Document Number: FSFL_Prod_UG
Rev. 1.3, 9/2015

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.

Version 5.x of the development kit now includes:

- Full source code for the sensor fusion libraries
- New & improved 6 and 9-axis Kalman filters with visibly better convergence properties
- 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 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.

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, FRDM-K46Z and FRDM-K64F) software. Its use and distribution are controlled by the license agreement in the Software Licensing section.

1.1 Software Licensing

1.2 Software Features

The development kit is shipped in the form of a zip file. Simply unzip to the directory of your choice on your development PC. The kit includes a number of CodeWarrior and Kinetis Design Studio canned sensor fusion projects. Collectively, these cover Kinetis ARM Cortex M0+, M4 and M4F implementations. They are intended as reference projects for you to refer to when porting to other MCUs and boards.

Freescale Freedom platforms supported “out-of-the-box” include:

- FRDM-KL25Z
- FRDM-KL26Z
- FRDM-K20D50M
- FRDM-KL46Z and
- FRDM-K64F

The boards above can be mixed and matched with the following Freescale sensor “shields”:

- FRDM-FXS-MULTI (deprecated)
- FRDM-FXS-MULTI-B (deprecated)
- FRDM-FXS-9AXIS (deprecated)
- FRDM-FXS-MULT2-B
- FRDM-STBC-AGM01

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

- Fusion and magnetic calibration algorithms
- Programmable sensor sample rate (Default= 200 Hz)
- Programmable sensor fusion rate (Default = 25 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 standard algorithms
    - Accelerometer only (tilt)
    - Magnetometer only eCompass (vehicle)
    - Gyro only orientation (relative rotation)

    - Accelerometer plus magnetometer 6-axis eCompass
    - Accelerometer plus gyroscope orientation (gaming)
    - Accelerometer plus magnetometer and gyroscope (full 9-axis)
- 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
- Freescale Application Note AN5018, Rev. 1.0:, Basic Kalman Filter Theory
- Freescale Application Note AN5019, Rev. 1.0:: Magnetic Calibration Algorithms
- Freescale Application Note AN5020, Rev. 1.0: Determining Matrix Eigenvalues and Eigenvectors by Jacobi Algorithm
- Freescale Application Note AN5021, Rev. 1.0: Calculation of Orientation Matrices from Sensor Data
- Freescale Application Note AN5022, Rev. 1.0: Quaternion Algebra and Rotations
- Freescale Application Note AN5023, Rev. 1.0: Sensor Fusion Kalman Filters

1.4 Requirements

1.4.1 Kinetis Platform

The Sensor Fusion Library currently runs on the following 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

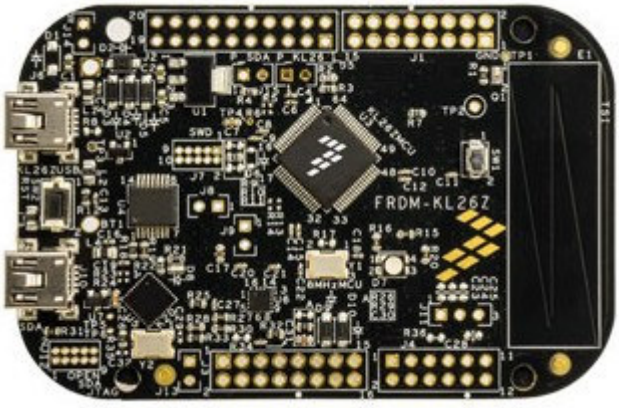

Figure 2. KL26Z Freedom Development Platform (FRDM-KL26Z)

- 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

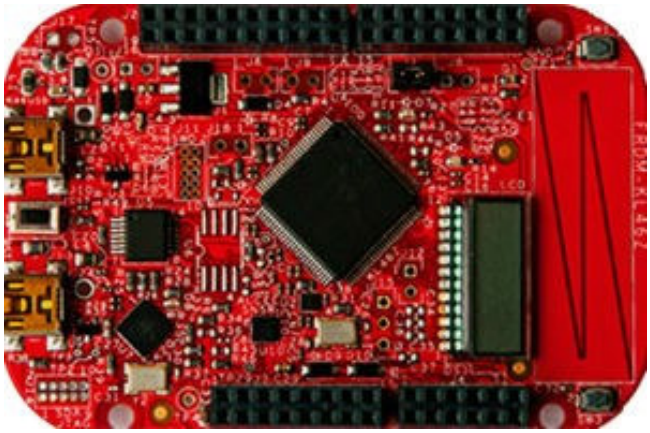

Figure 3. KL46Z Freedom Development Platform (FRDM-KL46Z)

- 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

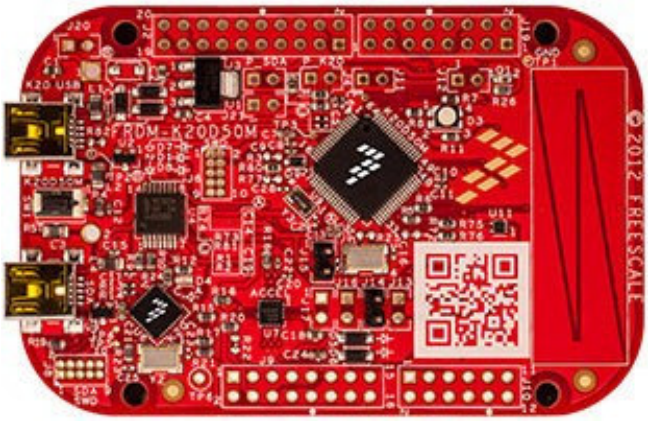

Figure 4. K20D50M Freedom Development Platform (FRDM-K20D50M)

- 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

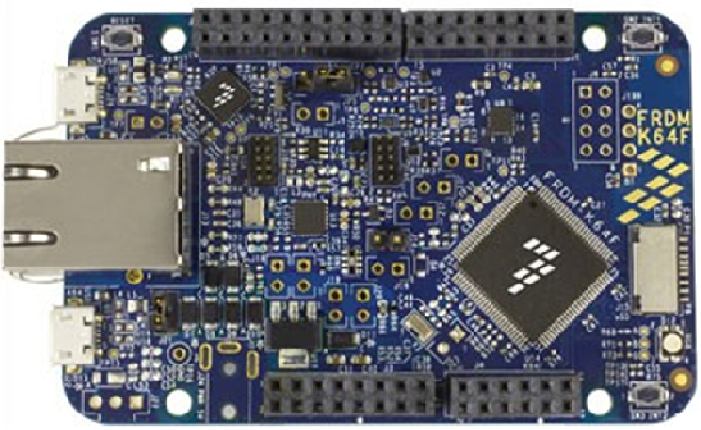

Figure 5. K64F Freedom Development Platform (FRDM-K64F)

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 a number of Freedom Sensor Development platforms compatible with the Freedom Development Kinetis Platforms discussed in Kinetis Platform. These are:

- FRDM-FXS-9-AXIS (deprecated), which contains an FXOS8700CQ 6-axis accelerometer/magnetometer combination sensor and an FXAS21000 gyroscope
- FRDM-FXS-MULTI (deprecated), which contains all features of the FRDM-FXS-9-AXIS plus additional sensors
- FRDM-FXS-MULTI-B (deprecated), which contains all features of the FRDM-FXS-MULTI plus Bluetooth Module and battery
- FRDM-STBC-AGM01, which replaces the FRDM-FXS-9-AXIS. It contains FXOS8700CQ 6-axis accelerometer/magnetometer combination sensor and an FXAS21002 gyroscope
- The FRDM-FXS-MULT2-B replaces the FRDM-FXS-MULTI and FRDM-FXS-MULTI-B. It uses the newer FXAS21002 gyroscope, but otherwise has the same sensor content.

The first three boards were built from the same PCB design and differ only in the number of parts on the platform. Those first three boards are no longer in production, but the Fusion Library works with any of the above boards. The FRDM-FXS-MULTI-B & FRDM-FXS-MULT2-B boards are the only ones that supports Bluetooth communications to the Freescale Sensor Fusion Toolbox for Android.

Key components of the FRDM-FXS-MULTI-B and FRDM-FXS-MULT2-B are identified in Figure 6. The two boards have essentially the same layout. The major difference being the new and improved FXAS21002 on the FRDM-FXS-MULT2-B.

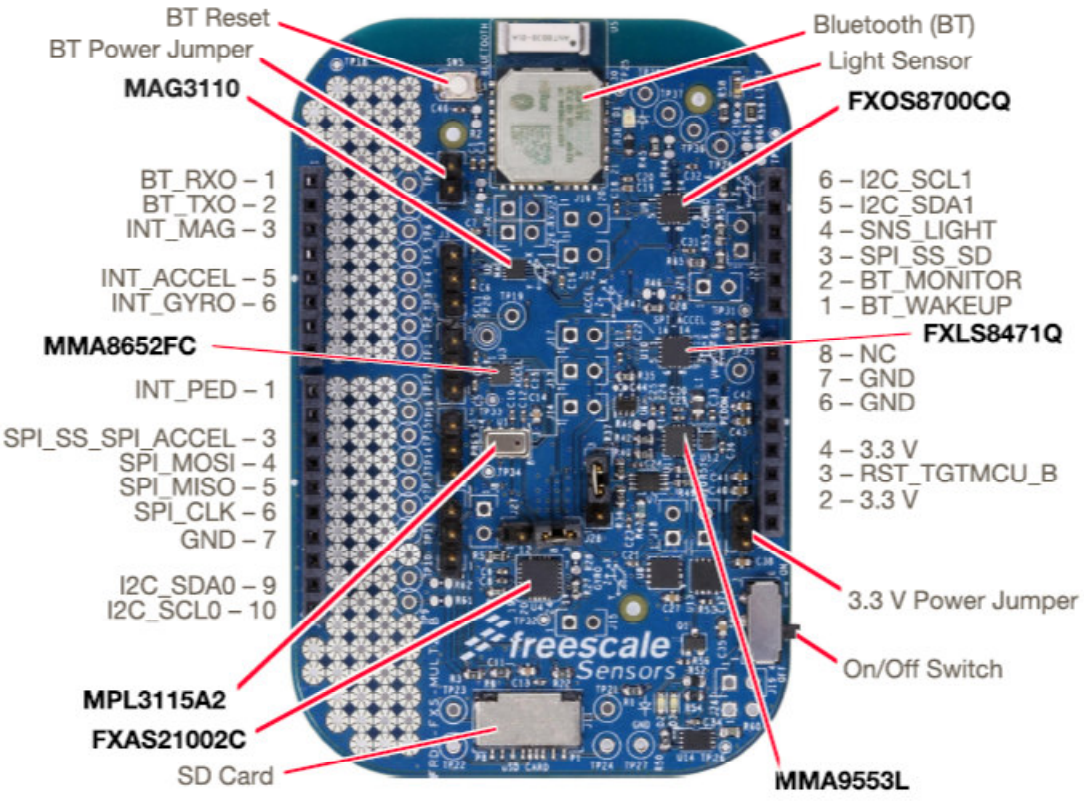

Figure 6. Freedom Development Platform Mult2-B with key components

NOTE: The user must remove the BT Power Jumper when using OpenSDA Communications Device Class [CDC] (UART/USB) to communicate with a PC when using a base board which uses one UART for both OpenSDA and to drive the Bluetooth interface. See Table 1 for details.

Figure 7 shows the FRDM-FXS-MULTI-B Sensor Development Platform plugging in to the FRDM-KL25Z Freedom Development Platform.

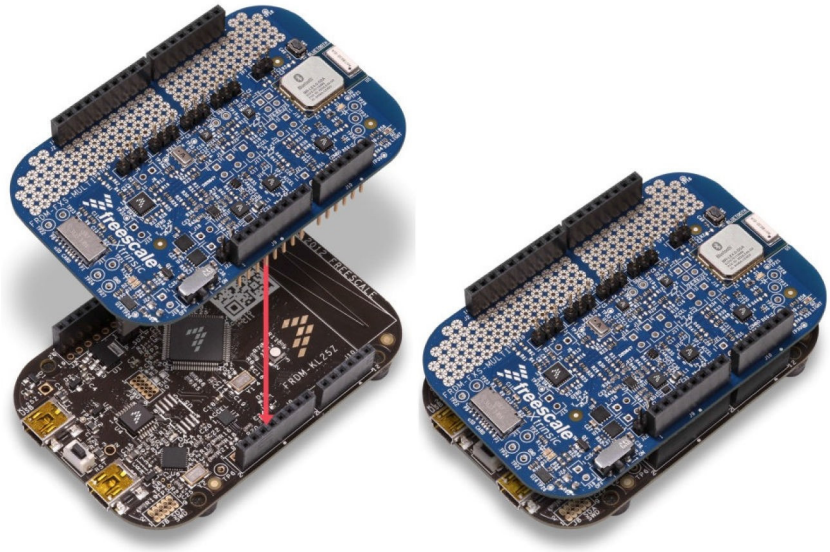

Figure 7. Freedom Development Platforms, FRDM-FXS-MULTI-B mated to FRDM-KL25Z

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.

The FRDM-STBC-AGM01 is shown in Figure 8. It is a much simpler board than those shown above, but still has all the components necessary to implement full 9-axis sensor fusion. It is compatible with all the base boards listed above.

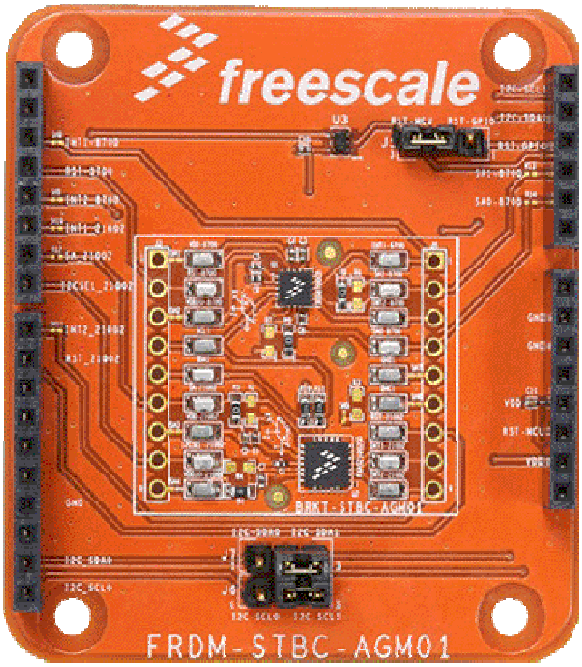

Figure 8. FRDM-STBC-AGM01 Freedom Development Platform

1.4.3 Peripherals Used by the Kit

Version 5.x of the product development software uses the virtual peripherals in Table 1. These are initialized via Processor Expert.

Changes from previous versions of the development kit:

- Prior versions reserved pins for blue LED and two test pins. These have been freed up in this release.
- Some Freedom boards use the same UART for the OpenSDA CDC[1] serial port communications and communicating with UART on the sensor shield. Others (FRDM-K20D50M and FRDM-K64F in the list below) utilized separate UARTS for those two functions. Version 4.22 of the sensor fusion library required the user to reprogram the UART in Processor Expert for those two boards, depending upon whether or not a wired or wireless link was desired. Version 5.0 of the kit includes two UARTS for all platforms. Either can be used to communicate with the reference applications. For FRDM-KL25Z/KL26Z/KL46Z, the second UART is essentially a dummy. But it's inclusion allows use of exactly the same code for all platforms, while enabling one binary per platform to support both wired and wireless communications on those boards with separate UARTs.

1 CDC = Communications Device Class

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		PTC3	PTB22	Red LED illuminated only during the period when a magnetic calibration is in progress.
LED_GREEN	BitIO_LDD	PTB19	PTE31	PTD5	PTD4	PTE26	Green LED flickers when the sensor fusion algorithms are executing.
FTM	TimerUnit_LDD	LPTMR0					Timer to drive the sensor read process at a typical rate of 200Hz.
UART_A	Serial_LDD	UART0 on PTA2:1 connects to both Bluetooth module and OpenSDA CDC			UART1 on PTE1:0 (BT)	UART3 on PTC17:16 (BT)	Serial port communications
UART_B		UART1 on PTE1:0 – “dummy” UART for code compatibility			UART0 on PTB17:16 used for CDC UART/USB communication		
I2C	I2C_LDD	I2C1 on PTC2:1			I2C0 on PTB1:0	I2C1 on PTC11:10	The I2C master bus communicating with the sensors.

1.4.4 **CodeWarrior**

CodeWarrior 10.6 or Kinetis Design Studio 3.0 (or above) must be used to build these applications. CodeWarrior can be downloaded from Freescale at http://www.freescale.com/codewarrior. Kinetis Design Studio can be downloaded from http://www.freescale.com/kds.

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 Installing the Development Kit

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

1. Download the zipfile containing the kit from www.freescale.com/sensorfusion. Expand the zip file using any standard expansion tool into the root directory of your choice. You will then see the directory structure shown in Figures 9 and 10. This version of the development kit has some *bonus material* in the form of bare board projects for tiltmeter and eCompass. By “bare board”, we mean “without RTOS”.

   The reference projects for sensor fusion will be under the *<toolname> FSFK* directories, and are the subject of this user guide.

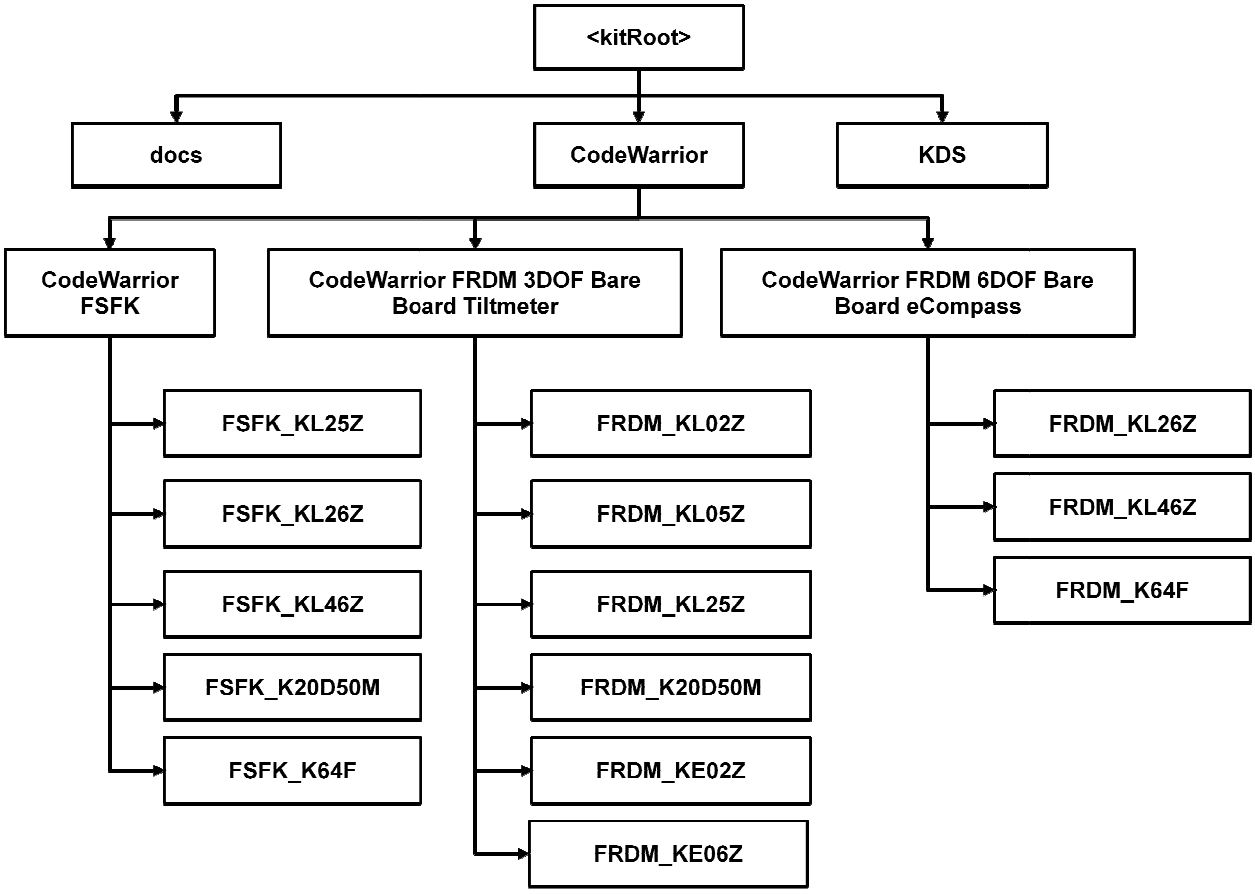

Figure 9. Kit structure with CodeWarrior subtree expanded

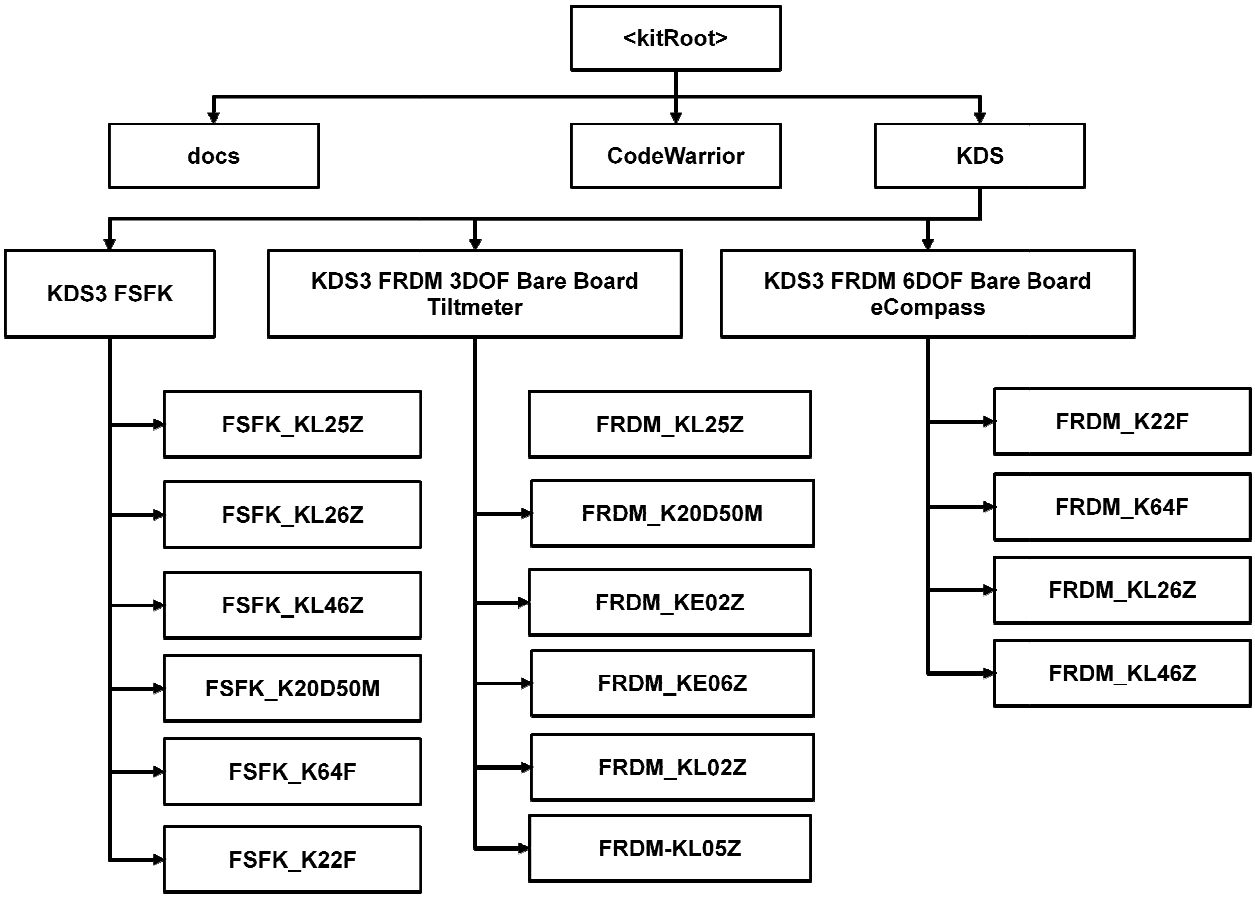

Figure 10. Kit structure with KDS subtree expanded

2. If not already done, Freescale recommends updating the OpenSDA bootloader for your Freedom Development Platform to the most recent release for that board. There are two generations of OpenSDA bootloaders in existence, with multiple vendor options supported for OpenSDA2.x. If you want to use the Freescale Sensor Fusion Toolbox for Windows GUI via a USB connection, you must select a variant that supports "virtual serial port" over USB. Alternately, if you wish to communicate via Bluetooth, then you must have the proper drivers for your Bluetooth module installed on your PC. In both cases, your board is then enumerated as a serial COM port by the toolkit.

   The OpenSDA2 ARM CMSIS-DAP[2] CDC drivers have been shown on other projects to offer more reliable serial communications to Windows PCs then the 1st generation OpenSDA. However CodeWarrior does not offer support for CMSIS-DAP, and other implementations may offer improved download speeds.

   At the time of writing, mbed implementations are available for FRDM-KL25Z/KL46Z/K20D50M and K64F. The FRDM-KL26Z does not have an mbed implementation available for download at this time. Check www.freescale.com/opensda to see if this has changed.

2 AKA "mbed"

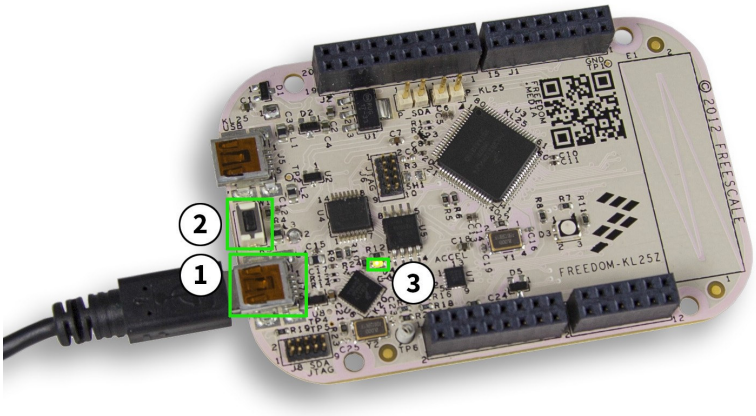

Figure 11. Freedom Development Platform KL-25Z

Procedure

The following procedure outlines the process for updating your board's OpenSDA. Refer to Figure as you perform step (c). The process is the same, regardless of which OpenSDA software you install.

a. Visit freescale.com/opensda, select your hardware platform via the pulldown menu located under the OpenSDA block diagram.
b. Download the most recent OpenSDA alternative of choice for your board.
c. 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 bootloader mode. Release the Reset button at this point.
d. Open Windows Explorer and locate the BOOTLOADER drive.
e. Drag the file you downloaded (either a .bin or .srec file) onto the BOOTLOADER.
f. 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. Inspect the contents of the *docs* directory. There you will find this document, a datasheet for the fusion library, and numerous other references.

2.2 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 installing zipfile contents for the kit, the file structure in the workspace should match Figure 9. Reference projects for sensor fusion are found under the *Codewarrior FSFK* directory. Each of these will contain a sources directory, where you should see:

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

```
fusion.h
magnetic.c
magnetic.h
main.c
matrix.c
matrix.h
mqx_tasks.c
mqx_tasks.h
orientation.c
orientation.h
types.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. The sources in the bare board bonus directories are also consistent within each of those two sets.

3. Open **CodeWarrior** and in the Workspace Launcher dialog, see Figure , choose the workspace where the product development project is located. For this example, we have located the kit at C:\FSFK.

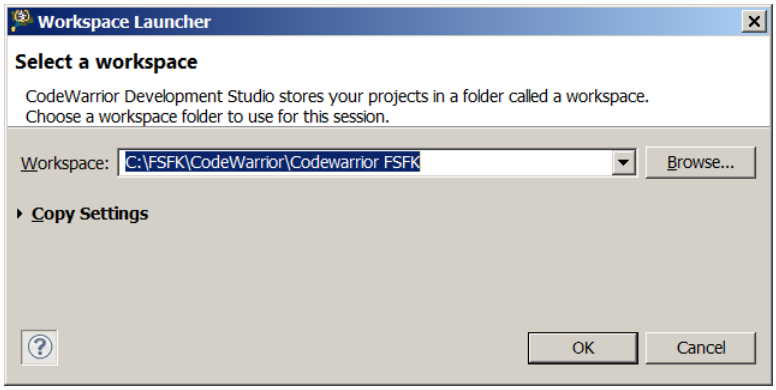

Figure 12. Workspace launcher select window

4. Import the project using File-> **Import** option, see Figure 8 select **General**->**Existing Projects into Workspace**

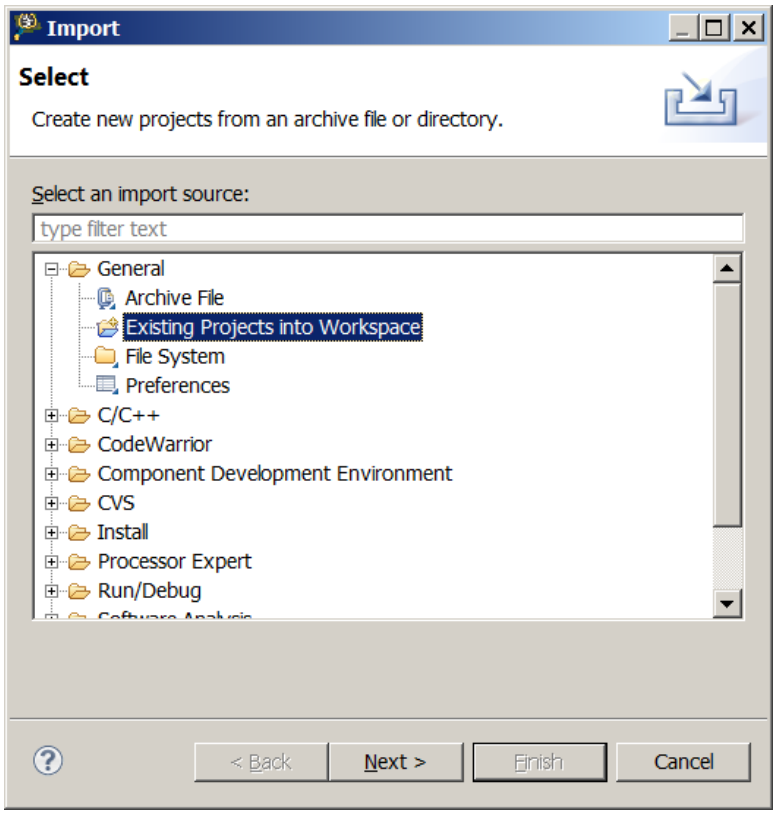

Figure 8. Import dialog window

5. Click **Next**. In the Import Window that appears, navigate to the folder containing the CodeWarrior FSFK directory, and select it. See Figure 9. Click “Finish”.

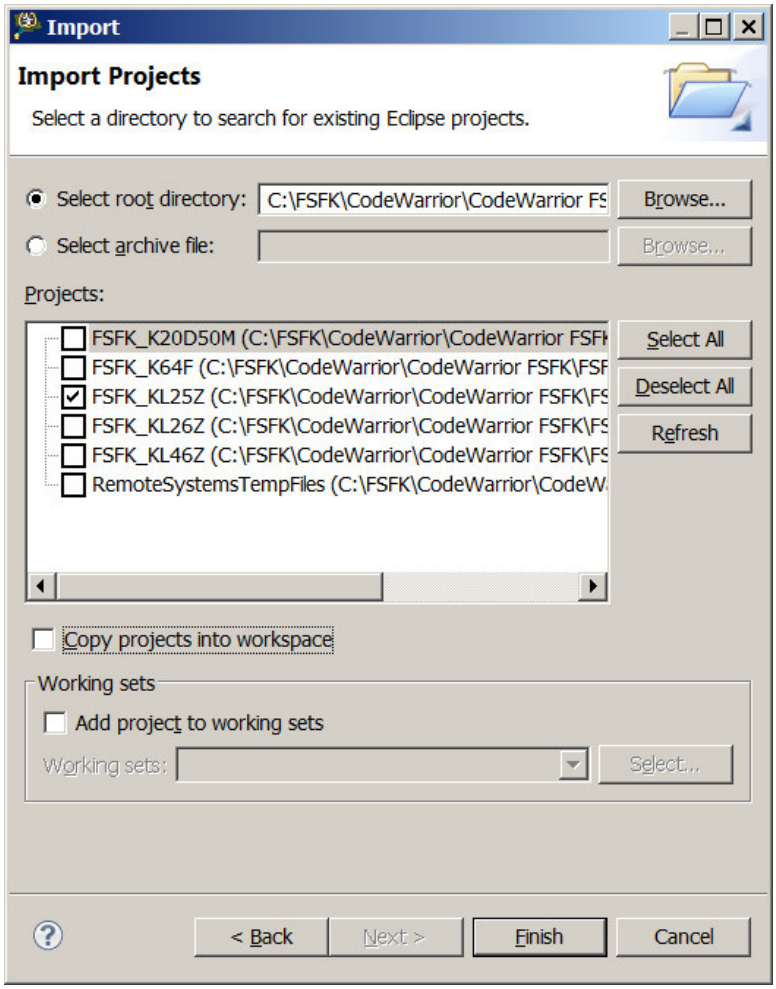

Figure 9. Import Projects window

6. In the project browser, double click on file name “ProcessorExpert.pe”. This will open the Processor Expert Components panel as shown in Figure 15.

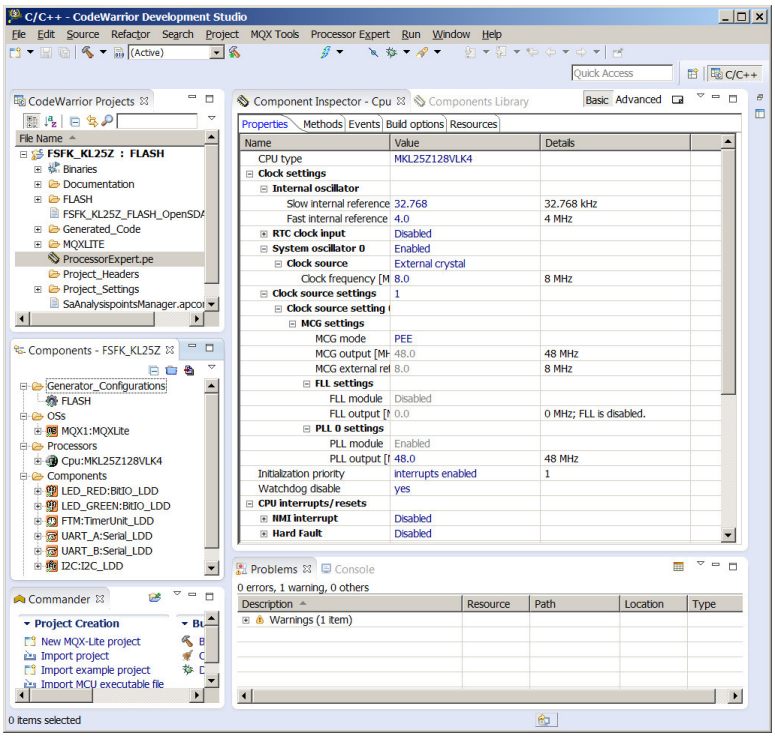

Figure 10. Imported Project with Components Window Expanded on Left

7. 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.
8. Select **Run**->**Run Configurations** to bring up the Run Configurations dialog, shown in Figure 11.

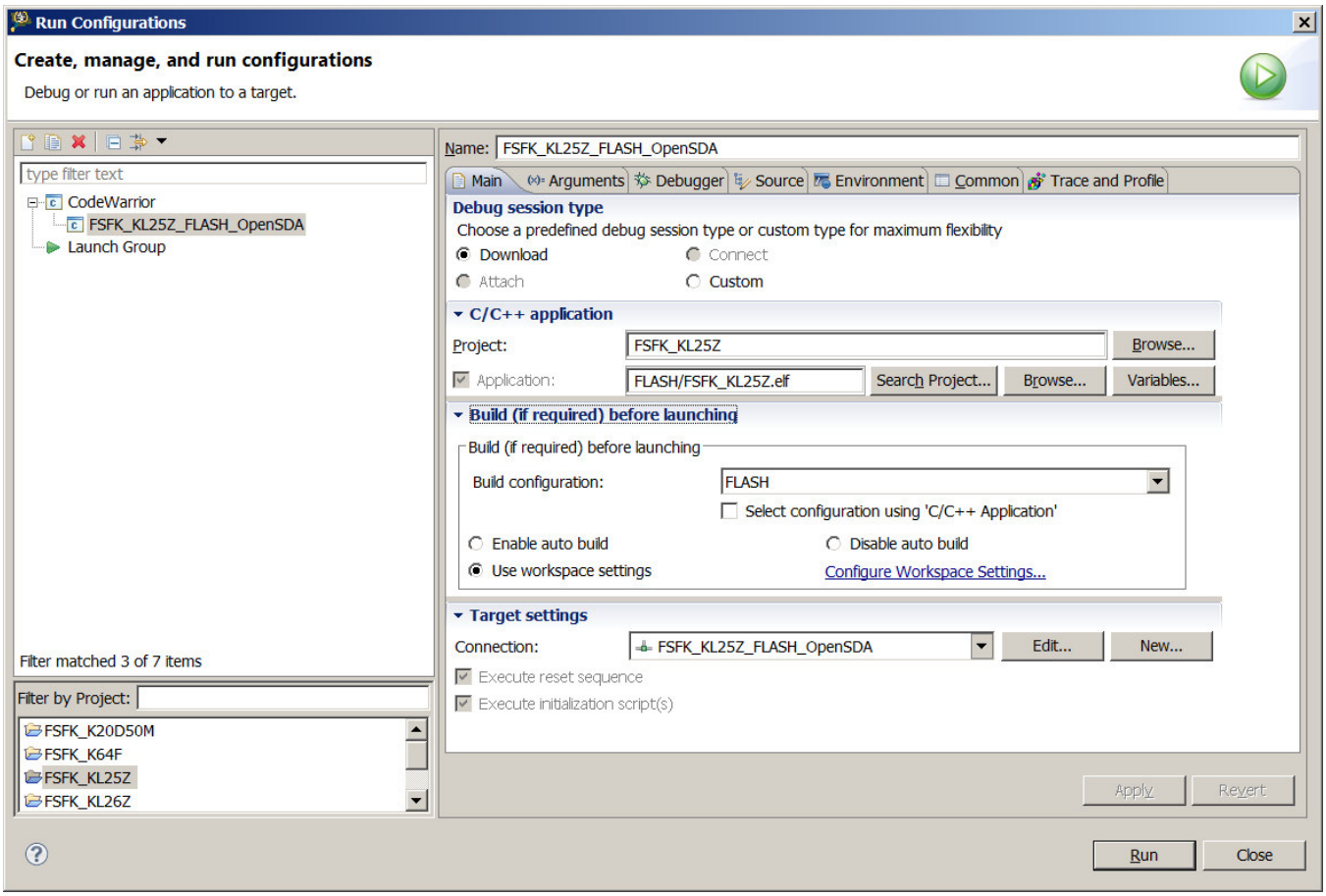

Figure 11. Run Configurations window

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 Easy Entry Points for Your Code

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.

11. Android Quick Start Guide.

2.3 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 Development with CodeWarrior will have the Kinetis Design Studio title displayed instead of CodeWarrior as was previously shown.

2.4 Limitations

Sensors supported using the development kit out-of-the box include FXOS8700CQ 6-axis accelerometer/magnetometer combination sensor and the FXAS21002 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.

2.5 Easy Entry Points for Your Code

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.

2.6 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. See 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 12.

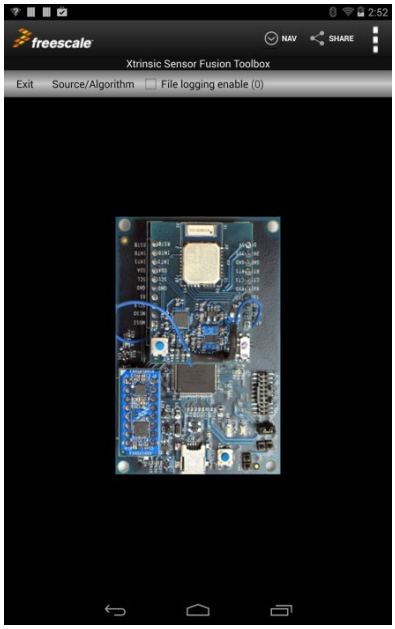

Figure 12. Sensor Fusion Toolbox

3. Open the **Preferences Screen** , see Figure 13 and check the **Automatically enable Bluetooth on entry** checkbox. Click **Save and Exit** option and then exit the application.

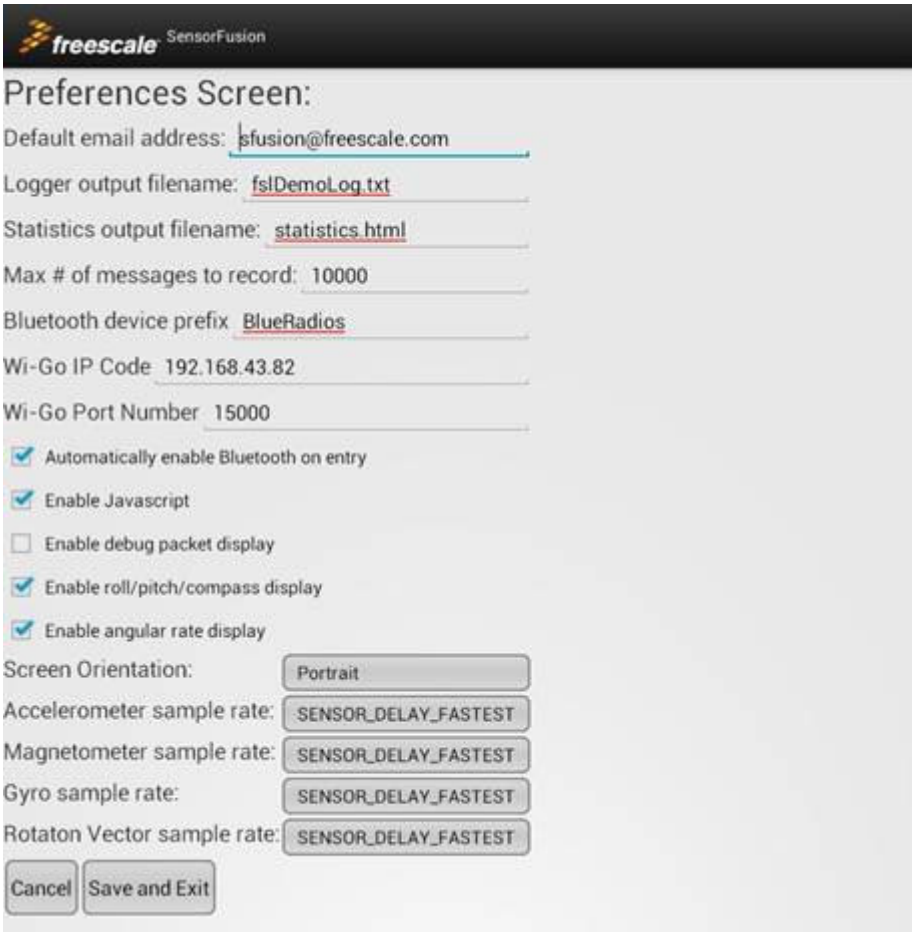

Figure 13. Sensor Fusion preferences window

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 that you pair to a single sensor fusion Freedom Development Platform at any given 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.

2.7 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. Attach Freedom/Sensor board stack-up via USB
5. File->Flash-><base board>-><shield board>-><sensor combination>
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. The high level architecture is shown in Figure 19 below.

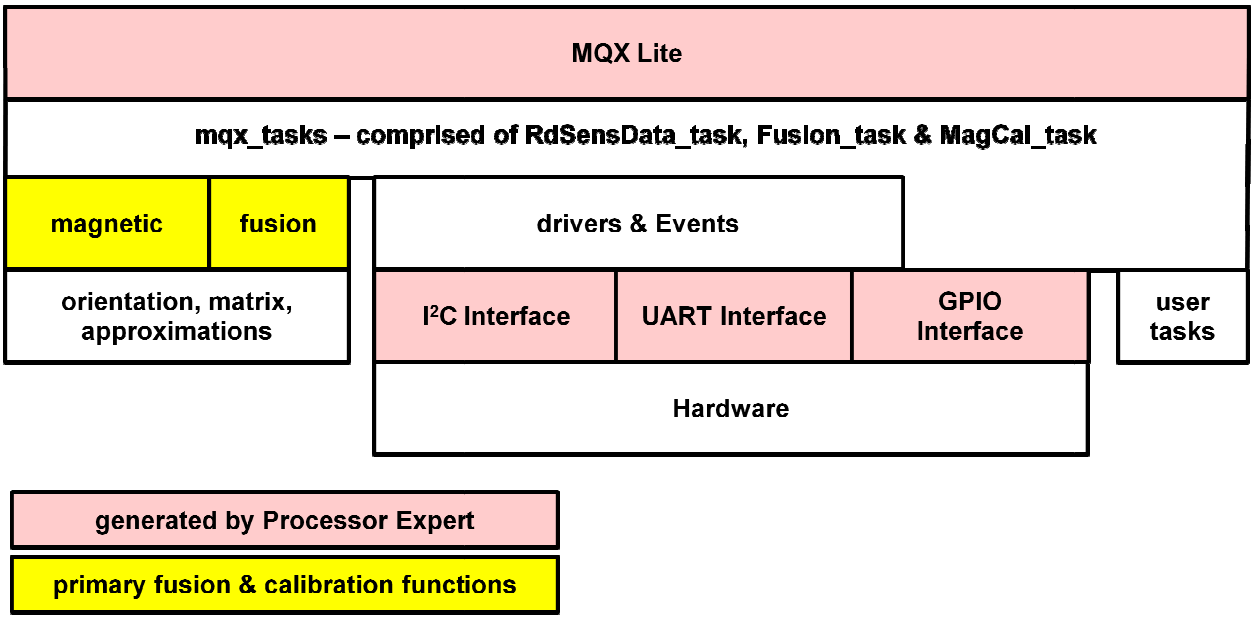

Figure 14. Function Hierarchy

3.1 Tasks Structure

Version 5.x of the fusion library has simplified the project structure. There are three main tasks managed by the RTOS. These are shown in Table 2, which lists the top level routines found in mqx_tasks.c.

Table 2. Top Level Sensor Fusion Routines

Sensor Fusion Routines	Description
`RdSensData_task()`	Performs sensor initialization and sampling (high frequency task)
`Fusion_task()`	Performs sensor fusion initialization and on-going fusion (medium frequency task)
`MagCal_task()`	Performances magnetic calibration in the background

For users who are planning to employ a different RTOS, the functions in this table make good interface points for that RTOS. 90% of the code in these functions is RTOS independent. The remainder, which does some event management, is easily identified and replaced. Look for the string "_lwevent".

The next few sub-sections provide additional detail for each of the three functions listed above.

3.1.1 RdSensData_task

The RdSensData_tasks is responsible for sensor initialization and sampling. It is the highest frequency task within the sensor fusion library.

3.1.1.1 RTOS and Timer Calls

By their nature, these functions are not portable to non-Kinetis architectures. The _lwevent and _task functions allow the fusion routines to coordinate tasks with the MQXLite RTOS. The FTM_SetPeriodTicks function is used to control the sensor sampling frequency. All of these functions were created via Processor Expert.

- _task_create_at
- _lwevent_create
- _lwevent_wait_for
- _lwevent_set
- FTM_SetPeriodTicks

3.1.1.2 GUI

LED_RED_ClrVal, LED_RED_SetVal, LED_GREEN_SetVal are generated by Processor Expert, and provide low level GPIO functions to control LED colors on the development hardware.

The LED is RED during sensor initialization.

LED operation is strictly for user feedback, and they can be eliminated with no effect on fusion operation.

3.1.1.3 Sensor Drivers

Each sensor type has an initialization and at least one ReadData function. These are all defined in drivers.c. These functions are hardware specific, and will need to be replaced if different hardware is used.

- MPL3115_Init / MPL3115_ReadData
- FXOS8700_Init / FXOS8700_ReadAccData / FXOS8700_ReadMagData
- FXAS2100X_Init / FXAS2100X_ReadData
- MMA8652_Init / MMA8652_ReadData
- FXLS8952_Init / FXLS8952_ReadData
- MAG3110_Init / MAG3110_ReadData

The drivers above are build on top of:

- WriteI2CByte
- ReadI2CBytes

Both of which are also defined in drivers.c.

3.1.1.4 User Tasks

user_tasks.c provides convenient entry points for developers to add their own code without having to modify the more involved mqx_tasks functions. This makes it easier to integrate future library upgrades from Freescale.

- UserStartup
- UserHighFrequencyTaskInit
- UserHighFrequencyTaskRun

UARTs are initialized via the UserStartup function. Most of the UART operation is encapsulated within the user functions, as it is viewed as being application specific. Events.c contains UART callbacks and command interpreter.

3.1.1.5 Magnetics

Both of the functions below are defined in magnetic.c.

- iUpdateMagnetometerBuffer updates the magnetic measurement buffer with most recent magnetic data (typically 200Hz).
- fInvertMagCal maps uncalibrated magnetometer data onto calibrated averaged data

3.1.2 Fusion_task calls

3.1.2.1 RTOS Calls

The _lwevent functions allow the fusion routines to coordinate tasks with the MQXLite RTOS.

- _lwevent_wait_for
- _lwevent_set

3.1.2.2 User Tasks

Both of these functions are defined in user_tasks.c.

- UserMediumFrequencyTaskInit
- UserMediumFrequencyTaskRun calls CreateAndSendPacketsViaUART (defined in drivers.c), which sends data from development board to Windows or Android via UART.

3.1.2.3 GUI

The Green LED is flashed to indicate that the fusion routines are running.

- LED_GREEN_NegVal

LED operation is strictly for user feedback, and they can be eliminated with no effect on fusion operation.

3.1.2.4 Fusion Functions

The standard sensor fusion application can run any combination of 6 algorithms plus pressure measurement based upon compile time constants set in build.h.

There's a standard bit of code wrapped around each of the fusion calls that is used to measure execution time using the systick counter in the ARM core. systick counts are automatically sent to the Sensor Fusion Toolbox (both Windows and Android), enabling an easy evaluation of algorithm computation costs.

- fInitFusion – forces a reset of all algorithms the next time they run
- fRun_1DOF_P_BASIC – 1-axis pressure measurement (low pass filtered)
- fRun_3DOF_G_BASIC – 3-axis accelerometer-based tilt computation
- fRun_3DOF_B_BASIC – 2-axis vehicle compass algorithm
- fRun_3DOF_Y_BASIC – 3-axis gyro integration algorithm
- fRun_6DOF_GB_BASIC – 6-axis tilt-compensated eCompass algorithm
- fRun_6DOF_GY_KALMAN – 6-axis accel/gyro Kalman filter for gaming
- fRun_9DOF_GBY_KALMAN – 9-axis Kalman filter

All of the functions listed above are defined in file fusion.c.

3.1.3 MagCal_task calls

3.1.3.1 RTOS Calls

The _lwevent functions allow the fusion routines to coordinate tasks with the MQXLite RTOS. In this case, to wait for a magnetic calibration event before running calibration.

- _lwevent_wait_for
- LED_RED_SetVal / LED_RED_ClrVal - The LED is flashed red during magnetic calibration. Again, LED calls can be removed without affecting fusion results.
- fInitMagCalibration - initialize magnetic calibration and magnetometer data buffer
- fRunMagCalibration – run magnetic calibration

3.2 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_1DOF_P_BASIC     // 1DOF pressure and temperature
#define COMPUTE_3DOF_G_BASIC     // 3-axis accel tilt
#define COMPUTE_3DOF_B_BASIC     // 3DOF mag eCompass (vehicle): (mag)
#define COMPUTE_3DOF_Y_BASIC     // 3DOF gyro integration: (gyro)
#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 default build has all 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 Table 3.

Table 3 Global Structures

Pointer Function	Structure Pointer (mqx_tasks.c)	Structure Type (types.h)
Altitude / Temperature	thisPressure	PressureSensor
Accelerometer	thisAccel	AccelSensor
Magnetometer	thisMag	MagSensor
Gyroscope	thisGyro	GyroSensor
Altitude / Temperature	thisSV_1DOF_P_BASIC	SV_1DOF_P_BASIC
2-axis automotive compass	thisSV_3DOF_B_BASIC	SV_3DOF_B_BASIC
3-axis gyro integration	thisSV_3DOF_Y_BASIC	SV_3DOF_Y_BASIC
3-axis tilt results	thisSV_3DOF_G_BASIC	SV_3DOF_G_BASIC
eCompass results	thisSV_6DOF_GB_BASI	SB_6DOF_GB_BASIC
accel+gyro results	thisSV_6DOF_GY_KALM	SV_6DOF_GY_KALMAN
9-axis results	thisSV_9DOF_GBY_KAL	SV_9DOF_GBY_KALMAN

3.2.1 **Writing Sensor Values into Global Structures**

Sensor drivers are designed to take two parameters:

- a pointer to the Processor Expert Logical Device Driver (LDD) structure associated with the serial port to which the sensor is attached. For the current library, this is I2C_DeviceData.
- a pointer to a generic structure for that class of sensor. For the current implementation, this is one of:
  - thisPressure
  - thisAccel
  - thisMag
  - thisGyro

Each driver is responsible for transforming (if necessary) sensor data into the THISCOORDSYSTEM frame of reference defined in build.h. drivers.c contains utility functions to assist in this effort. Drivers are also responsible for filling the fields of the generic sensor structure. The sensor fusion and magnetic calibration functions will read the data directly from these structures.

3.3 Reading Sensor Values

NOTE: CHX, CHY and CHZ are conveniently defined as 0, 1 and 2 for readability in the library. Sensor data types are defined in types.h.

3.3.1 **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.iGs[CHX]*thisAccel.fgPerCount;
fAcc[Y] = (float) thisAccel.iGs[CHY]*thisAccel.fgPerCount;
fAcc[Z] = (float) thisAccel.iGs[CHZ]*thisAccel.fgPerCount;
```

Alternately, simply read the averaged measurements, which are already stored as floats:

```
fAcc[X] = thisAccel.fGsAvg[CHX];
fAcc[Y] = thisAccel.fGsAvg[CHY];
fAcc[Z] = thisAccel.fGsAvg[CHZ];
```

3.4.1 Gyroscope

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

```
float fAngularRate[3]; // gyro sensor output
fAngularRate[X] = (float) thisGyro.iYs[CHX]*thisGyro.fDegPerSecPerCount;
fAngularRate[Y] = (float) thisGyro.iYs[CHY]*thisGyro.fDegPerSecPerCount;
fAngularRate[Z] = (float) thisGyro.iYs[CHZ]*thisGyro.fDegPerSecPerCount;
```

3.3.2 **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 fBr[3]; // mag - raw average over OVERSAMPLE_RATIO measurements
float fBc[3]; // mag - iBp after calibration
fBr[X] = (float) thisMag.iBsAvg[X]*thisMag.fCountsPeruT;
fBr[Y] = (float) thisMag.iBsAvg[Y]*thisMag.fCountsPeruT;
fBr[Z] = (float) thisMag.iBsAvg[Z]*thisMag.fCountsPeruT;
fBc[X] = (float) thisMag.iBcAvg[X]*thisMag.fCountsPeruT;
fBc[Y] = (float) thisMag.iBcAvg[Y]*thisMag.fCountsPeruT;
fBc[Z] = (float) thisMag.iBcAvg[Z]*thisMag.fCountsPeruT;
```

Alternately, you can read values directly in floating point:

```
fBr[X] = thisMag.fBsAvg[X];
fBr[Y] = thisMag.fBsAvg[Y];
fBr[Z] = thisMag.fBsAvg[Z];
fBc[X] = thisMag.fBcAvg[X];
fBc[Y] = thisMag.fBcAvg[Y];
fBc[Z] = thisMag.fBcAvg[Z];
```

3.4 Reading Fusion Results

Global Data Structure variables

Each of the various fusion options have similar output structures. Index directly to the data you need using the pointer names in Table 3.

Table 4 Fusion Algorithm Options correspond to the following:

- G = SV_3DOF_G_BASIC (accel only)
- B = SV_3DOF_B_BASIC (2-axis auto compass)
- Y = SV_3DOF_B_BASIC (gyro integration)
- GB = SV_6DOF_GB_BASIC (accel + mag eCompass)
- GY = accel + gyro Kalman
- GBY = 9-axis Kalman

Table 4 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 and Chi
- f prefix = floating point variable
- LP = low pass filtered
- Pl suffix = post apriori estimate from Kalman filter
- Gl suffix = global frame
- R = rotation / orientation
- Se = sensor frame

Table 4. Location of individual variables within the global structures

Description	data type			Fusion Algorithm Options			
		G (accel)	B (auto compass)	Y (gyro)	GB (eCompass)	GY (accel + gyro)	GBY (9-axis)
roll in degrees	float	fLPPhi	fLPPhi	fPhi	fLPPhi	fPhiPl	fPhiPl
pitch in degrees	float	fLPThe	fLPThe	fThe	fLPThe	fThePl	fThePl
yaw in degrees	float	fLPPsi	fLPPsi	fPsi	fLPPsi	fPsiPl	fPsiPl
compass heading in degrees	float	fLPRho	fLPRho	fRho	fLPRho	fRhoPl	fRhoPl
tilt angle in degrees	float	fLPChi	fLPChi	fChi	fLPChi	fChiPl	fChiPl
magnetic inclination angle in degrees	float	N/A	N/A	N/A	fDelta, fLPDelta	N/A	fDeltaPl
gyro offset in degrees/sec	float	N/A	N/A	N/A	N/A	fbPL[3]	fbPL[3]
linear acceleration in the global frame in gravities	float	N/A	N/A	N/A	N/A	fAccGl[3]	fAccGl[3]
quaternion (unitless)	fquaternion	fLPq, fq	fLPq, fq	fq	fLPq, fq	fqPl	fqPl
angular velocity in dps	float	fOmega[3][1]	fOmega[3][1]	fOmega[3]	fOmega[3]	fOmega[3][2]	fOmega[3][2]
orientation matrix (unitless)	float	fR[3][3] fLPR[3][3]	fR[3][3] fLPR[3][3]	fR[3][3]	fR[3][3] fLPR[3][3]	fRPl[3][3]	fRPl[3][3]
rotation vector	float	fLPRVec[3]	fLPRVec[3]	fRVec	fLPRVec[3]	fRVecPl[3]	fRVecPl[3]
time interval in seconds	float	fdeltat	fdeltat	fdeltat	fDeltat	fKalmandeltat	fKalmandeltat

1. Yaw is not supported for 3-axis accelerometer. Only yaw is supported for the 2D automotive compass.
2. Physical gyro angular rate corrected to subtract computed offset.

Example: 3.5.1 Reading Quaternion Values

The Freescale library uses the convention that $q0 = \cos(\alpha/2)$ 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_3DOF_B_BASIC.fLPq  // OR
//fq = thisSV_3DOF_Y_BASIC.fq    // 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 accel only 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 5. 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	This task has the highest priority.
sensor 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	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.

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

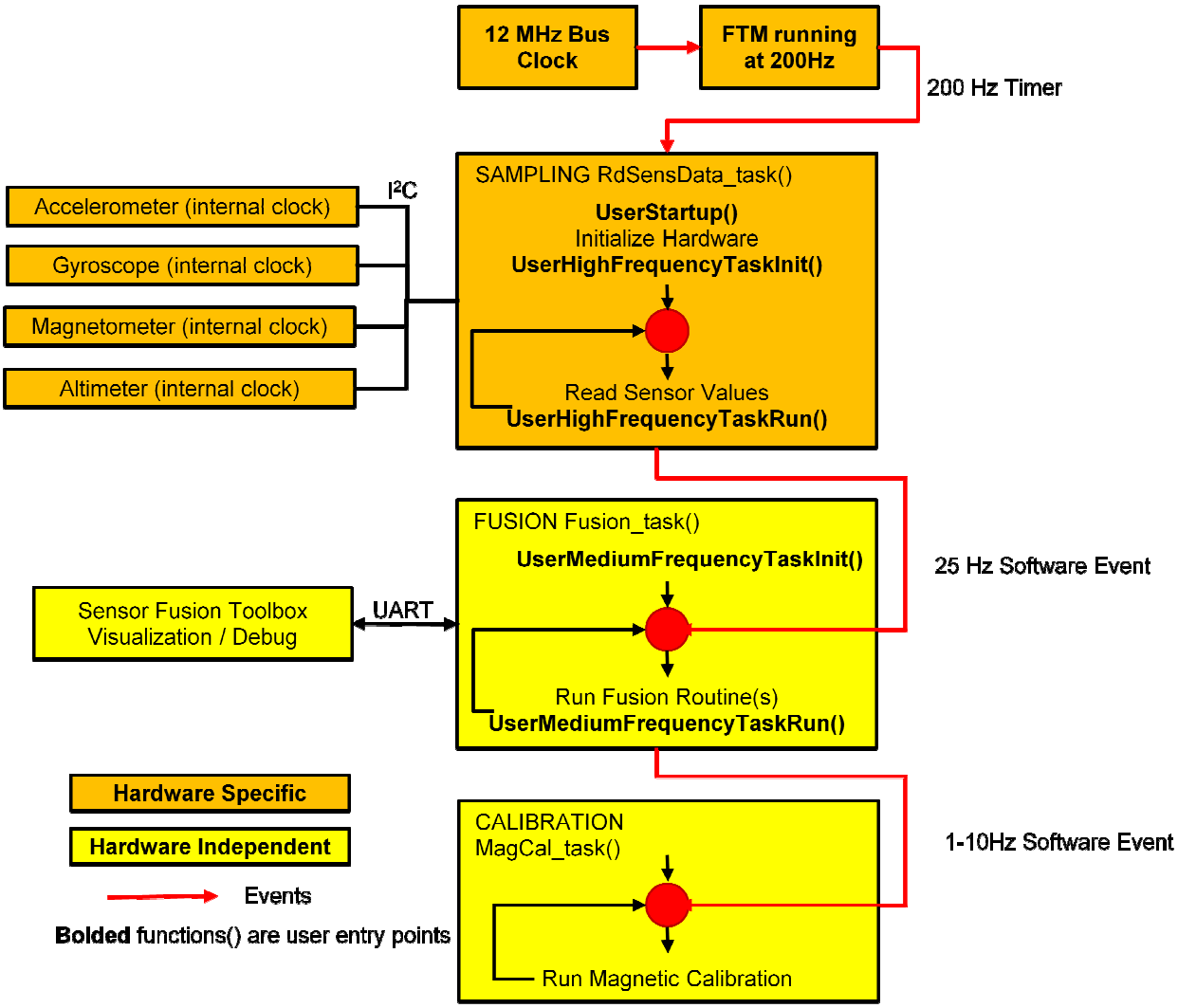

Figure 15. Software Task Scheduling Diagram

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

4.2 Timing Diagram

Figure 16 shows the relative timing and priorities of the sensors and the three software tasks.

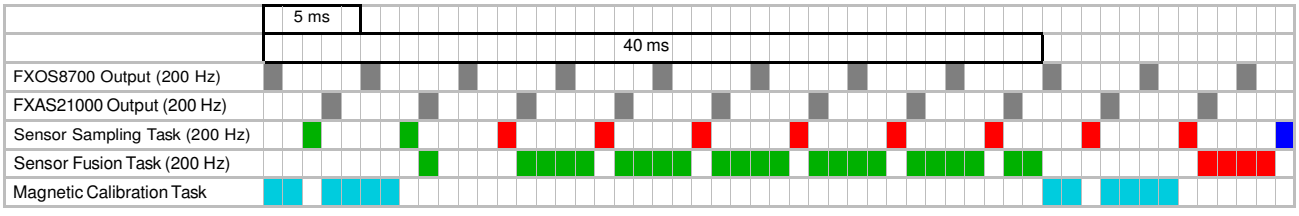

Figure 16. Software Task Timing Diagram

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 16 are serial communications from the embedded MCU and an external controller and/or GUI. In the case of the supplied project templates, this is comprised of UART communications to an external Bluetooth module or Windows client via virtual serial port. 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.

```
// Copyright (c) 2014, 2015, Freescale Semiconductor, Inc.
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are met:
//     * Redistributions of source code must retain the above copyright
//       notice, this list of conditions and the following disclaimer.
//     * Redistributions in binary form must reproduce the above copyright
//       notice, this list of conditions and the following disclaimer in the
//       documentation and/or other materials provided with the distribution.
//     * Neither the name of Freescale Semiconductor, Inc. nor the
//       names of its contributors may be used to endorse or promote products
//       derived from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
// ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL FREESCALE SEMICONDUCTOR, INC. BE LIABLE FOR ANY
// DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
// LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
// ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// If you want to take a simplistic approach of using the existing templates
// as your starting point, simply adding your code for startup, sampling task (eg. 200Hz)
// and fusion tasks (eg. 25Hz), then this is the file for you.  Just put your code
// where ever you see "PUT YOUR CODE HERE"
//
#include "Cpu.h"
#include "UART_A.h"
#include "UART_B.h"

#include "mqx_tasks.h"
#include "Events.h"
#include "drivers.h"
#include "UART_A.h"

// global structures
uint8 sUARTOutputBuffer[256];  // larger than the nominal 124 byte size for outgoing packets
uint8 sUART_A_InputBuffer[32]; // larger than nominal 1 byte command in case of buffering
uint8 sUART_B_InputBuffer[32]; // larger than nominal 1 byte command in case of buffering

void UserStartup(void)
{
   // initialize the BlueRadios Bluetooth module
   BlueRadios_Init(UART_A_DeviceData);

   // trigger a callback when any single character is received into
   // the UART_A or UART_B buffers.
   UART_A_ReceiveBlock(UART_A_DeviceData, sUART_A_InputBuffer, 1); // Bluetooth on shield
   UART_B_ReceiveBlock(UART_B_DeviceData, sUART_B_InputBuffer, 1); // OpenSDA and USB

   // UART callbacks and command interpreter are located in Events.c

   // 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)
{
    static int32 iThrottle = 0;

    // This function is called after the Kalman filter loop at a rate of
    // SENSORFS / OVERSAMPLE_RATIO Hz. With the default settings this is
    // 200Hz/8=25Hz giving a smooth video quality display on the PC and
    // Android user interfaces. The UART (serial over USB and over Bluetooth)
    // is limited to 115kbps which is more than adequate for the 31kbps
    // needed at the default 25Hz output rate but insufficient for 100Hz or
    // 200Hz output rates.  There is little point is providing output data
    // faster than 25Hz video rates but since the UARTs can
    // support a higher rate, the limit is set to MAXPACKETRATEHZ=40Hz.

    // check for any need to throttle the output rate
#define MAXPACKETRATEHZ 40
#define RATERESOLUTION 1000
    if (((int32)MAXPACKETRATEHZ * (int32)OVERSAMPLE_RATIO) >= (int32)SENSORFS)
    {
        // no UART bandwidth problem: transmit the packets over UART (USB and Bluetooth)
        CreateAndSendPacketsViaUART(UART_A_DeviceData, UART_B_DeviceData);
    }
    else
    {
        // throttle back by fractional multiplier
        //    (OVERSAMPLE_RATIO * MAXPACKETRATEHZ) / SENSORFS
        // the increment applied to iThrottle is in the range 0 to (RATERESOLUTION - 1)
        iThrottle += ((int32)OVERSAMPLE_RATIO * (int32) MAXPACKETRATEHZ *
               (int32)RATERESOLUTION) / SENSORFS;
        if (iThrottle >= RATERESOLUTION)
        {
            // update the throttle counter and transmit the packets over
            // UART (USB and Bluetooth)
            iThrottle -= RATERESOLUTION;
            CreateAndSendPacketsViaUART(UART_A_DeviceData, UART_B_DeviceData);
        }
    }

    //
    // PUT YOUR CODE HERE
    //

    return;
}
```

4.4 UART Communications

`CreateAndSendPacketsViaUART` is responsible for sending Bluetooth packets to the Fusion Toolbox. Variable array `sUARTOutputBuffer[]` contains information to be transmitted to the toolbox. The

code snippet below, which may be found in drivers.c, illustrates how a DEBUG packet (packet type 0x02) can easily be constructed.

```
if (globals.DebugPacketOn)
   {
      // [0]: packet start byte
      sUARTOutputBuffer[iIndex++] = 0x7E;

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

      // [2]: packet number byte
      sBufAppendItem(sUARTOutputBuffer, &iIndex, &iPacketNumber, 1);
      iPacketNumber++;

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

      // [6-5] systick count / 20
      sBufAppendItem(sUARTOutputBuffer, &iIndex, (uint8*)&isystick, 2);

      // [7 in practice but can be variable]: add the tail byte for the debug packet type 2
      sUARTOutputBuffer[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 MQXLITE. 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 Section 4.1. 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.

Choice of peripherals and peripheral pins 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 6 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 6. Source files which may need to be modified

Files	Description
Events.c *Events.h*	Callback functions for hardware events. Also includes the UART command interpreter.
drivers.c *drivers.h*	Initialization of hardware timers and I2C drivers for sensors. Also includes UART output routines (which are called by user_tasks.c)
mqx_tasks.c *mqx_tasks.h*	Creates and runs the Sampling, Fusion and Calibration tasks which call functions in tasks.c.
user_tasks.c *user_tasks.h*	Application specific entry points. They also encapsulate much of the UART communication.
build.h	Build options consolidated into a single file.
main.c	Initializes and executes MQX.

5.1 File-by-File Changes

5.1.1 Events.c

This file contains event handlers for:

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

UART communications include functions `UART_A_OnBlockReceived()` and `UART_B_OnBlockReceived()`. These implement decoding of incoming Bluetooth packets sent by the Freescale Sensor Fusion Toolbox. The user can replace that code with whatever communications is appropriate for their application.

5.1.2 drivers.c

This file is highly hardware-dependent. It contains:

1. Timer utility functions
2. Sensor initialization and sampling functions
3. Hardware Abstraction Layer (HAL) mapping functions
4. UART utility functions
5. `CreateAndSendPacketsViaUART()`, 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, *and they are oriented the same way as the reference PCBs*, they will probably not need to make any changes to items 1–4 above. If you use a different communications mechanism or packet structure, item 5 above will need to be replaced.]

If your PCB does have a different layout of the accelerometer, magnetometer and gyro sensors, you may need to modify the HAL (Hardware Abstraction Layer) functions in drivers.c. These are:

- `ApplyAccelHAL()`
- `ApplyMagHAL()`

- `ApplyGyroHAL()`

The purpose of these functions is to transform sensor readings from sensor frame, to `THISCOORDSYSTEM` (as defined in build.h). If your physical sensor package orientations differ from those shown in Figure 6, then you will need to account for the change in the HAL functions.

5.1.3 **mqx_tasks.c**

This file contains:

- `RdSensData_task()`: High frequency sensor sampling
- `Fusion_task()`: Top level control loop for sensor fusion
- `MagCal_task()`: Low frequency task for magnetic calibration

If the user's project needs additional MQX-Lite tasks for their project, they should be added here. matching Processor Expert tasks should be created under the MQX1 object in the Processor Expert Components pane.

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.

5.1.4 **user_tasks.c**

This file has been defined as an easy place for you to **`PUT YOUR CODE HERE`**. It also contains initialization for UART communications and calls to stream data to the Sensor Fusion Toolbox via those UARTS. If you don't need those UART communications, be sure to delete the code here, as well as additional functions in Events.c.

5.1.5 **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_KINETIS should then be set to the identifier of the new 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.

```
// Kinetis processor base boards:
// 5 bit code 0 to 31 inclusive transmitted in bits 4-0
#define RESERVED0          0
#define FRDM_KL25Z         1
#define FRDM_K20D50M       2
#define SPARE0             3
#define FRDM_KL26Z         4
#define FRDM_K64F          5
#define SPARE1             6
#define FRDM_KL46Z         7
#define SPARE2             8
#define FRDM_K22F          9
#define SPARE3             10
#define FRDM_KL05Z         11
#define SPARE4             12
#define FRDM_KL02Z         13
```

```
#define FRDM_KE02Z          14
#define FRDM_KE06Z          15
```

Assuming your MCU and sensors are on the same board, you will also want to include the following:

```
#define THIS_SHIELD SHIELD_NONE
```

Coordinate System

The coordinate system to be used is set by THISCOORDSYSTEM with valid values being NED, ANDROID and WIN8.

```
// 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
```

Fusion Models

The software is capable of running any or all of the available fusion functions in parallel. The default setting computes all variants.

```
#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)
```

Simply comment any algorithms which you do not need.

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
```

5.1.6 **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 7. 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
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

Bluetooth Packet Structure

Byte #	Function	Units
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	Two numeric fields with the following possible values: Base Board = Bits 4:0: 0. = RESERVED 1. = FRDM-KL25Z 2. = FRDM-K20D50M 3. = RESERVED 4. = FRDM-KL26Z 5. = FRDM-K64F 6. = RESERVED 7. = FRDM-KL46Z 8. = RESERVED 9. = FRDM-K22F 10. = RESERVED 11. = FRDM-KL05Z 12. = RESERVED 13. = FRDM-KL02Z 14. = FRDM-KE02Z 15. = FRDM-KE06Z Sensor Shield Board = Bits 7:5 0. = FRDM-FXS-MULTI-B 1. = None 2. = FRDM-STBC-AGM01 3. = FRDM-STBC-AGM02 All others are reserved. Contact sfusion@freescale.com if the user is interested in having another board added to this list.
35	Packet END = 0x7E	None

Table 7 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 8. Debug Packet Format

Byte #	Function	Units
0	Packet Start = 0x7E	None
1	Packet Type = 0x02	
2	Packet Number	None
4:3	Software Version Number	None
6:5	Systick count / 20	Bytes 6:5 now carry the sysTick count/20
…	Variable Payload	
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 9. 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
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 10. 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	Phi (Roll)	1 LSB = 0.1degree
10:9	Theta (Pitch)	1 LSB = 0.1degree
12:11	Rho (Compass)	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 11. 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 meters
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 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 accelerometer quaternion (tilt) in standard packet
- Q3M# = transmit magnetometer quaternion (auto compass) in standard packet
- Q3G# = transmit gyro-based quaternion in standard packet
- 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)
- 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 7.

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.

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

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

Table 12. Revision history

Rev. No.	Date	Description
0	02/2014	Initial version

Rev. No.	Date	Description
1.0	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	11/2014	Sensor Fusion Release 4.22
1.3	9/2015	Library Version 5.00. This version of the sensor fusion library • has completely rewritten Kalman filters • additional auto compass and gyro-based quaternion computation option • no longer uses an installer. The library is shipped as a zip file. • has a flattened software hierarchy over Release 4.22 The manual has been completely reviewed and updated for consistency with V5.00.

Appendix A – CodeWarrior Run Configurations

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:

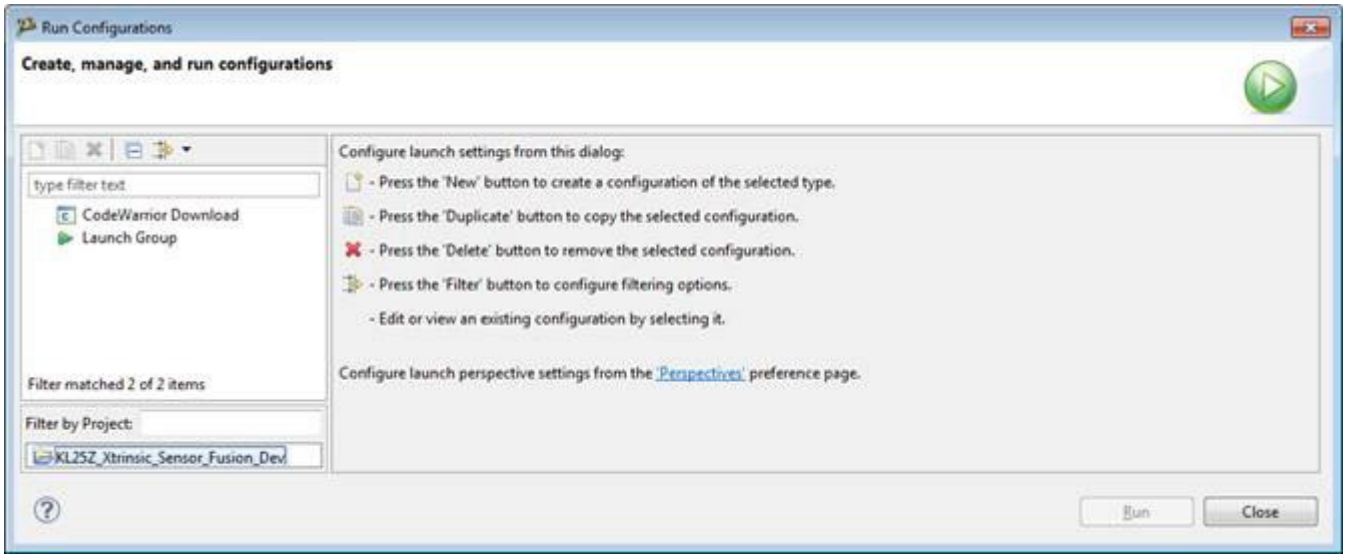

Figure A-1. CodeWarrior Run Configuration window

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

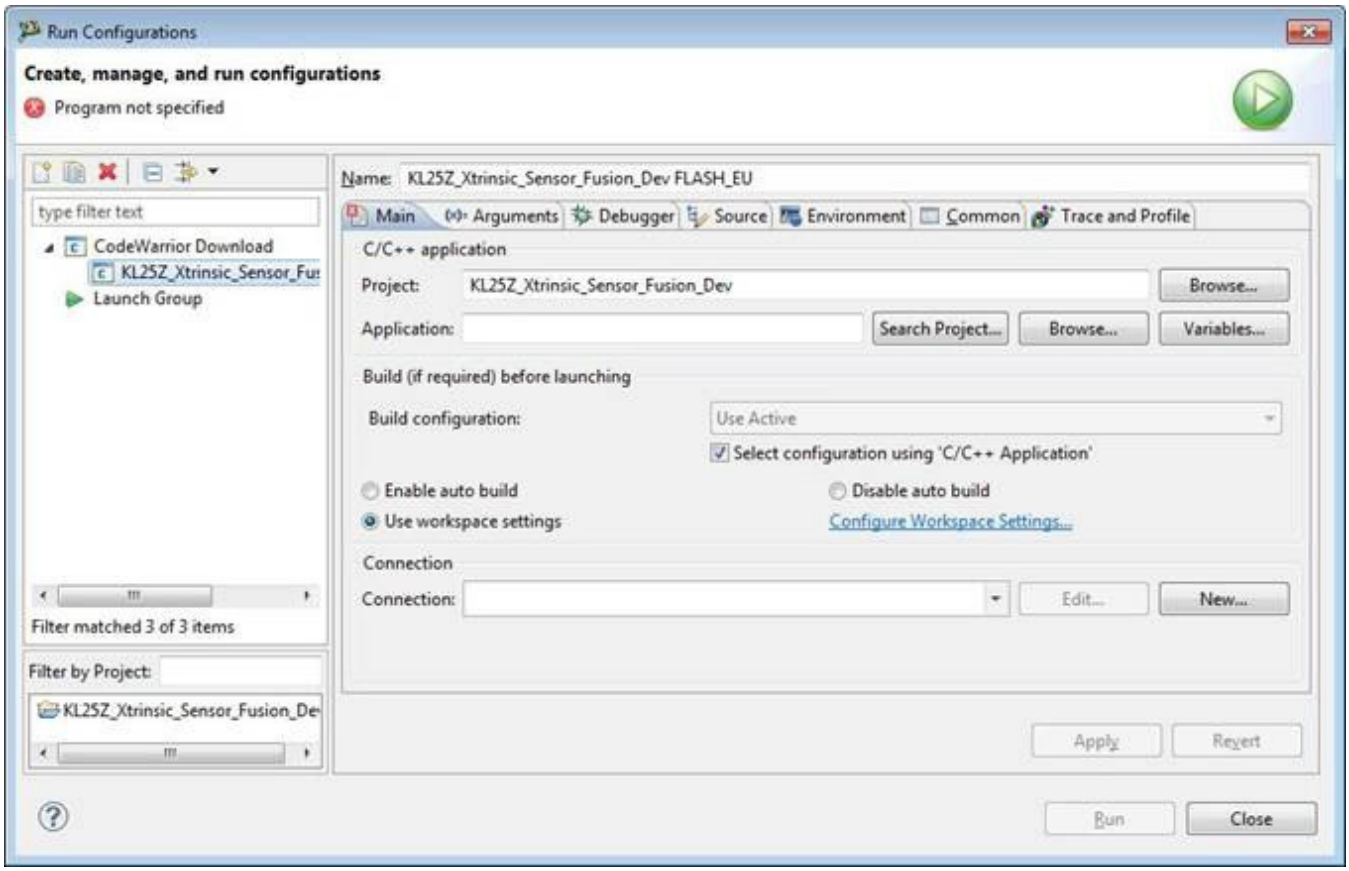

Figure A-2. CodeWarrior creating new launch configuration

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.

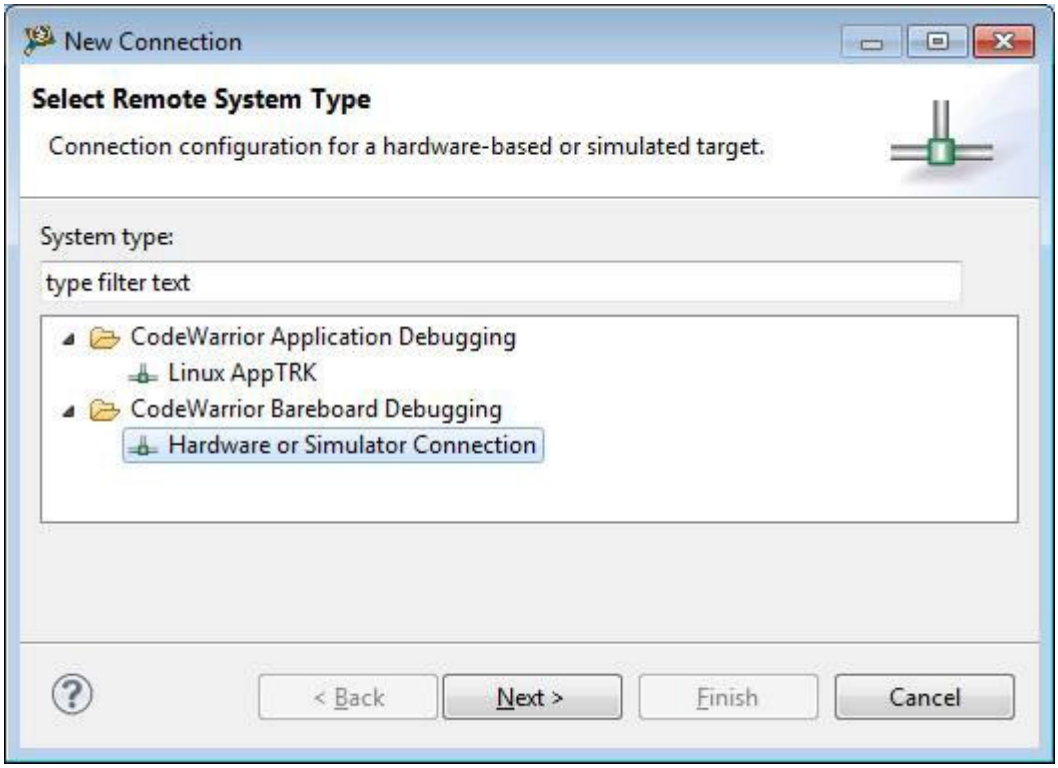

Figure A-3. Setting CodeWarrior Remote System Type

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

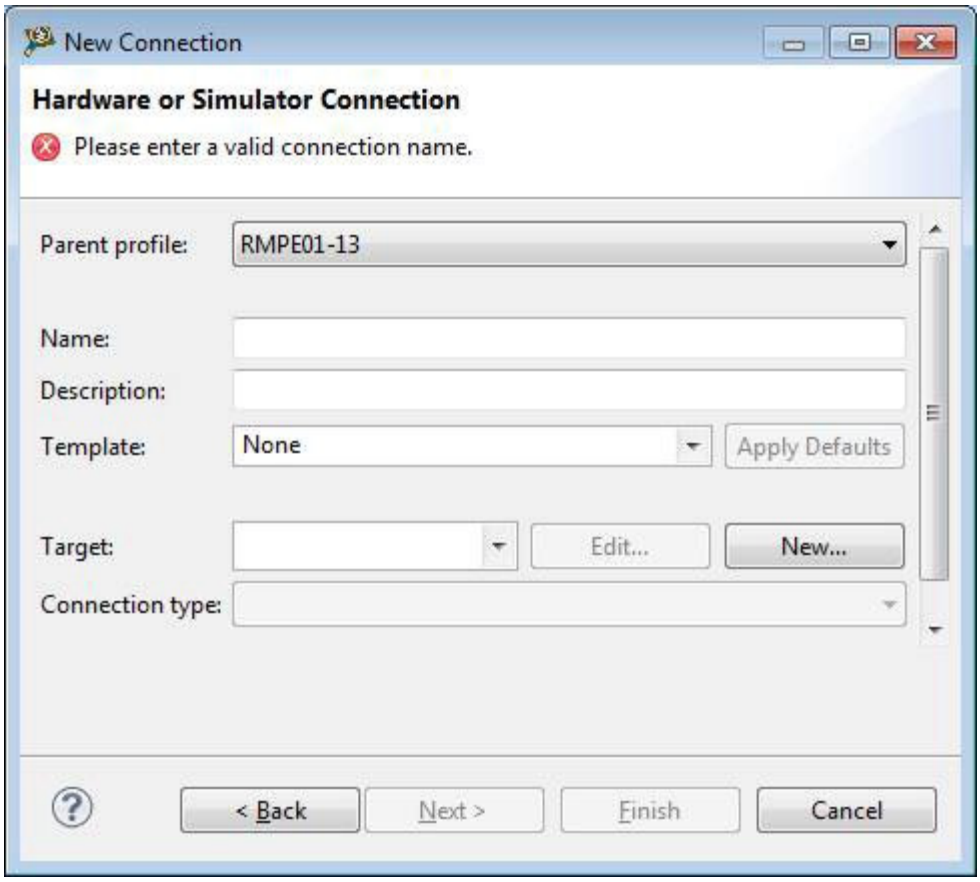

Figure A-4. Naming new CodeWarrior connection

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.

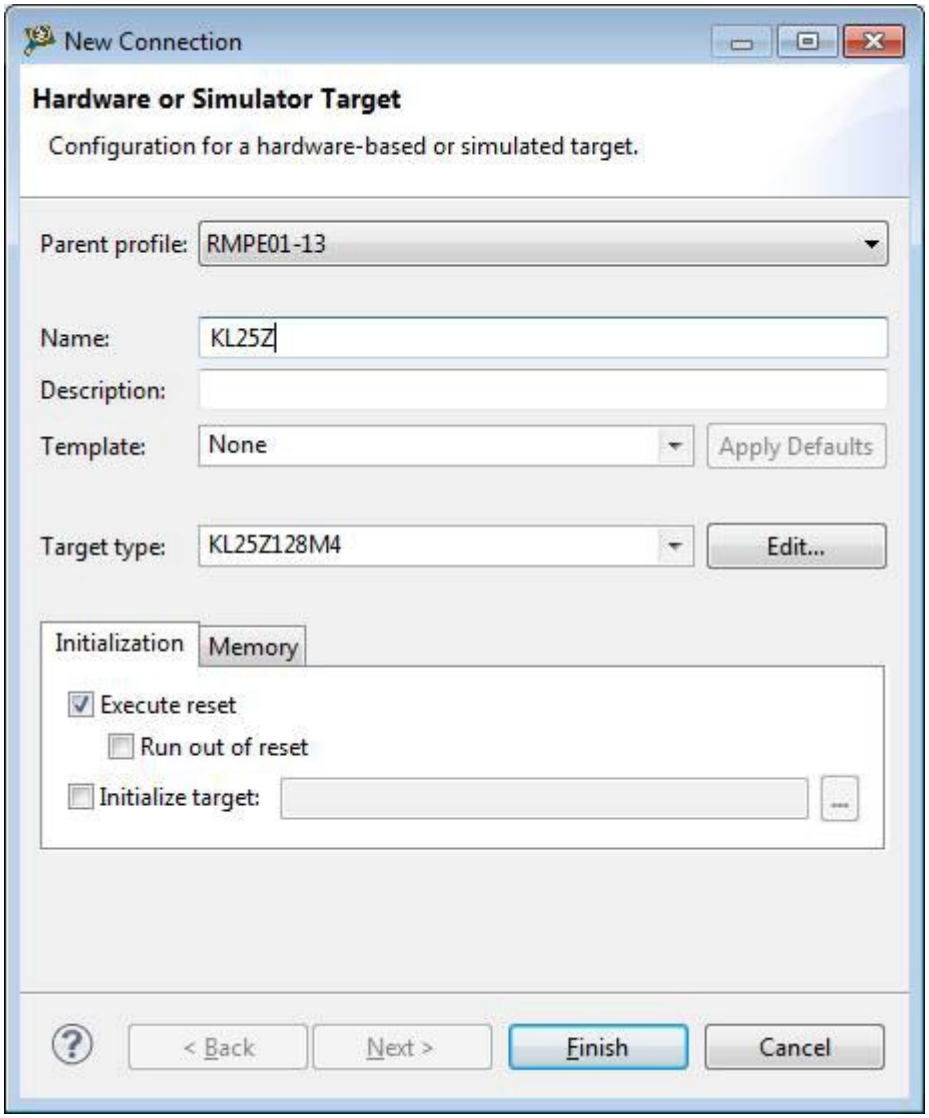

Figure A-5. Selecting CodeWarrior Target type

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.

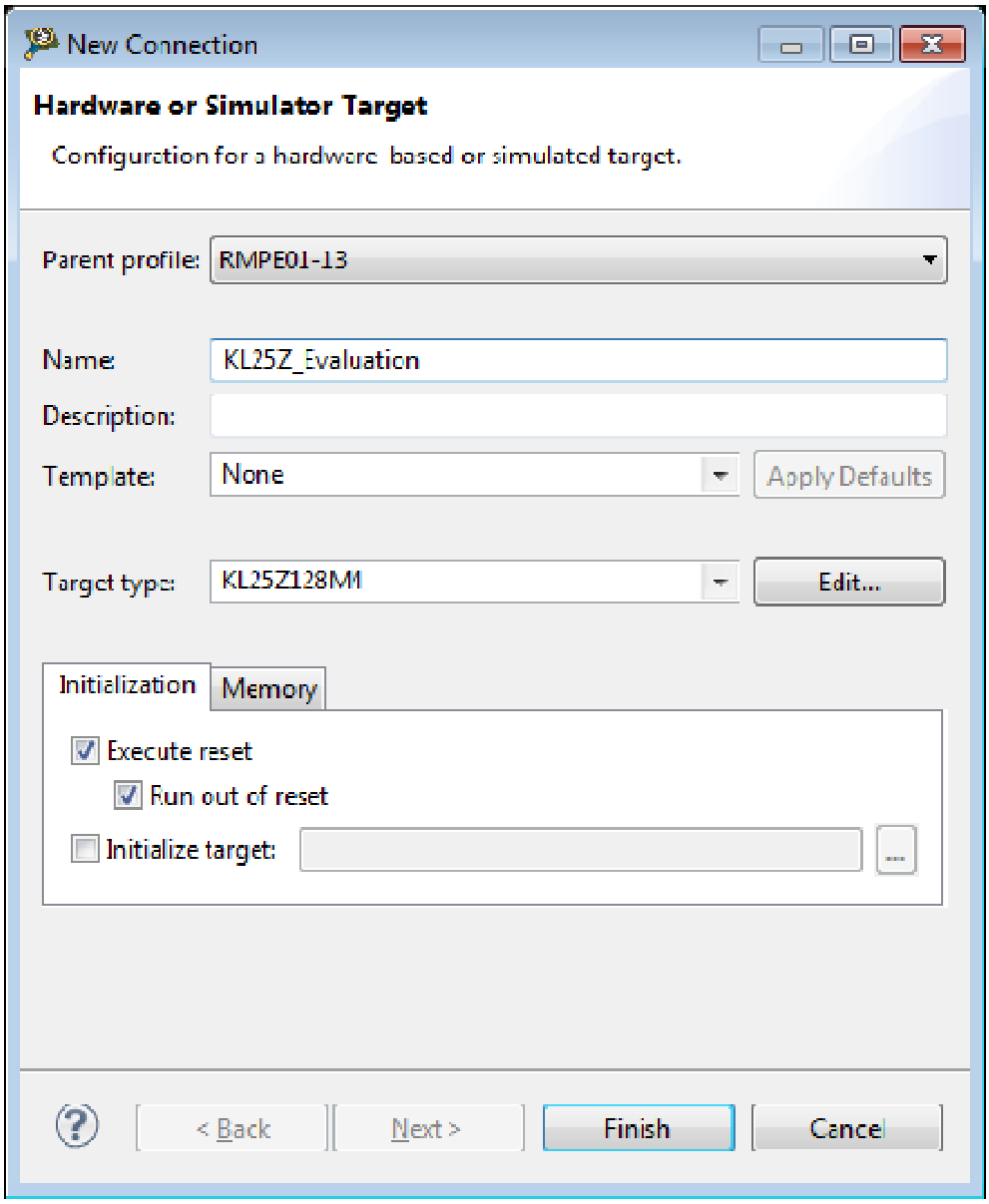

Figure A-6. CodeWarrior new connect example setup

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.

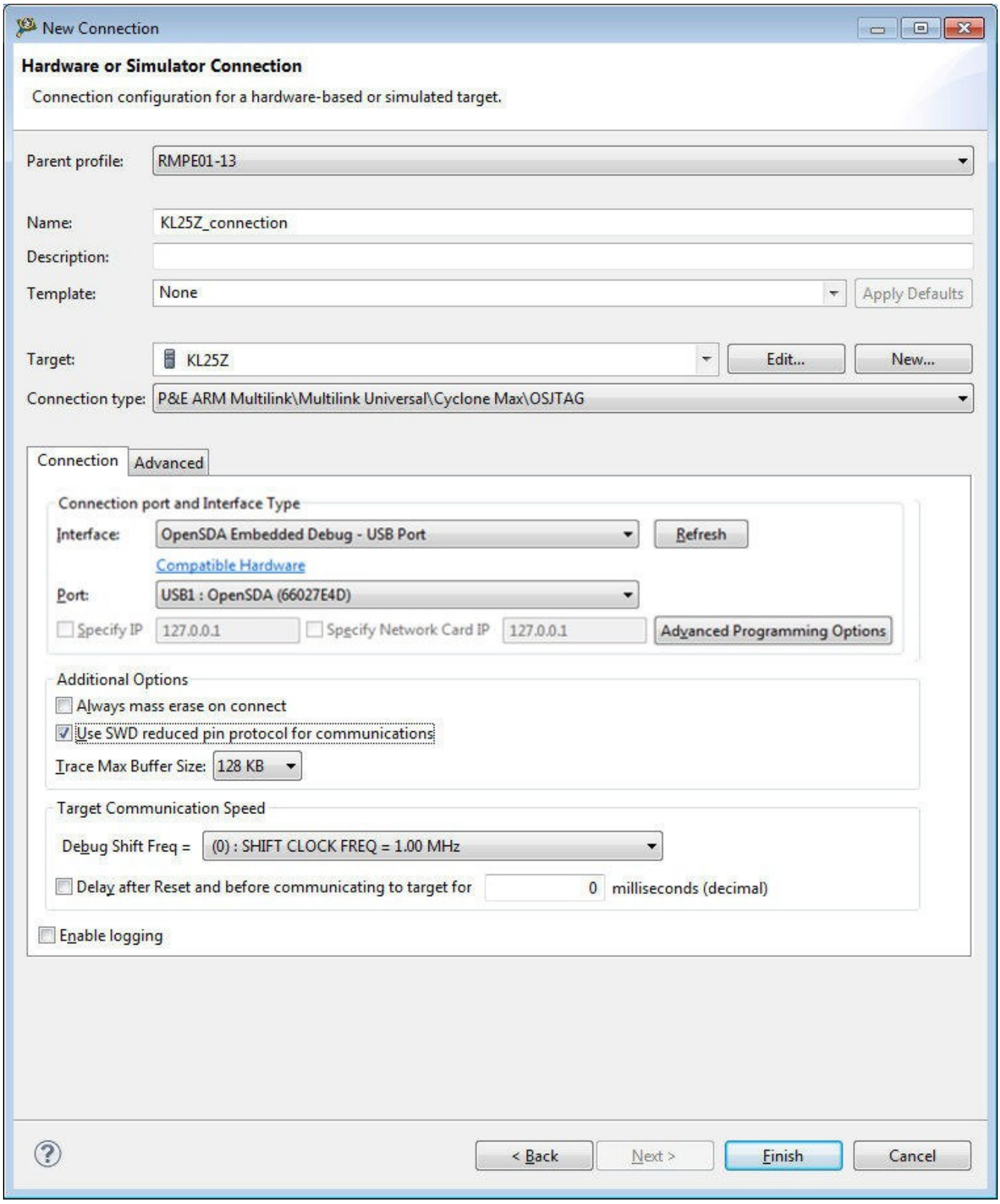

Figure A-7. CodeWarrior run configuration dialog

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.

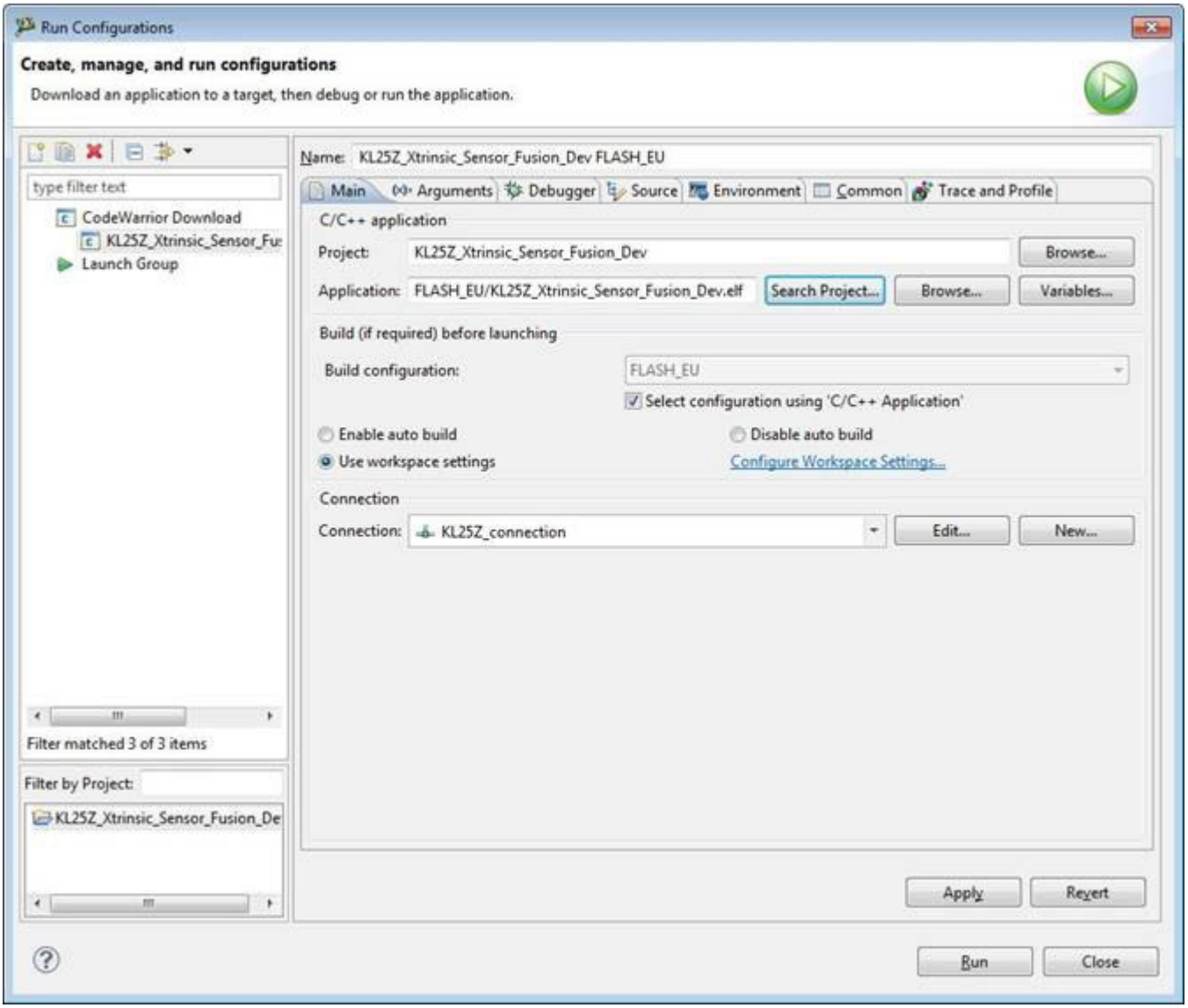

Figure A-8. Setting the elf file in CodeWarrior

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.

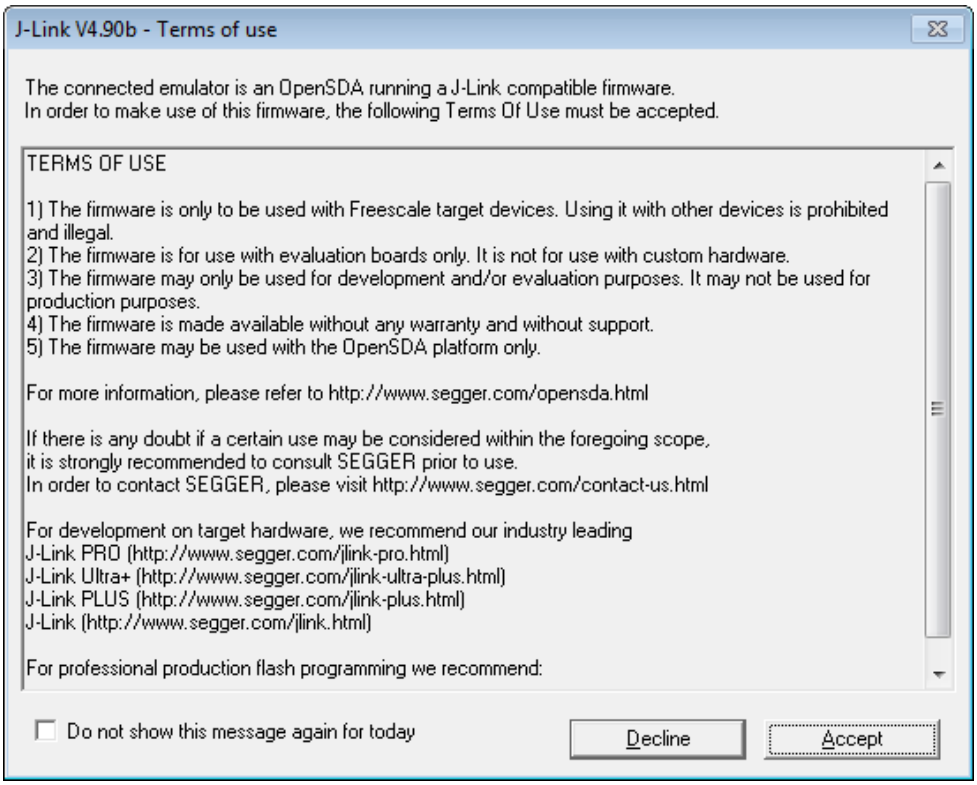

Figure A-9. Freescale Software Terms of Use

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

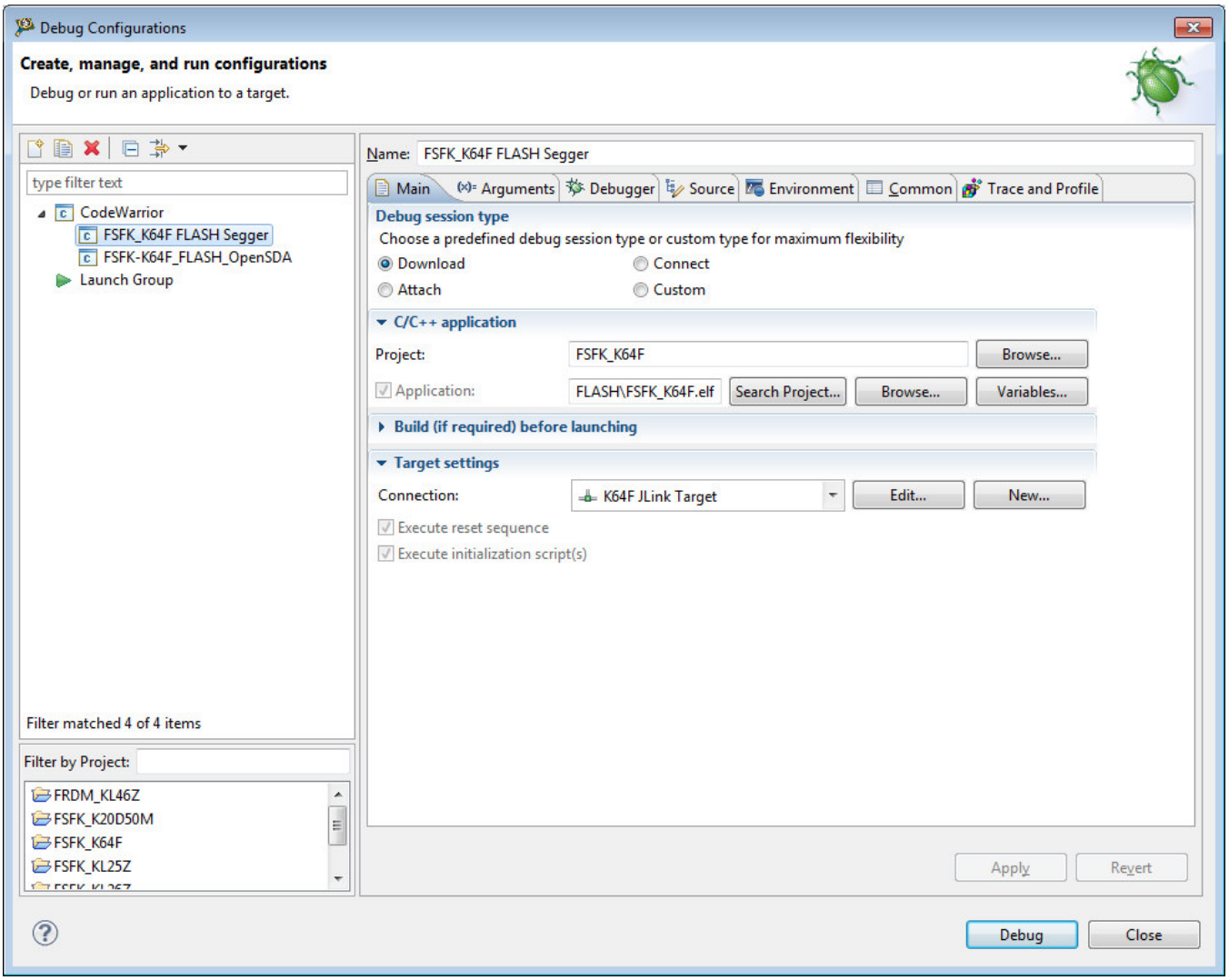

Figure A-10. Segger OpenSDA Debug Configuration

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: FSFL_Prod_UG
Revision Rev. 1.3, 9/2015