI C90 standard and does not use any of the ANSI C99 or C11 extensions.

Integers are a mixture of standard C long (four byte) integers with typedef as int32 and C short (2 byte) integers with typedef as int16. Floating point variables are all single precision of size four bytes.

7.2 Floating Point Libraries

The Sensor Fusion Library software uses single precision floating point arithmetic. C functions, like `sqrt()`, which return a double are immediately cast back into single precision.

Floating point arithmetic is performed using software emulation on the processors with integer cores or directly on the FPU by processors with an internal FPU.

7.3 Error Handling

The Sensor Fusion Library software is believed incapable of triggering exceptions. All conditions that could lead to exceptions are detected before the relevant function call is made. Conditions tested to prevent runtime exceptions are:

- division by zero, whether floating point or integer
- inverse sine or cosine of an argument outside the range –1 to +1
- tangent of –90 degrees or 90 degrees angles
- square root of a negative number.

7.4 Numerical Accuracy

The Sensor Fusion Library software is numerically accurate to the limits of single precision floating point arithmetic.

Small angle approximations are only used when the relevant angle is, in fact, small so no accuracy is lost compared to using the standard libraries. For example, sines and cosines of small angles are computed with a MacLaurin series to give similar accuracy to the standard sine and cosine libraries but at lower computational expense.

Conditions that lead to inaccurate results are detected and the results computed using an alternative algorithm suitable for those cases where the primary algorithm fails. An example is the calculation of the orientation quaternion from a rotation matrix where the primary algorithm differencing elements across the diagonal approaches a 0/0 calculation. The alternative algorithm used near 180 degree rotations operates on the matrix diagonal elements instead.

The sensor fusion algorithms have been tested to be stable for tens of millions of 200 Hz iterations. Accumulation of rounding errors in the rotation quaternions or matrices is prevented by regular renormalization.

8 Revision History

Revision number	Revision date	Description
0	02/2014	Initial version
1	04/2014	Updated license scheme documentation. Demo and (new board-specific) Production licenses explained. Quick start instructions for software installation via the installer added. The software revision includes many enhancements - the major ones are as follows: 1. Added MPL3115 I2C driver. I2C driver for MPL3115 now detects whether MPL3115 is present so as to support the 9 axis board 2. Debug packet (Type 2) transmits firmware build in place of padding byte 3. Added Altitude / Temperature packet 4. Added Kalman packet 5. Added Magnetics packet 6. Added FXAS21002 driver and sensor option 7. Support for Sequential/Parallel algorithm 8. RST command implemented 9. Code and comments cleaned up and made clearer throughout 10. ALT+/ALT- commands added to support Altitude packet 11. Global variable definitions are now consolidated into two structs named `globals` and `mqxglobals`.
1.1	05/2014	Updated documentation for additional board support of FRDM-K64F. Added instructions for UART selection based on the MULTI board being used.
1.2	01/2015	1. Removed all references to license checks 2. The fusion library is now open source 3. Added KL46Z support 4. Added KDS support

Appendix A

A.1 Creating a Run Configuration for OpenSDA 1.0 boards

The user must have a valid run configuration created within CodeWarrior in order to download code to the Freedom Development Platform. The following sequence shows how to create one if you do not already have it. See Figure A-1.

1. Select **Run->Run Configurations**. The user should see a window similar to the screenshot below:

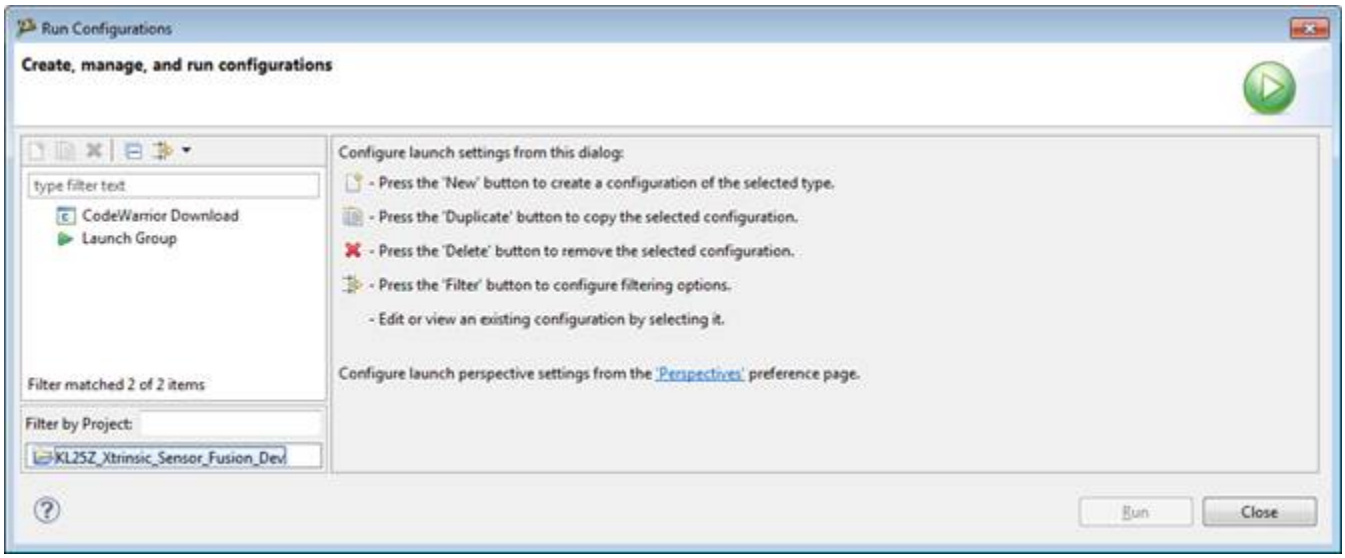


Figure A-1. CodeWarrior Run Configuration window

2. Select **CodeWarrior Download** and then click the **New Launch Configuration** icon on the left, see Figure A-2.

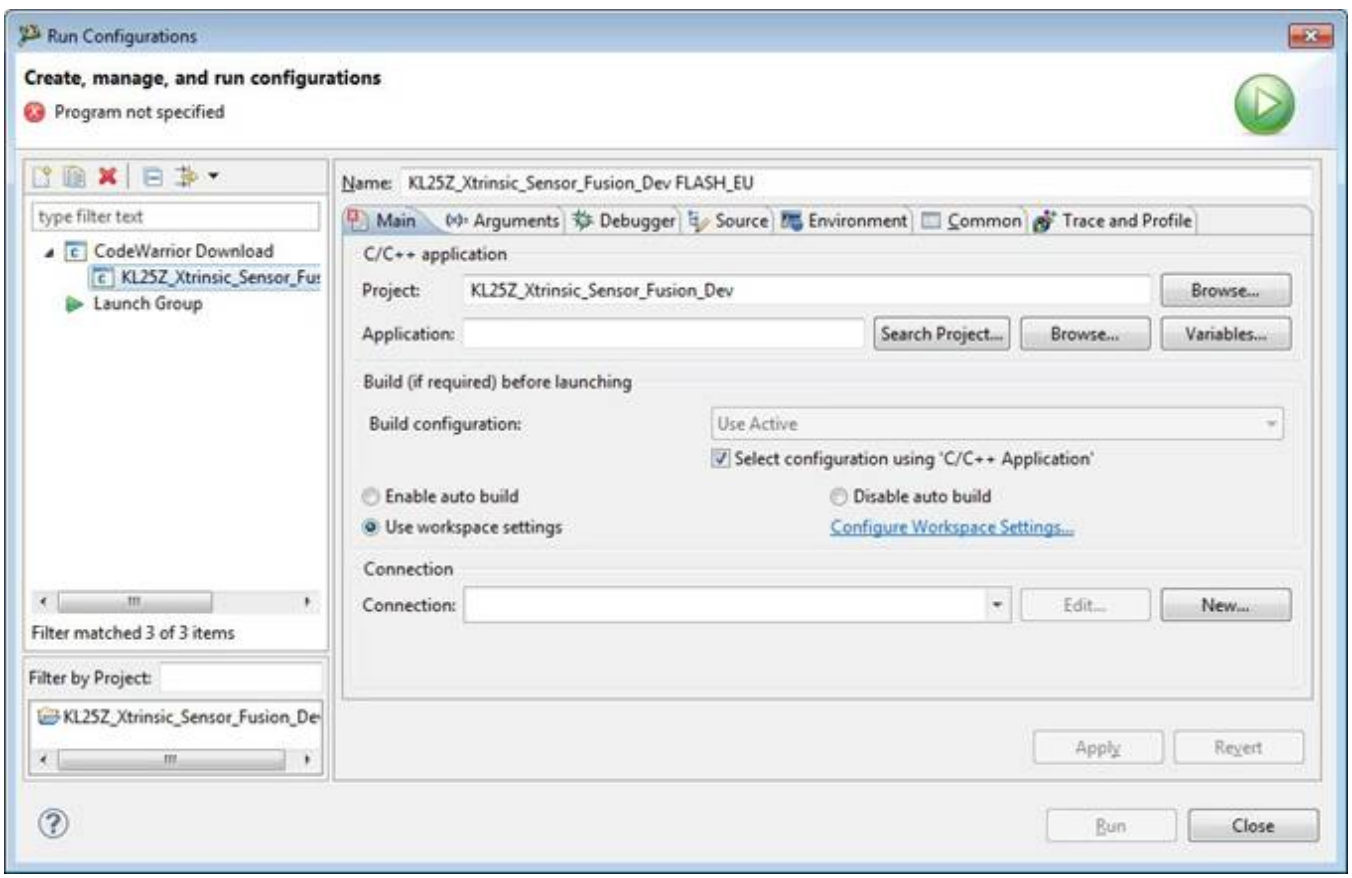


Figure A-2. CodeWarrior creating new launch configuration

3. Click the **New…** button to the right of the Connection field, see Figure A-2.
4. Select Hardware or Simulator Connection, and Click **Next**. See Figure A-3)

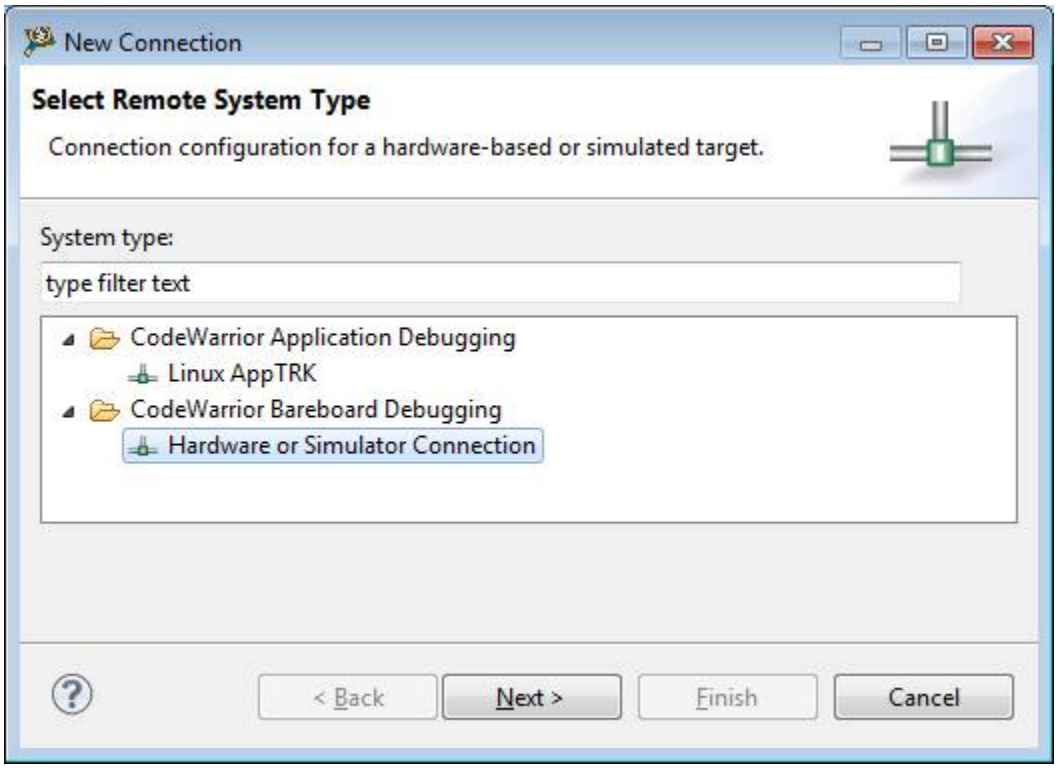


Figure A-3. Setting CodeWarrior Remote System Type

5. Enter a valid connection name in the name field, see Figure A-4.

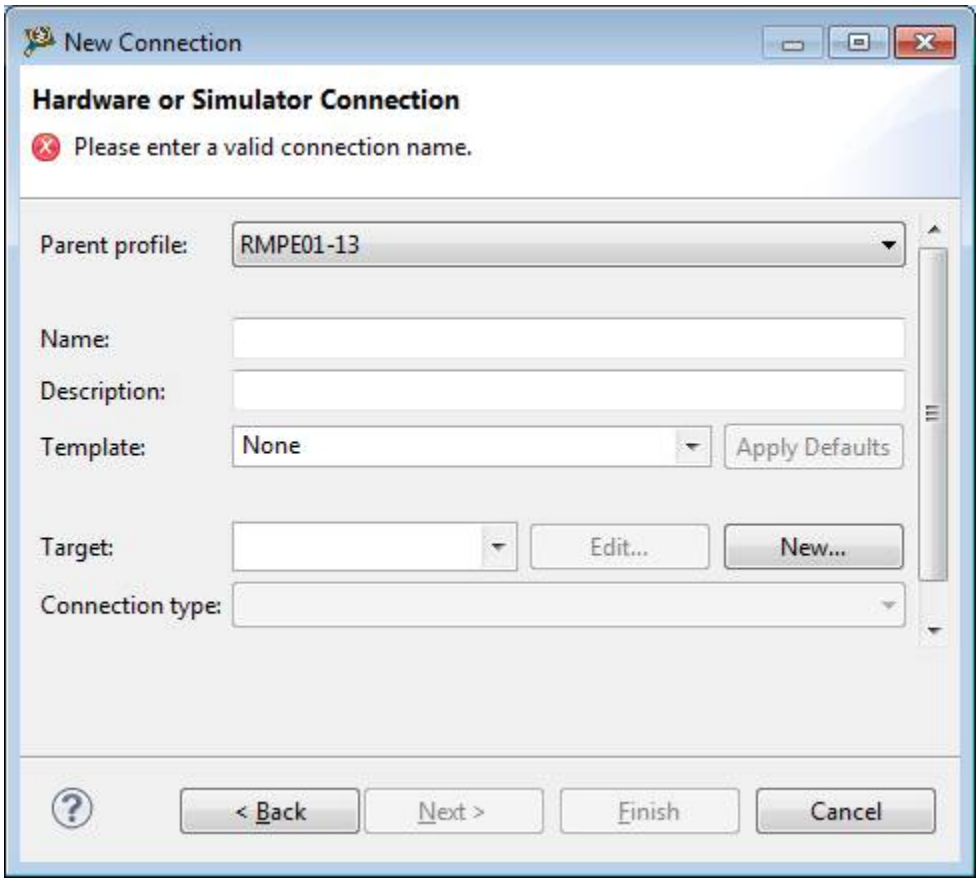


Figure A-4. Naming new CodeWarrior connection

6. In the Target type field click **New...**, see Figure A-4.
7. Click the Edit button on the **Target type** field, see Figure A-5.

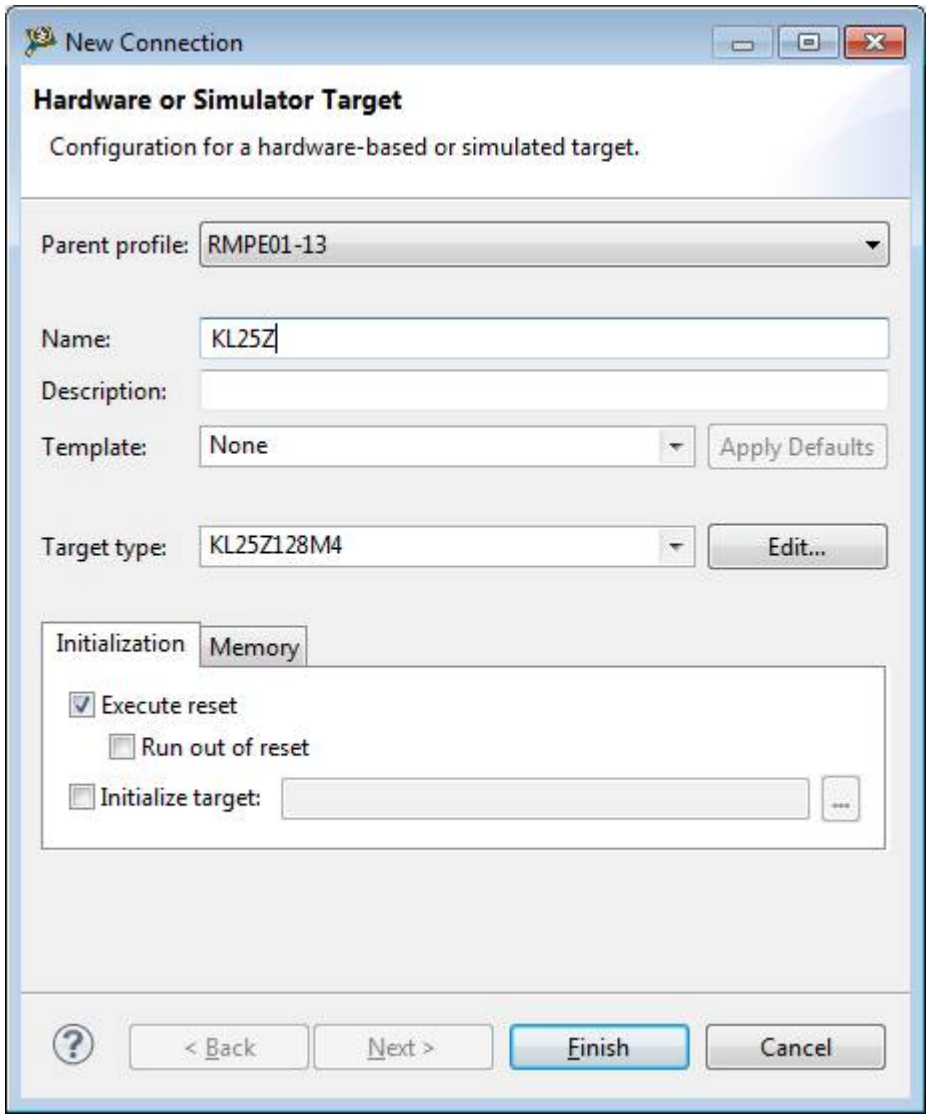


Figure A-5. Selecting CodeWarrior Target type

If you have a KL25Z board, scroll down to KL25Z, expand the sub-list and select KL25Z128M4.
If you have a K20D50M board, scroll down to K20D, expand the sub-list and select K20DX128.

8. Check the "**Execute reset**" check box, (see Figure A-5) then enter a value in the name field and click **Finish**.

NOTE

If issues are encountered with a connection created in this way and a new connection needs to be recreated, using the recommended settings shown in Figure A-6.

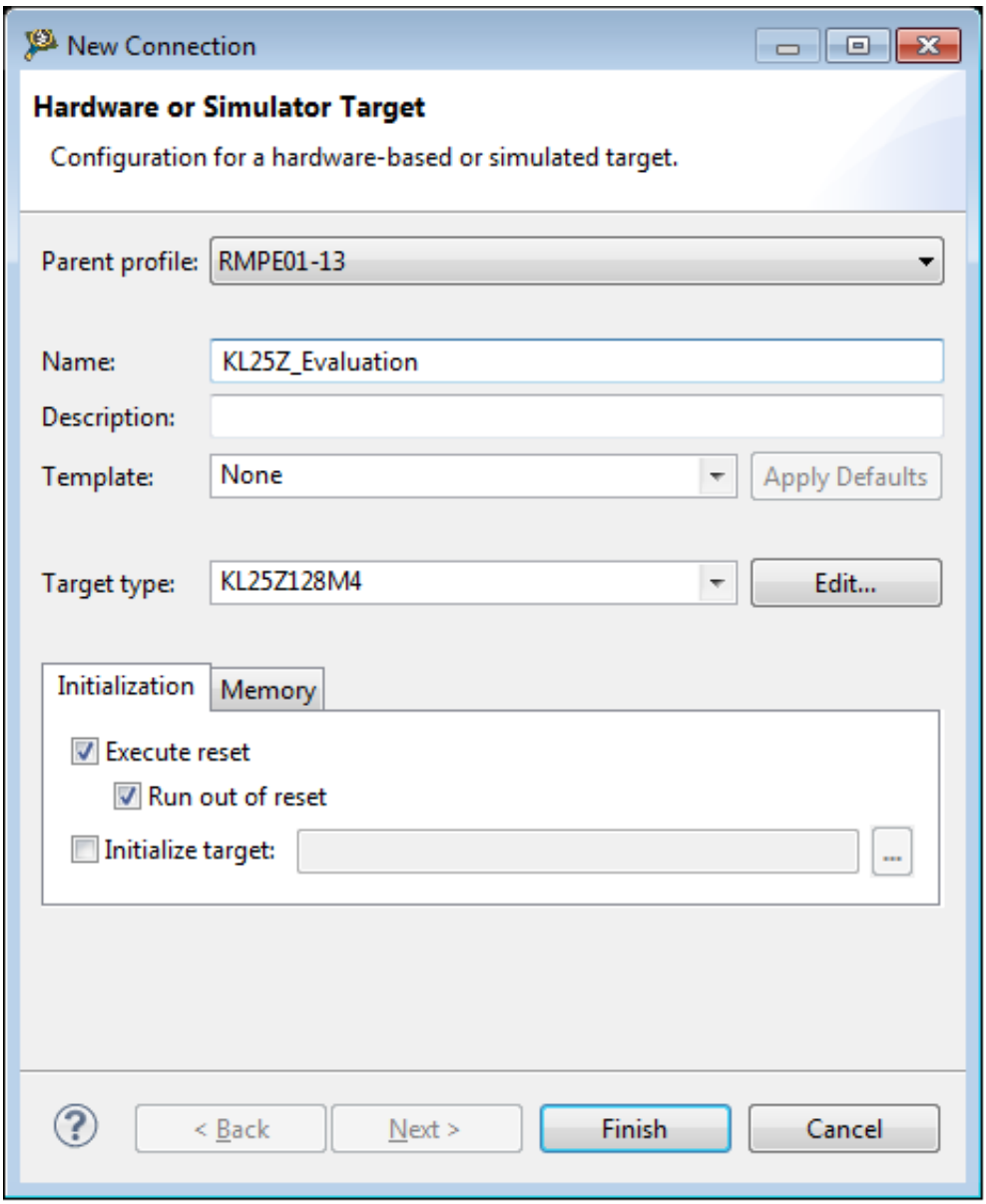


Figure A-6. CodeWarrior new connect example setup

9. Configure the **New Connection** dialog as shown in Figure A-7. (If you have a K20D50M board, change KL25Z to K50D50M) and then click the **Finish** button. The focus will return to the **Run Configuration** dialog.

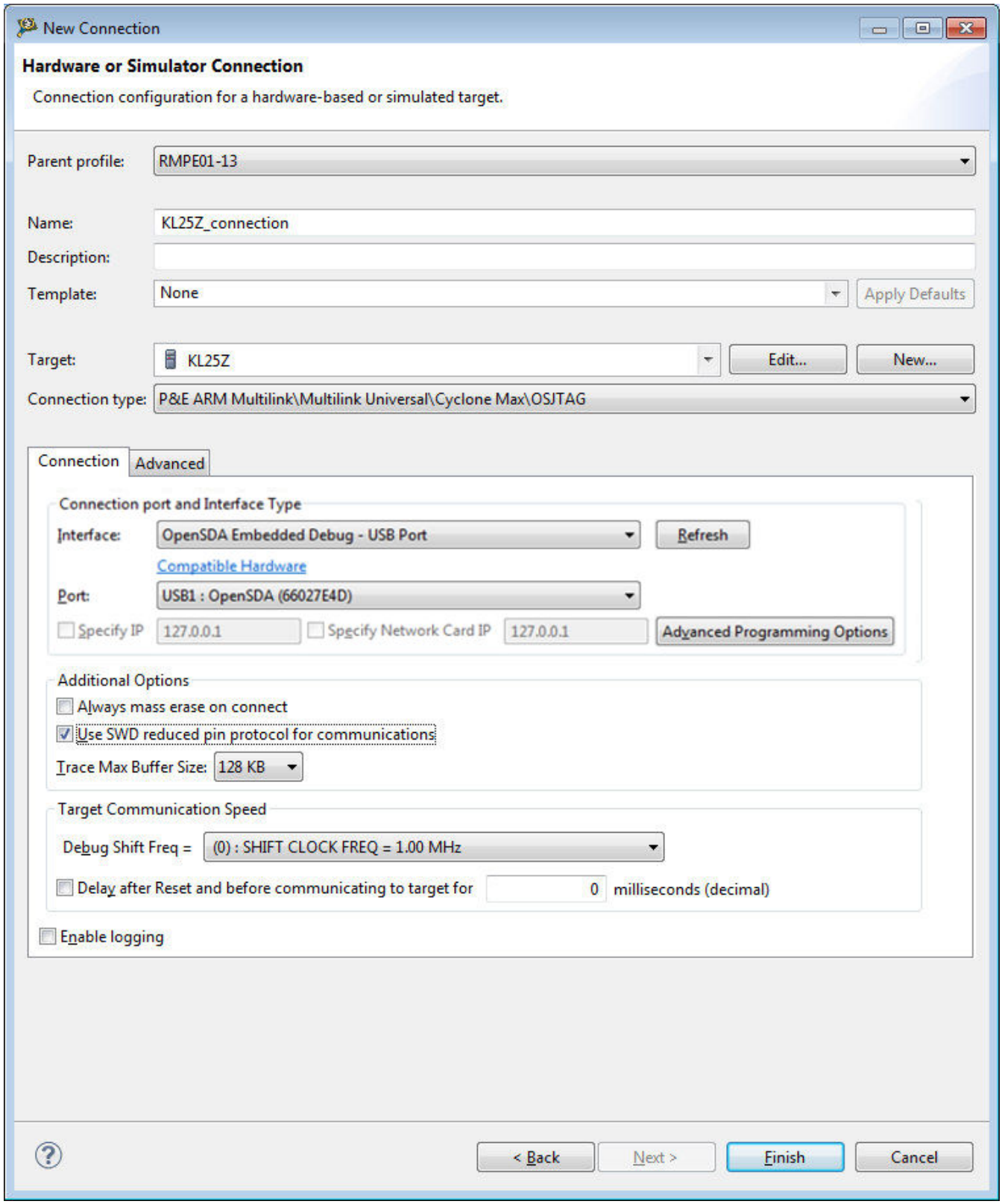


Figure A-7. CodeWarrior run configuration dialog

10. Click **Finish**
11. Modify the **Application** field on the dialog (shown in Figure A-8) to include the elf file (produced during the **Build** step) to download.

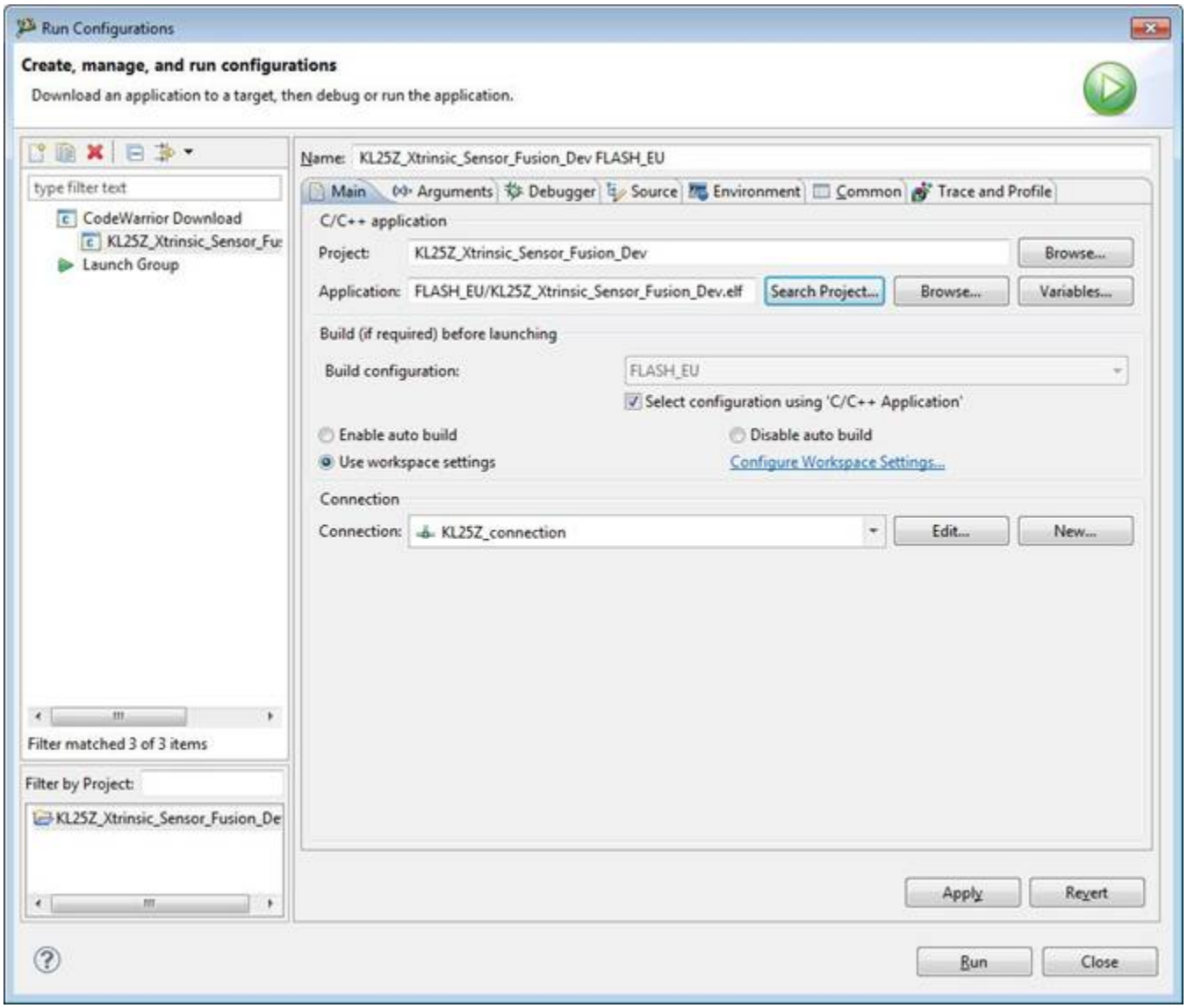


Figure A-8. Setting the elf file in CodeWarrior

12. Click **Apply** and **Run** to download code to the board.

A.2 Creating Run Configuration for Segger OpenSDA

The user must have a valid run configuration created within CodeWarrior in order to download code to their development board. The following sequence shows how to create one if you do not already have one for the Segger OpenSDA version 2.0 used with the Freedom Development Platform K64F. Begin by accepting the software terms of use statement, see Figure A-9.

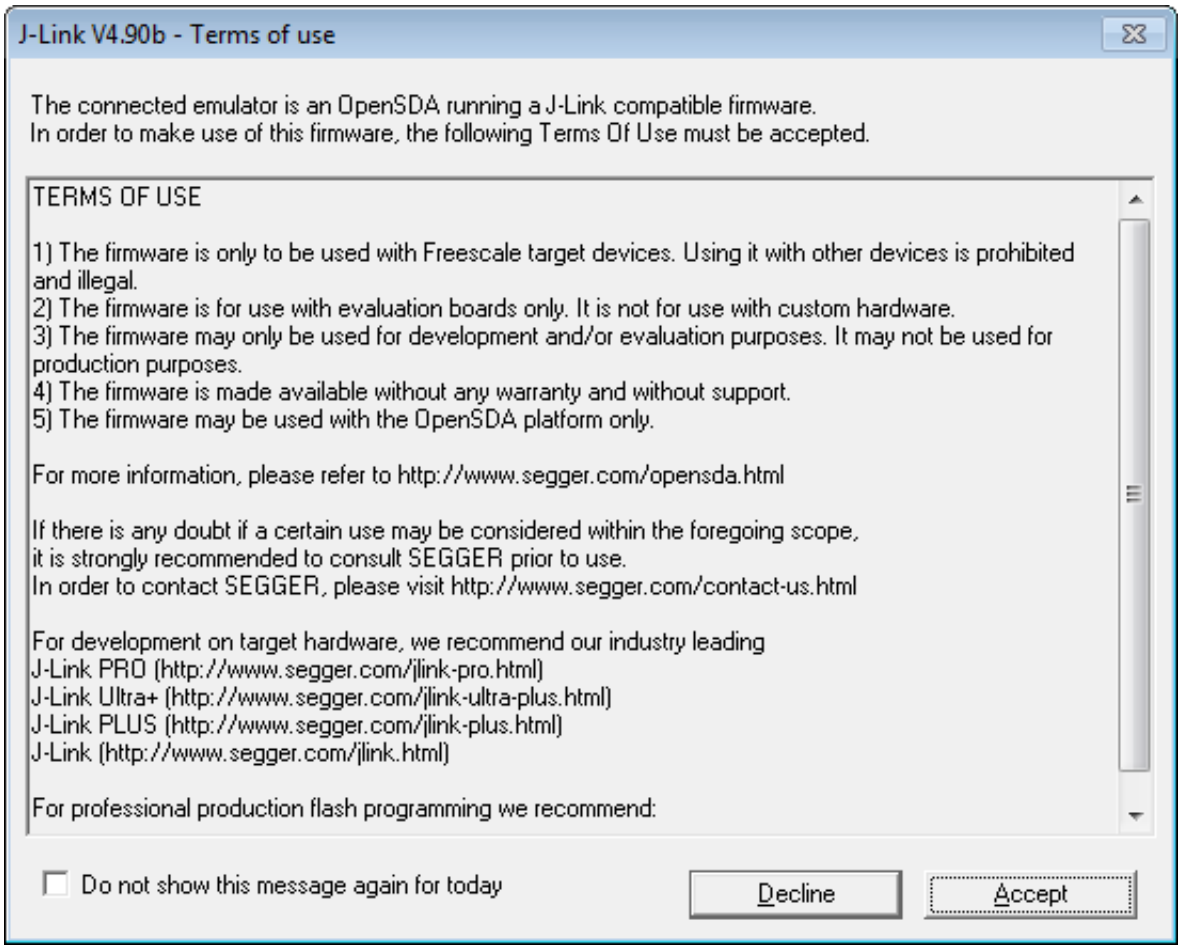


Figure A-9. Freescale Software Terms of Use

Set the Segger OpenSDA Debug configuration as in Figure A-10.

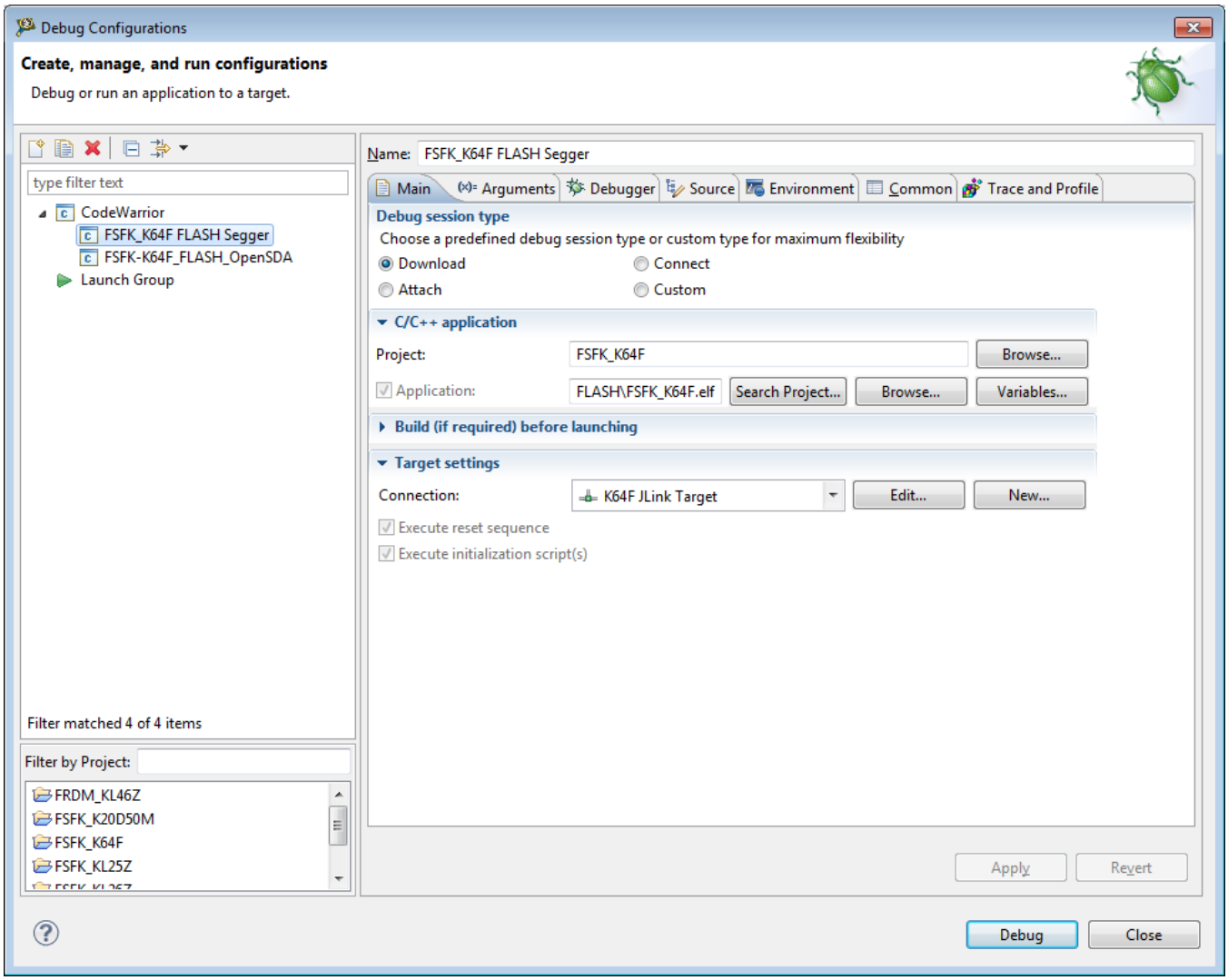


Figure A-10. Segger OpenSDA Debug Configuration

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number XSFL_Prod_UG
Revision 1.2, 1/2015

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

NXP:
FRDM-K64F