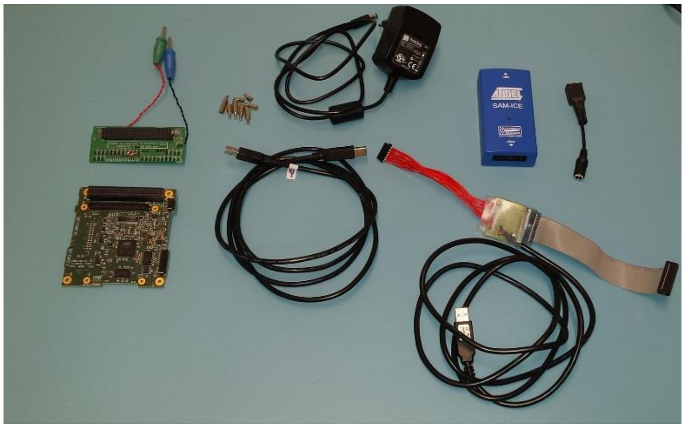
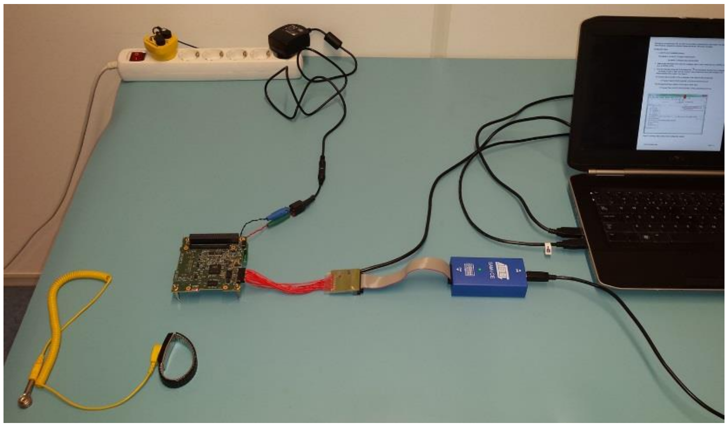
Version : 2.2
Date: 17-08-2015

Innovative Solutions In Space

ISIS-OBC Quickstart Guide





Important Notice: Before performing the hardware related steps described in the ISIS-OBC Quickstart Guide, please read through the ISIS-OBC Datasheet for information on safety instructions related to the handling of the ISIS-OBC hardware.

Table of Contents

1 Introduction

The ISIS On Board Computer (ISIS-OBC) is a general purpose computing board that adheres to the Cubesat standard. It provides powerful processing capabilities and comes with an extensive set of software tailored specifically for space applications. The basic interfaces are standardized improving robustness, while a high level of customization can be provided through the daughterboard concept without the need to sacrifice additional volume.

This document provides aid for getting started with embedded software development for the ISIS-OBC hardware. Chapter 2 provides a general discussion on embedded systems that provides a primer for people unfamiliar with embedded systems and embedded software development. Chapter 3 dives into the details of installing drivers and the ISIS-OBC Software Development Kit (SDK). Chapter 4 describes how to connect the ISIS-OBC hardware to the development machine. Eclipse configuration in Chapter 5 concludes the setup of the IDE. Code development and compiling/running/debugging application code is discussed in Chapters 6 and 7. Chapter 8 provides information on flashing code to the non-volatile memory. Chapter 9 describes how to get support.

2 Embedded Systems Software Design

This section provides a primer to embedded systems design. It can be skipped without loss of information.

An embedded system is a generic term that refers to any computer system that is located along with the hardware it controls. The main control unit of the system is referred to as the controller, and has historically been carefully selected to supply just enough processing power to perform their controller function while keeping cost and power consumption to a minimum. These carefully selected controllers are referred to as micro-controllers. With processing power becoming less expensive in terms of power consumption and cost, increasing amounts of embedded systems employ a more generic and much more powerful controller unit referred to as a central processing unit (CPU). The ISIS-OBC uses an ARM architecture based processing unit.

A computer running Microsoft Windows is used to develop and deploy embedded software to the ISIS-OBC controller hardware. This machine will be referred to as the *development machine*. The development machine requires drivers to be installed to allow interfacing with attached hardware, as well as programs that use the drivers to perform various tasks. ISIS provides a Software Development Kit (SDK) to provide all required programs.

To interact with the ISIS-OBC hardware two types of equipment are used: a programmer to deploy and debug embedded programs running on the ISIS-OBC processor, and an USB-to-Serial converter which provides a separate full-duplex data channel between the development machine and the ISIS-OBC controller. Hardware interfaces to the serial data connection lines are usually provided for in Electronic Ground Support Equipment (EGSE), and provide a mechanism for relaying debug messages, commanding of embedded programs during ground testing and setting final configurations before flight.

Programs that control the hardware attached to the development machine by interacting with the installed drivers are contained within the ISIS-OBC Software Development Kit (SDK) installer package, along with applications that provide translation of program code to machine instructions (the compiler/linker toolchain) and a programming environment that aids an embedded developer in efficiently producing embeddable software. The collection of these programs is usually brought together and controlled from within a single application, which is referred to as the Integrated Development Environment (IDE).

Once coding of an embeddable program has been completed, the IDE is instructed to perform a build using the toolchain programs. A successful build produces a file that contains machine instructions specific to the embedded processor architecture. The IDE can then instruct the programmer hardware to transfer the machine code to the embedded processor and execute the program. The embedded software developer has included code within the program that emits debug trace and/or error messages, which are relayed over the USB-to-Serial interface to the development machine. The debug messages are received and displayed in a terminal application, which allows the developer or e.g. test conductor to verify the program's behavior. The terminal application also allows sending data over the USB-to-Serial interface to the embedded processor, at which point code included by the embedded software developer can take appropriate action.

Once correct program behavior has been verified by using the programming hardware, the program is 'burned' into embedded memory using some type of non-volatile memory. Frequently used non-volatile memory are NAND- or NOR-FLASH, while FRAM is a near-future contender due to the many improvements it provides. From this point onwards the hardware can perform its functions without the need for an attached development machine.

3 Setting up the Integrated Development Environment

To set up the integrated development environment on the development machine several software items must be installed and configured. The installation is comprised of the following steps:

1. Install J-Link driver of the SAM-ICE programmer/debugger
2. Install driver of the FTDI USB-to-Serial device
3. Install the ISIS-OBC SDK containing required IDE application software
4. Extract the ISIS-OBC libraries.
5. Extract the ISIS-OBC first-project.

Each of the installation steps will be described in the following sections.

3.1 Install Programmer Hardware Driver: JLink from Segger

Download and install the Jlink drivers for the programmer/debugger, shown in Figure 1. The newest driver version can be installed from:

https://www.segger.com/jlink-software.html

Older versions can be downloaded from:

https://www.segger.com/j-link-older-versions.html

Note: version 4.52 of the JLink driver software has been extensively used during testing. Use of other/newer versions could provide different behavior and might not work properly. Several versions of the driver can be installed on a development machine and will be able to run independently according to Segger. The driver version used during debugging is set using the GDB Server configuration explained in Section 5.2

The serial number can be found on the bottom of the SAM-ICE debugger as shown in Figure 2.

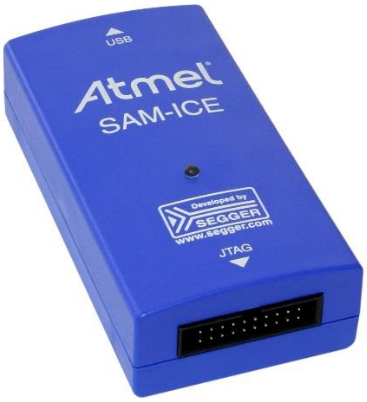


Figure 1: SEGGER SAM-ICE Programmer/Debugger Atmel OEM version (src: segger.com)

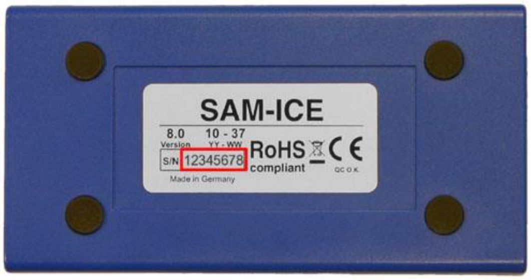


Figure 2: SAM-ICE S/N is located on the bottom (src: segger.com)

3.2 Install USB-to-Serial Hardware Driver: FTDI

Download and install drivers for the FTDI USB-to-Serial adapter from:

http://www.ftdichip.com/Drivers/VCP.htm.

Locate the setup executable that matches the Windows architecture used on the development machine.

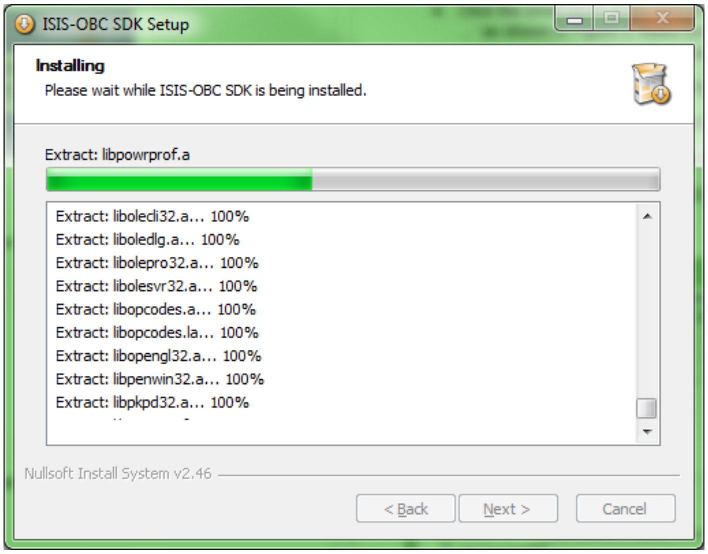
3.3 Install ISIS-OBC SDK

Install the ISIS-OBC SDK (Software Development Kit) provided by ISIS. The installer 'ISIS-OBC SDK.exe' can be found with the board's software package.

The installer allows optional install of the following applications:

- Eclipse IDE
- Atmel SAM-BA for interfacing with ARM processor/FLASH memory hardware
- PuTTY Terminal Client
- MinGW GNU compiler/linkers for Windows
- ARM Toolchain for compiling code for the ARM processor architecture

Note: All software is installed into a fixed directory with path: C:\ISIS\



ISIS-OBC SDK Setup

Installing

Please wait while ISIS-OBC SDK is being installed.

Extract: libpowrprof.a

```
Extract: liboledli32.a... 100%
Extract: liboledlg.a... 100%
Extract: libolepro32.a... 100%
Extract: libolesvr32.a... 100%
Extract: libopcodes.a... 100%
Extract: libopcodes.la... 100%
Extract: libopengl32.a... 100%
Extract: libpenwin32.a... 100%
Extract: libpkpd32.a... 100%
```

Figure 3: The ISIS-OBC SDK installer

3.4 Extract ISIS-OBC First-project

The ISIS-OBC first-project is an Eclipse project that serves as a template for new ISIS-OBC code development projects. Each new project can be started by making a copy-of the template project after which search-and-replace is performed to update project name and file linkage to that of the new project.

Simply extract the 'isis-obc-first-project.zip' contents into e.g. a subfolder of the ISIS root folder located at C:\ISIS\. For our purposes we'll use the folder C:\ISIS\ISIS-OBC\ to place new ISIS-OBC development projects in. However any other location will do.

3.5 Extract ISIS-OBC Libraries

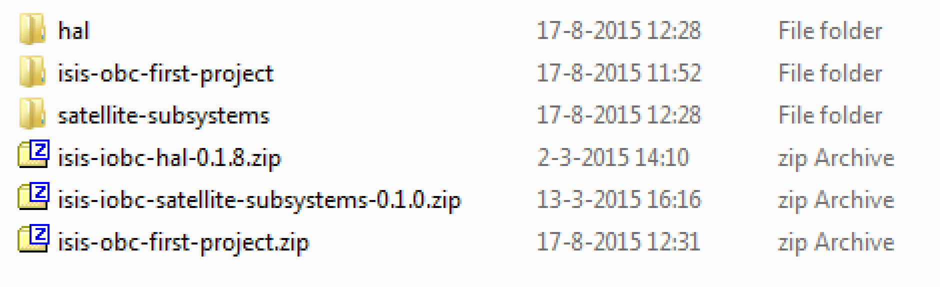
The ISIS-OBC libraries provide means for rapid application development by acting as a layer between the low-level hardware and the developed application.

There are three different modules available at the time of writing:

- Hardware Abstraction Layer (HAL): provides low level generic drivers for the SAMG20 processor peripherals
- Satellite Subsystems (SSU): provides control functions for commonly used satellite subsystems such as ISIS-MTQ, ISIS-TRXUV/VU. It uses the HAL library for performing lower level operations.
- Mission Support (MIS): provides higher level application functions. It uses the HAL library for performing lower level operations.

Depending on the purchased package, either only the HAL, HAL+SSU or the HAL+SSU+MIS libraries have been provided as versioned zip files. The content of these zip files should be extracted to the same folder as where the 'isis-obc-first-project' resides.

The default location that is used within this document is C:\ISIS\ISIS-OBC\ (see Section 3.4). The resulting directory structure for the HAL+SSU option looks as is shown in Figure 4.



hal	17-8-2015 12:28	File folder
isis-obc-first-project	17-8-2015 11:52	File folder
satellite-subsystems	17-8-2015 12:28	File folder
isis-iobc-hal-0.1.8.zip	2-3-2015 14:10	zip Archive
isis-iobc-satellite-subsystems-0.1.0.zip	13-3-2015 16:16	zip Archive
isis-obc-first-project.zip	17-8-2015 12:31	zip Archive

Figure 4: Final file structure of the C:\ISIS\ISIS-OBC\ main project folder. Keeping the zip files is optional.

4 Setting-up the Hardware

This section will provide instructions on how to set up the ISIS-OBC hardware and related tooling.

Important Notice: Before performing the hardware related steps described in this section, please read through the ISIS-OBC Datasheet for information on safety instructions related to the handling of the ISIS-OBC hardware.

The following steps outline the required connections to setting up the ISIS-OBC hardware.

- Connect the programming adapter to the ISIS-OBC.
- Connect the SAM-ICE programmer/debugger to the programming cable.
- Connect the ISIS-OBC to the 3.3V Wall Adapter, and then to the power socket. When a separate power supply is used, it is recommended to set the current limit to 150 mA for safety.
- Connect the USB cable to the SAM-ICE and to your computer.
- Connect the Debug port (USB-to-Serial) cable to your computer.

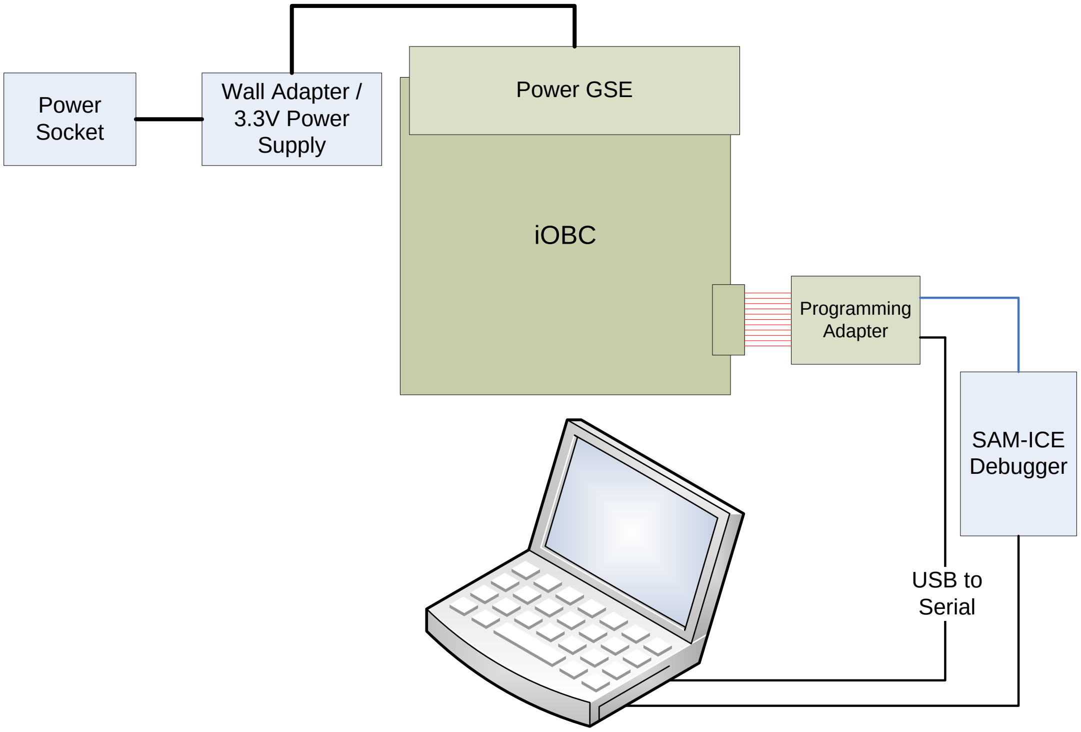


Figure 5: ISIS-OBC embedded software development hardware configuration

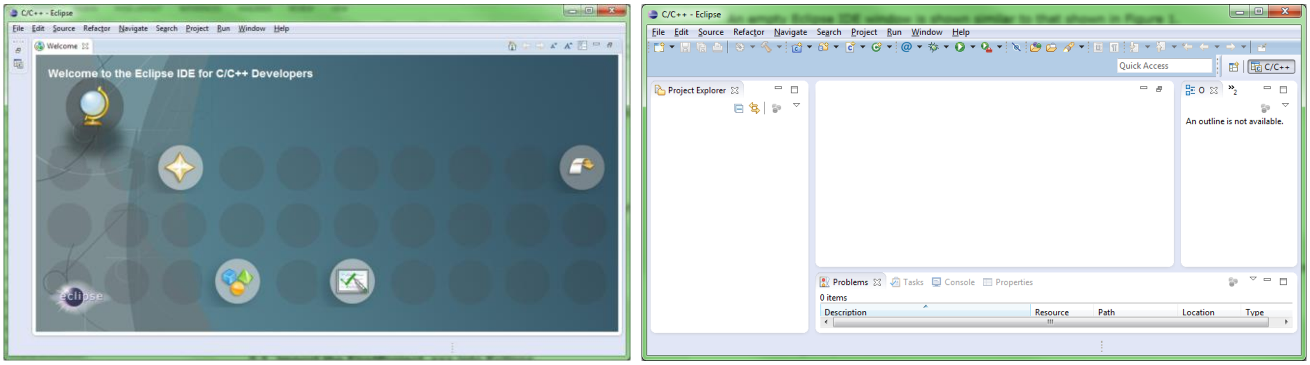
5 Configuring Eclipse IDE

The Eclipse IDE application can now be opened using the start menu links.

To do this navigate to and open:

Start > All Programs > ISIS-OBC SDK > Eclipse IDE

An empty Eclipse IDE window is shown similar to that shown in Figure 6 (left). The initial welcome screen can be closed straight away with the x symbol on the 'Welcome' tab. The default project screen will appear as shown in Figure 6 (right).



Welcome to the Eclipse IDE for C/C++ Developers

An outline is not available.

0 items

Figure 6: Eclipse IDE welcome screen (left) and project screen (right) shown after closing the welcome screen

The following first-time configuration steps need to be performed to allow development within the Eclipse environment:

1. Import "isis-obc-first-project" into Eclipse
2. Add the Jlink GDB server launch configuration to Eclipse
3. Configure path to J-Link GDB server executable
4. Add the Debug/Release launch configurations to Eclipse

These steps will be described in the following sections.

5.1 Import the 'isis-obc-first-project' into Eclipse

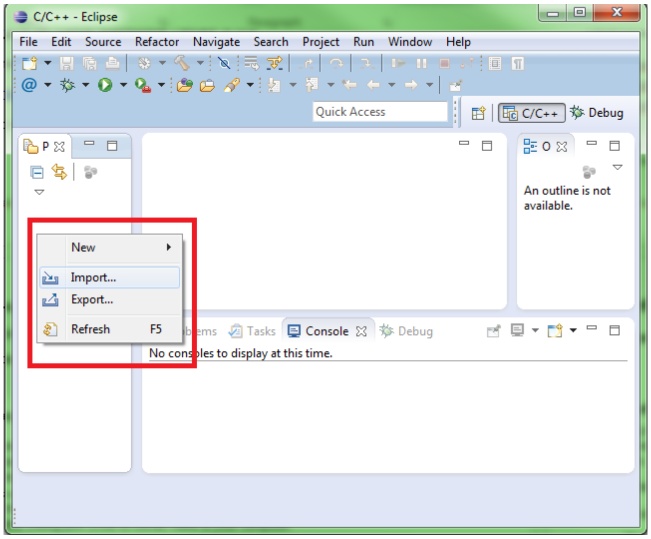
To import the 'isis-obc-first-project' into Eclipse, which was copied as described in section 3.5, the following steps need to be performed:

1. Right click in Eclipse Project Explorer and click 'Import … '

   The Eclipse Project Explorer is found on the left of the Eclipse application window as shown in Figure 7.

2. Select 'General' > 'Existing Projects into Workspace' and click Next. See Figure 8.
3. Select 'Choose root directory' and select the location of the 'isis-obc-first-project' folder. Default used in this document: C:\ISIS\ISIS-OBC\.

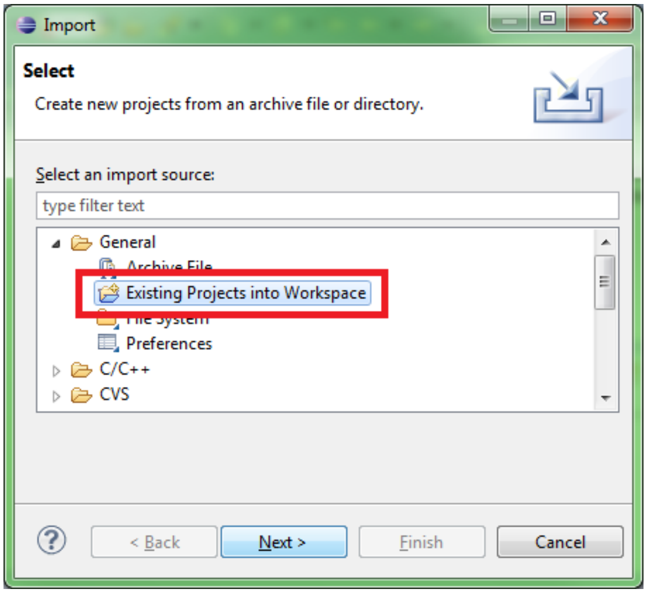
4. The ‘isis-obc-first-project’ should be selected automatically. See Figure 9.
5. Click ‘Finish’ to complete the import



An outline is not available.

No consoles to display at this time.

Figure 7: Select Eclipse Project Explorer import function

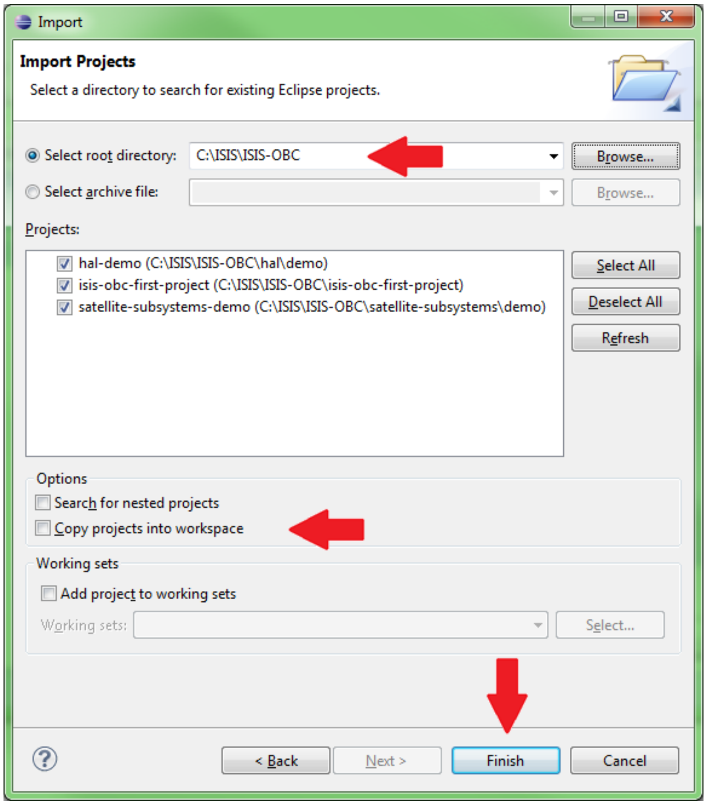


Create new projects from an archive file or directory.

Select an import source:

type filter text

Figure 8: (import …) Select General > ‘Existing Projects into Workspace’



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\ISIS\ISIS-OBC

○ Select archive file:

Projects:

☑ hal-demo (C:\ISIS\ISIS-OBC\hal\demo)
☑ isis-obc-first-project (C:\ISIS\ISIS-OBC\isis-obc-first-project)
☑ satellite-subsystems-demo (C:\ISIS\ISIS-OBC\satellite-subsystems\demo)

Options

□ Search for nested projects
□ Copy projects into workspace

Working sets

□ Add project to working sets

Working sets:

Figure 9: isis-obc-first-project and any library demo projects are selected for import. IMPORTANT: 'Copy projects into workspace' should be deselected, to make sure the project source files are not copied into the workspace.

5.2 Configure the J-Link GDB Server Configuration

Debugging services are provided for using a GDB server application, for which an executable is installed by the JLink installer. The following steps should be completed to make sure Eclipse can find the GDB server executable that should have been installed as described in Section 3.1.

To add the launch configuration do the following:

1. Select the Jlink-samg20-GDB.launch file in the Eclipse Project Explorer window
2. Click the small down arrow next to the debug-tools icon, expand 'Run As' and click Jlink-samg20-GDB

The Jlink GDB debug launch configuration is now added to Eclipse.

The steps to update the path to the JLink GDB server application are as follows:

1. Locate the JLink installation directory.

   The default is (on win32) C:\Program Files\SEGGER

   (on win64) C:\Program Files (x86)\SEGGER

2. Make a note of the folder name within the installation folder. It has a format such as JLinkARM_Vxxxx (e.g. JLinkARM_V478c).

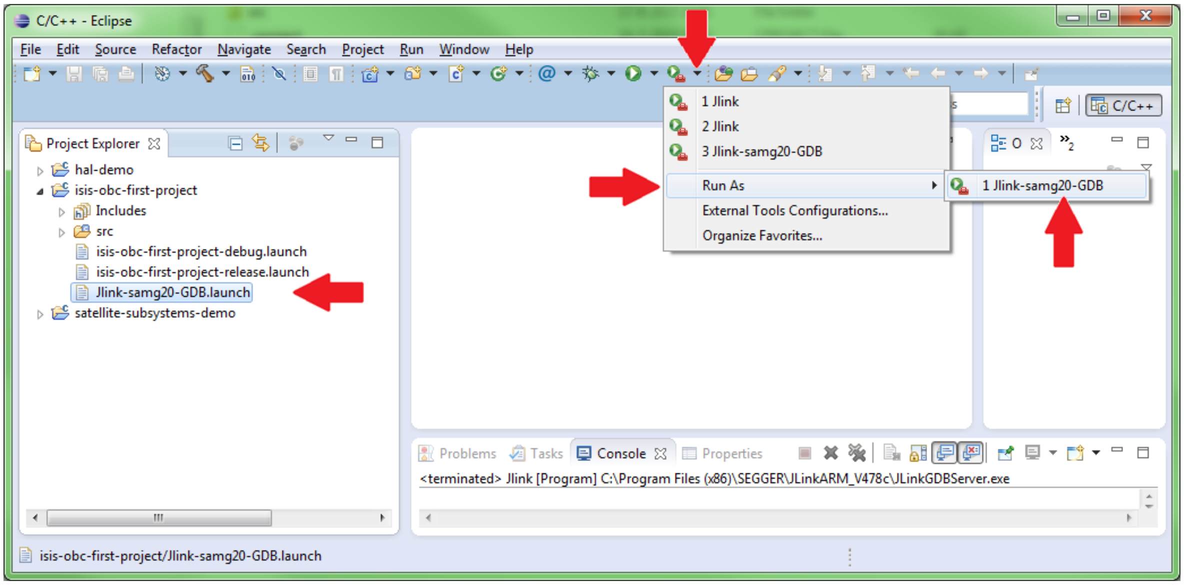
3. Click the small down arrow next to the debug-tools icon and choose '*External Tools Configurations …'* as shown in Figure 11.
4. Select the 'Jlink-samg20-GDB'.
   Note: This is the only configuration that needs to be updated when this configuration is the only one that will be used during debugging.
5. Make sure the '*Location*' string contains the correct path using the JLink version installed on the system. Copy/paste the path to an explorer for quick verification. See Figure 12.

E.g. the location string with JLinkARM_V478c on Windows 32-bit using the default install path becomes:

C:\Program Files\SEGGER\JLinkARM_V478c\JLinkGDBServerCL.exe

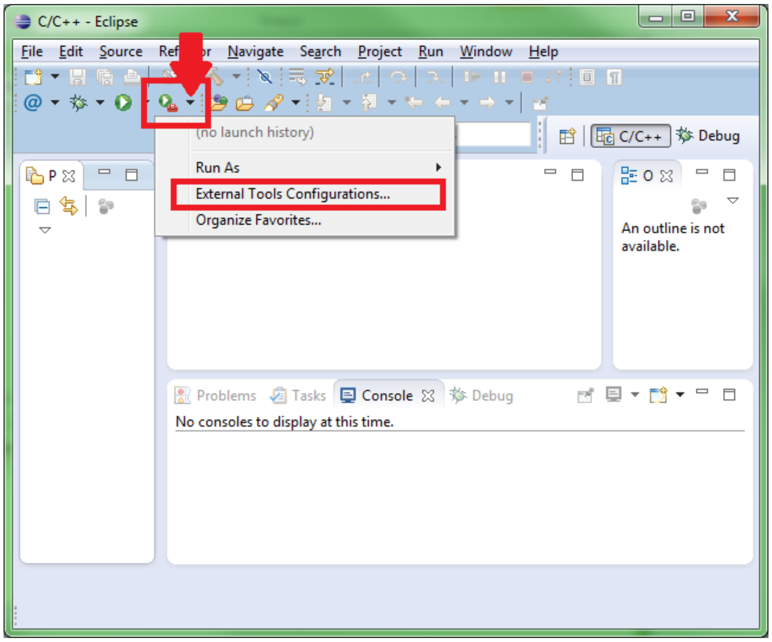
The Windows 64-bit has a different 32-bit program files folder name and becomes:

C:\Program Files (x86)\SEGGER\JLinkARM_V478c\JLinkGDBServerCL.exe


C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
1 Jlink
2 Jlink
3 Jlink-samg20-GDB
Run As
External Tools Configurations...
Organize Favorites...
1 Jlink-samg20-GDB
C/C++
Project Explorer
hal-demo
isis-obc-first-project
Includes
src
isis-obc-first-project-debug.launch
isis-obc-first-project-release.launch
Jlink-samg20-GDB.launch
satellite-subsystems-demo
Problems
Tasks
Console
Properties
isis-obc-first-project/Jlink-samg20-GDB.launch

```
<terminated> Jlink [Program] C:\Program Files (x86)\SEGGER\JLinkARM_V478c\JLinkGDBServer.exe
```

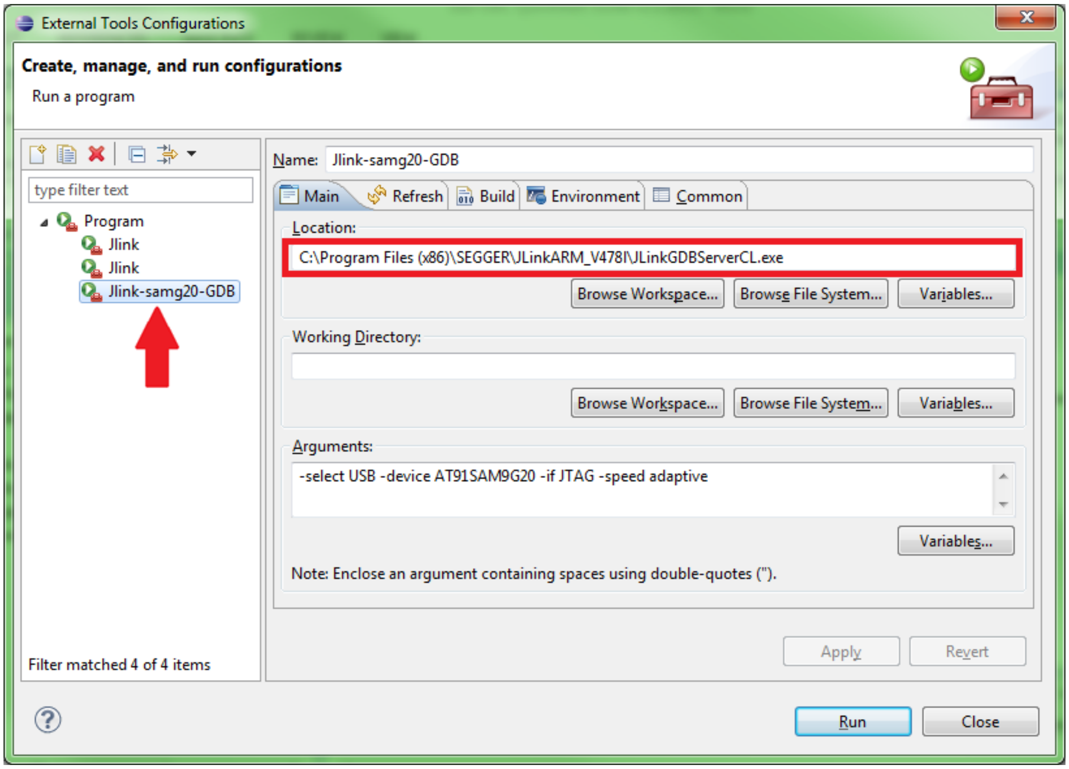
Figure 10: Verify/Add the GDB launch configuration


C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
(no launch history)
Run As
External Tools Configurations...
Organize Favorites...
C/C++
Debug
Problems Tasks Console Debug

An outline is not available.

No consoles to display at this time.

Figure 11: Opening Eclipse external tools configuration window



Name: Jlink-samg20-GDB

Location: C:\Program Files (x86)\SEGGER\JLinkARM_V478l\JLinkGDBServerCL.exe

Working Directory:

Arguments: -select USB -device AT91SAM9G20 -if JTAG -speed adaptive

Note: Enclose an argument containing spaces using double-quotes (").

Figure 12: Update/Verify that the correct path to the GBD server executable is in place for the ‘JLink-samg20-GDB’ launch configuration.

5.3 Check/Add the Debug Launch Configuration

The debug launch configuration should be added to the debug list by performing the following steps.

1. Select isis-obc-first-project-debug.launch in the project explorer.

2. Click the <u>small down arrow</u> next to the debug icon
3. Expand 'Debug As'
4. Click isis-obc-first-project-debug.

This adds the debug launch configuration to the list. The same can be performed for the release config. See Figure 13.

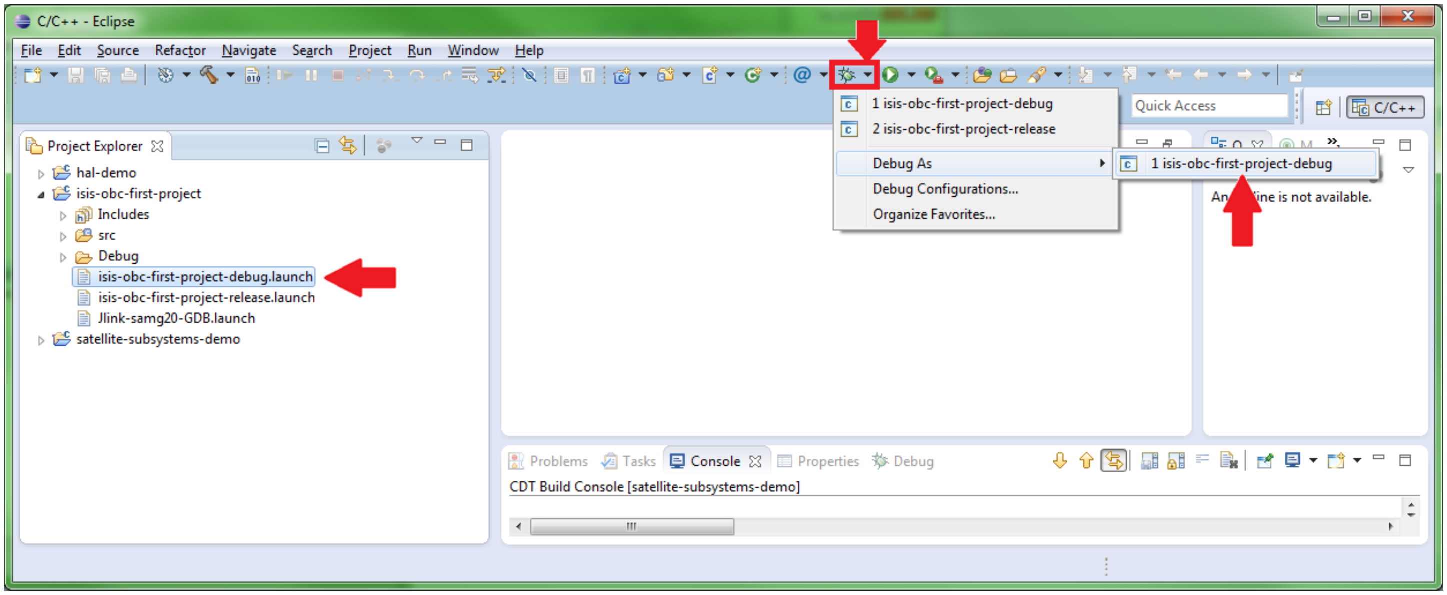

C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
1 isis-obc-first-project-debug
2 isis-obc-first-project-release
Debug As
1 isis-obc-first-project-debug
Debug Configurations...
Organize Favorites...
Quick Access
C/C++
Project Explorer
hal-demo
isis-obc-first-project
Includes
src
Debug
isis-obc-first-project-debug.launch
isis-obc-first-project-release.launch
Jlink-samg20-GDB.launch
satellite-subsystems-demo
An outline is not available.
Problems
Tasks
Console
Properties
Debug
CDT Build Console [satellite-subsystems-demo]

Figure 13: Adding the debug launch configuration

6 Code Development and Debug/Release Builds

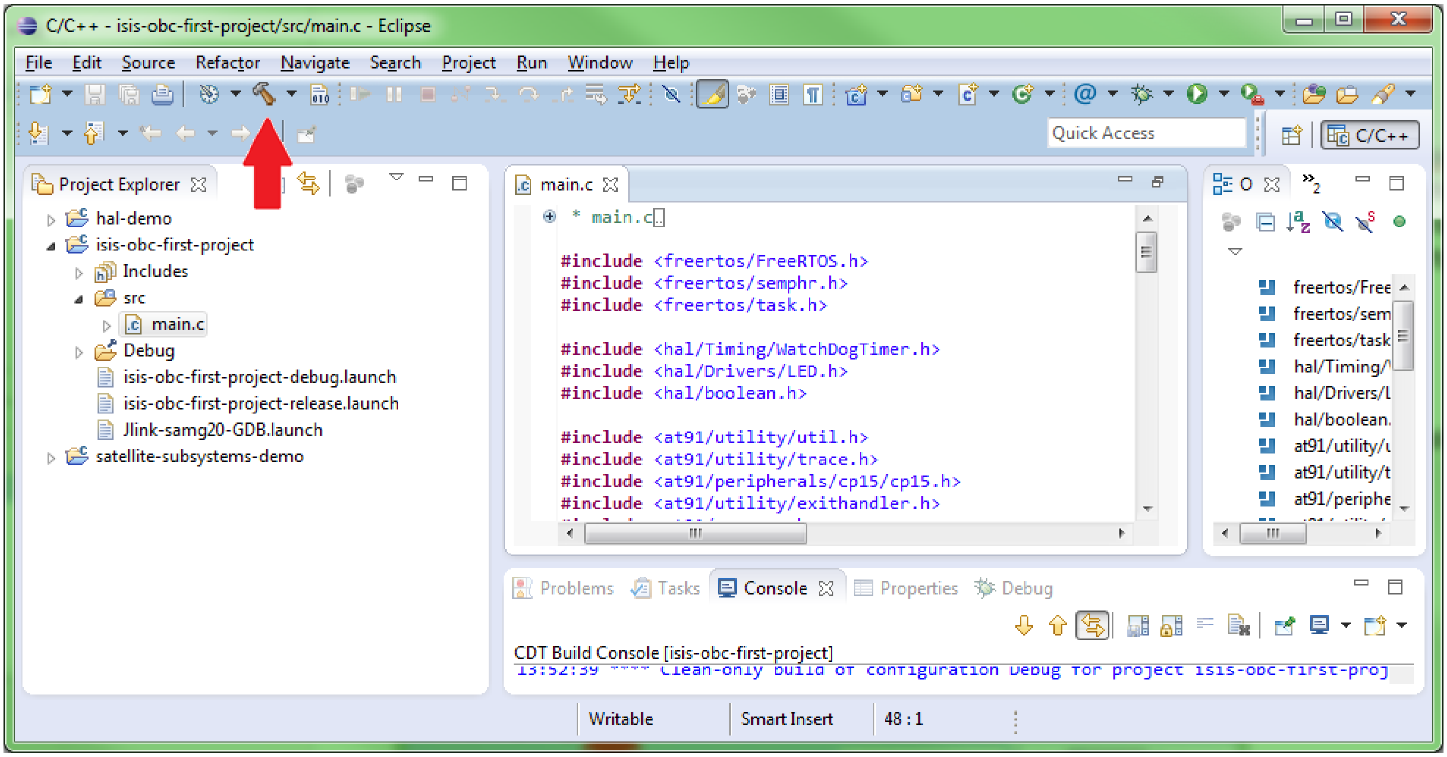
With the IDE set-up as described in the previous sections, the code provided in the isis-obc-first-project can be build and used for checking correct IDE behaviour. The main.c starts up a main task that enters an infinite loop and blinks the ISIS-OBC LEDs in specific patterns.

6.1 Compiling Code

To compile the project do the following:

- Make sure the project is selected by clicking on its name in the Project Explorer window, which is located on the left-hand side of the window.
- Click the build 🔨 icon.

This is shown in Figure 14.



```
 * main.c

#include <freertos/FreeRTOS.h>
#include <freertos/semphr.h>
#include <freertos/task.h>

#include <hal/Timing/WatchDogTimer.h>
#include <hal/Drivers/LED.h>
#include <hal/boolean.h>

#include <at91/utility/util.h>
#include <at91/utility/trace.h>
#include <at91/peripherals/cp15/cp15.h>
#include <at91/utility/exithandler.h>
```

```
CDT Build Console [isis-obc-first-project]
13:52:39 **** Clean-only build of configuration Debug for project isis-obc-first-proj
```

Figure 14: isis-obc-first-project opened in the Eclipse IDE, with the red arrow indicating the Build icon

6.2 ISIS-OBC First-project as Template

The ISIS-OBC first-project can be used as a template for new projects by a simple copy/paste operation. This can be directly performed from the Project Explorer. The Project Explorer also allows seamless copy and paste operations to/from the Windows (file) explorer.

6.2.1 Creating a New Project Using the Template

To create a new project using the template the following steps should be performed:

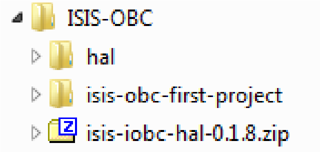
1. Create a copy-in-place of the isis-obc-first-project directory

2. Open each of the Eclipse files located in the new project directory (.cproject, .project and *.launch) using a text editor (e.g. notepad) and perform a rename of all occurrences of *isis-obc-first-project* to the new project name.

6.2.2 Library Linkage

The linkage to the libraries within a project make use of relative paths, and expect the library folder in the same folder as the project root directory. This means the library root directories must be in the same directory as the project root directory.

An example of a directory structure is shown in Figure 15. The example shows the isis-obc-hal-0.1.8 zip and the 'hal' directory that was extracted from this zip. The first-project can now find the 'hal' library and will successfully build. Note that the zip file is not used and could optionally be removed.



ISIS-OBC
hal
isis-obc-first-project
isis-iobc-hal-0.1.8.zip

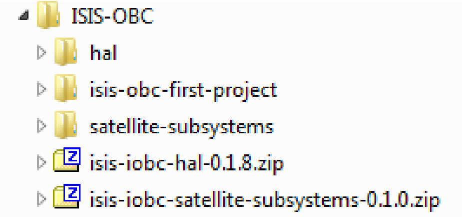
Figure 15: Relative file locations. The library directory should be in the same directory as the project.

Library updates are simple: just replace the existing library directory with the new version of the library. The projects will automatically use the new libraries during the next build.

6.2.3 Linking an Additional Library

Several libraries are available through ISIS. To use an additional library the path to the library directory should be set up properly. This involves setting include paths and adding the library name in Eclipse project properties. In this example the satellite-subsystems library will be added.

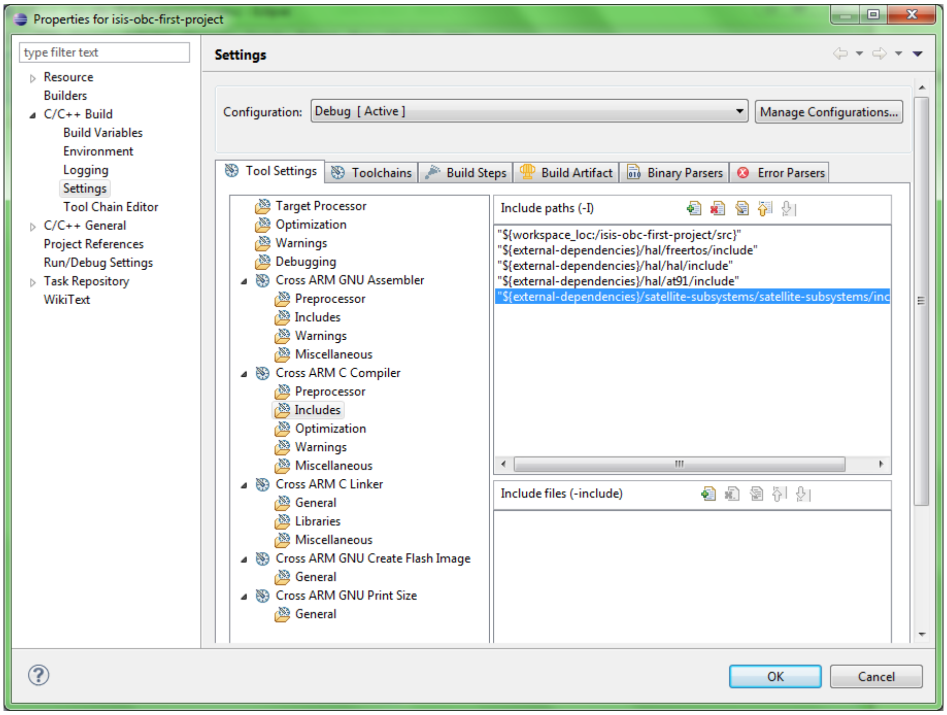
1. First the new library is extracted to the directory that also includes the project. The directory is shown in Figure 16.



ISIS-OBC
hal
isis-obc-first-project
satellite-subsystems
isis-iobc-hal-0.1.8.zip
isis-iobc-satellite-subsystems-0.1.0.zip

Figure 16: Satellite-subsystems library added to the directory.

2. Add the include path to the library: Open Eclipse, right click the first project and open project properties. Navigate to *C/C++ Build > Settings*. Under Tool Settings choose *Cross ARM C Compiler > Includes*. Add the path to the *include* folder of the library. For satellite subsystems this is "${external-dependencies}/satellite-subsystems/satellite-subsystems/include". See Figure 17.



Configuration: Debug [Active]

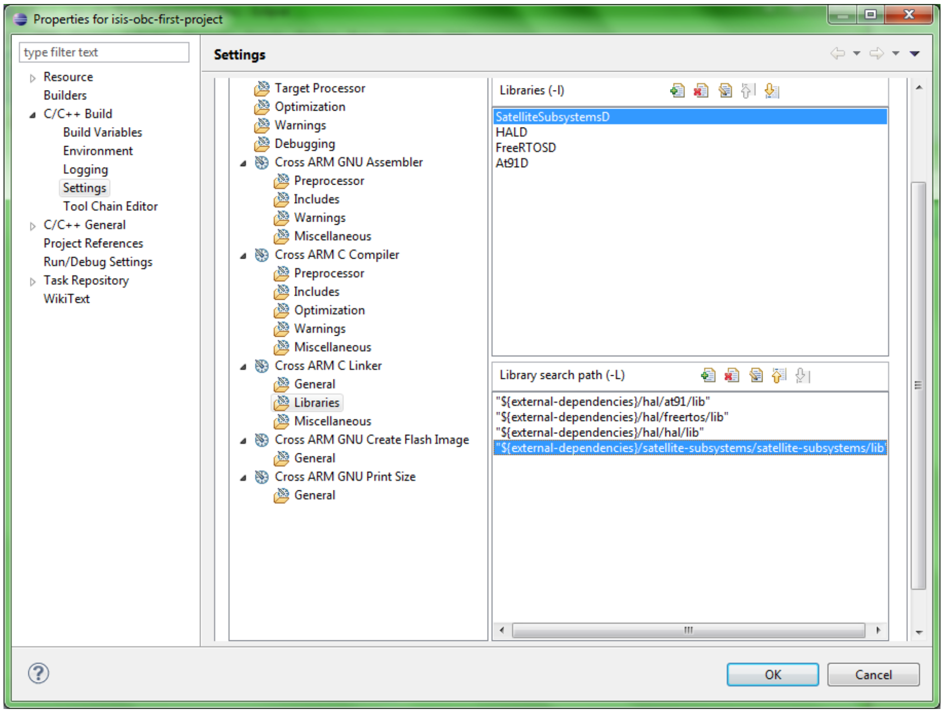
Include paths (-I)

```
"${workspace_loc:/isis-obc-first-project/src}"
"${external-dependencies}/hal/freertos/include"
"${external-dependencies}/hal/hal/include"
"${external-dependencies}/hal/at91/include"
"${external-dependencies}/satellite-subsystems/satellite-subsystems/inc
```

Include files (-include)

Figure 17: Adding the library include path

3. Add the library to the linker settings: Select *Cross ARM C Linker > Libraries*. Add the library name with a capital D at the end to the *Libraries* list. Add the path to the *lib* directory to the *Library search path* list. See Figure 18.



Libraries (-l)

```
SatelliteSubsystemsD
HALD
FreeRTOSD
At91D
```

Library search path (-L)

```
"${external-dependencies}/hal/at91/lib"
"${external-dependencies}/hal/freertos/lib"
"${external-dependencies}/hal/hal/lib"
"${external-dependencies}/satellite-subsystems/satellite-subsystems/lib"
```

Figure 18: Add the library to the linker settings

Click ok. The library can now be used from within the project.

6.3 Eclipse Build Configuration

Eclipse provides so called ‘Build Configurations’. The active build configuration is used when code is build. The standard configurations provided are Debug and Release.

6.3.1 Debug Build

A Debug Build is a non-optimal compilation of the embedded software to facilitate efficient debugging. The IDE instructs the compiler to not use code optimization and to add debug symbols within the compiled program. This allows the debugging service to perform its duty of providing the developer with program state information. The ISIS-OBC code also adds additional code that allows debug messages to be emitted. The Debug Configuration is therefore larger and slower than necessary for unattended execution of the program.

6.3.2 Release Build

A Release Build is an optimized compilation of the embedded software targeted at unattended/autonomous execution of the program. This configuration should be used once the code performs to satisfaction using the Debug Build, and the program is to be officially put through verification test procedures.

Note: Due to code optimizations applied by the compiler and excluding debug related functions from the build, the resulting machine code can be quite different. The functional aspect should remain the same, however in some particular cases differences might still occur. It is therefore very important to use the release build for verification purposes, to be able to catch any problems arising from release build optimizations.

6.3.3 Changing the Active Build Configuration

The current Build Configuration used by Build can be set through the context menu by right clicking the project and choosing ‘*Build Configurations*’ as shown in Figure 19.

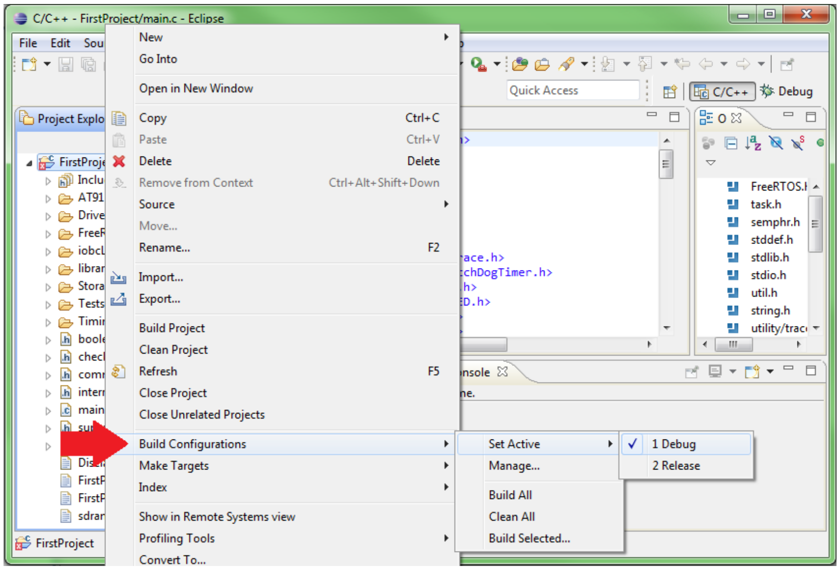


Figure 19: Setting the build configuration

7 Running your Program

After a program has been successfully build the code can be transferred to the embedded processor and executed on the embedded hardware. Setup the hardware as described in Chapter 4.

For debugging purposes the programmer/debugger hardware and software takes care of setting up the embedded processor so it will run the code build within the IDE. It provides functionality to pause code execution, using a so called breakpoint, and read-out memory on the embedded processor. The code itself is directly copied to the volatile memory of the processor, such that the write-cycles of the non-volatile flash based program memory remain intact.

For release purposes the build code will be stored in the non-volatile flash based program memory using a dedicated program called sam-ba. The ISIS-OBC controller will then able to execute the stored program autonomously and independent from the development machine.

In both cases the USB-to-Serial data interface is available to transfer commands and data between the ISIS-OBC processor and a computer which has at least the FTDI driver installed.

7.1 Debugging your Code

To perform code debugging at least the programmer/debugger must be connected. Generally also the USB-to-serial connection is used in parallel that is used to emit debug and program state messages. The debugging services are provided by the widely used GNU DeBugger (GDB).

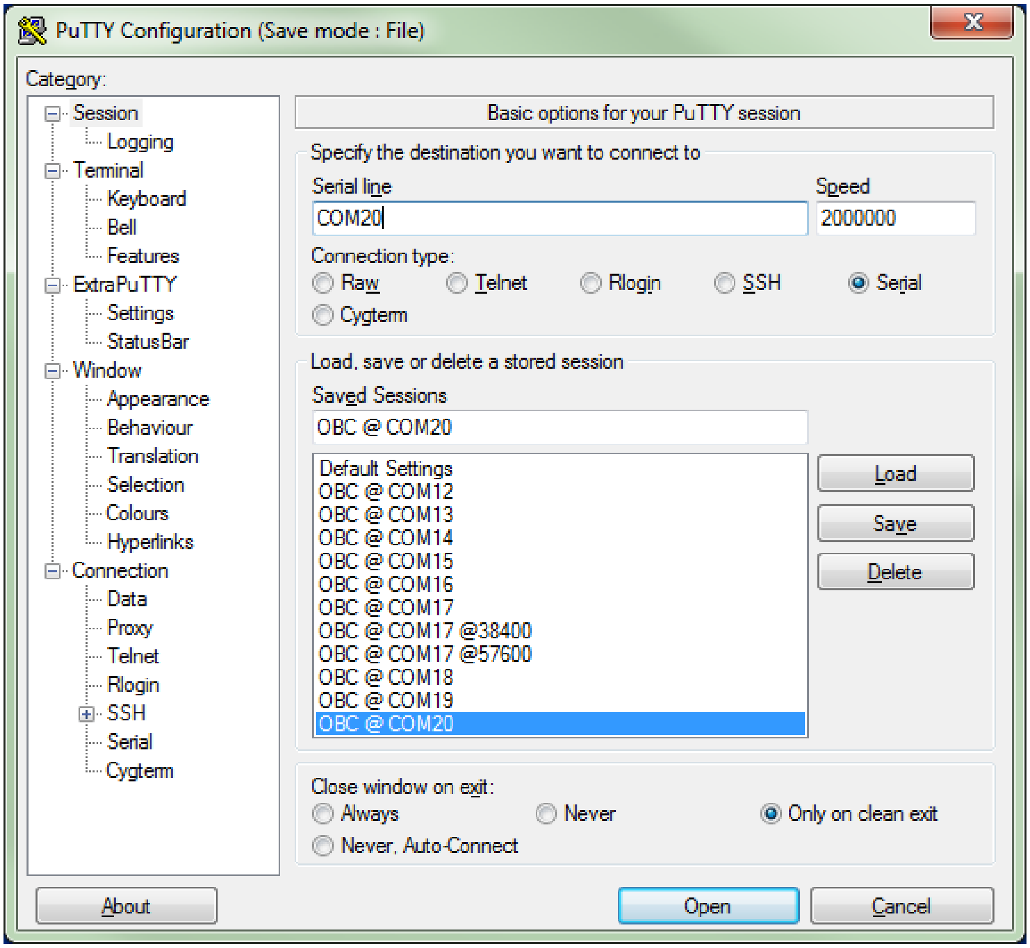
7.1.1 Start the GDB Server

To start the GDB server, click on the small down pointing arrow next to the External Tools icon and then on Jlink-samg20-GDB. This launch item has been configured in chapter 5.

7.1.2 Start PuTTY for the USB-to-Serial Interconnect

For usage of the serial interface the PuTTY application is provided.

- Start the putty executable found via Start > All Programs > ISIS-OBC SDK > PuTTY. A window similar to Figure 20 is shown. Select Serial and set the speed to 2000000 (2MHz).
- Set the COM port that has been assigned to the FTDI USB-to-Serial device as COM##, where the ## is the com port number used. This can be found in the windows device manager. Go to start and right click Computer. Select properties. Select device manager and expand to the Ports section. The device is USB Serial Port (COM##). See Figure 21.
- Optionally provide a name in the ‘saved sessions’ textbox and click ‘Save’ to save the configuration. This can be loaded later by clicking the name and then clicking ‘Load’.
- Click Open.



Specify the destination you want to connect to

Serial line: COM20

Speed: 2000000

Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial ○ Cygterm

Load, save or delete a stored session

Saved Sessions: OBC @ COM20

Default Settings
OBC @ COM12
OBC @ COM13
OBC @ COM14
OBC @ COM15
OBC @ COM16
OBC @ COM17
OBC @ COM17 @38400
OBC @ COM17 @57600
OBC @ COM18
OBC @ COM19
OBC @ COM20

Close window on exit: ○ Always ○ Never ● Only on clean exit ○ Never, Auto-Connect

Figure 20: Putty configuration window

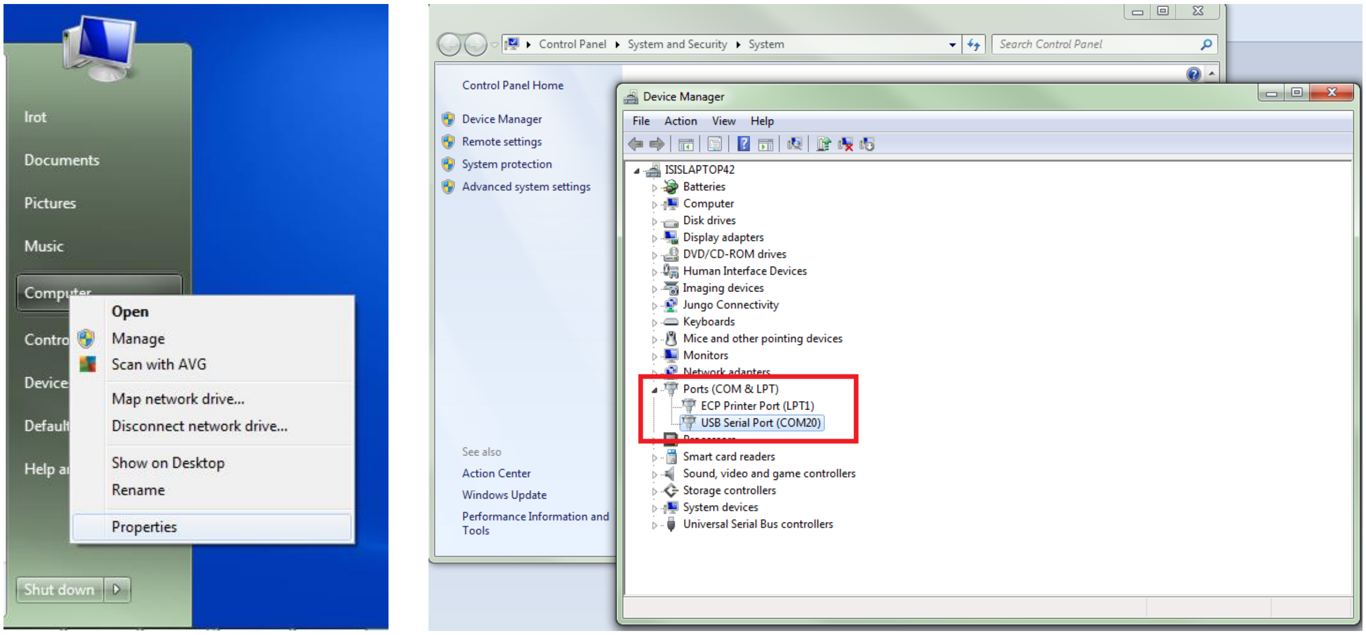


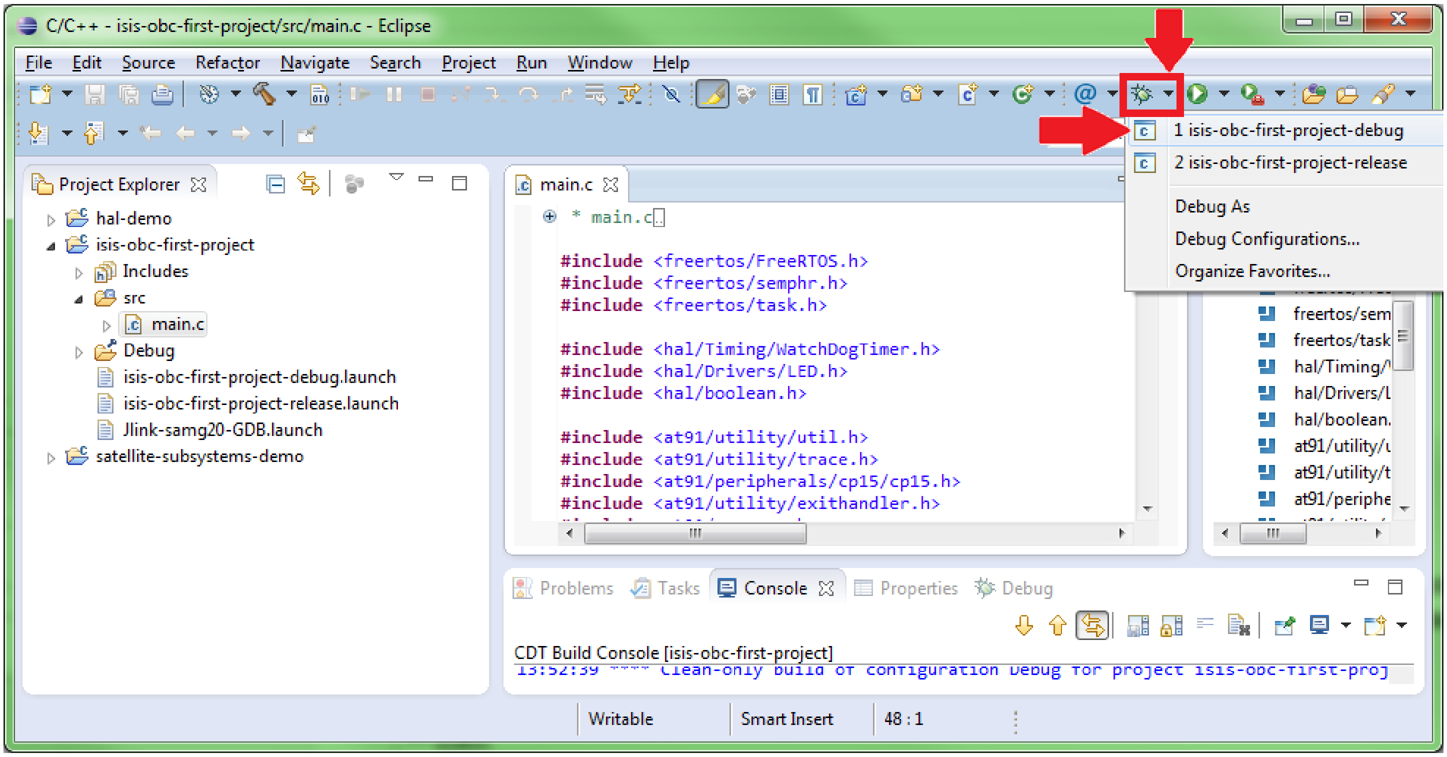
Figure 21: Retrieve the COM port assigned to the USB-to-Serial from the device manager

7.1.3 Starting a Debug Session

Launch the Debug of your project by clicking on the small down pointing arrow next to the debug icon. Then click the name of the debug configuration. See Figure 22.

Optionally the demo project for a library (e.g. hal-demo) can be run in debug or release mode. To do this first select the project and then select the debug/release launch. This will launch demo/test applications for that particular library.

IMPORTANT NOTICE: the demo/test code serves as simple examples of how to use the library functionality but is by no means guaranteed to have high quality/flight quality nor uses all available functionality. The demo/test code should only be used as a first directional hint when starting to use a new part of the library. Always refer to the library documentation for a full explanation and overview of all available functions, and carefully structure the code according to your preferred coding style. It is highly discouraged to directly copy/paste blocks of code from the demo/test software during development.



```
* main.c

#include <freertos/FreeRTOS.h>
#include <freertos/semphr.h>
#include <freertos/task.h>

#include <hal/Timing/WatchDogTimer.h>
#include <hal/Drivers/LED.h>
#include <hal/boolean.h>

#include <at91/utility/util.h>
#include <at91/utility/trace.h>
#include <at91/peripherals/cp15/cp15.h>
#include <at91/utility/exithandler.h>
```

```
CDT Build Console [isis-obc-first-project]
13:52:39 **** Clean-only build of configuration Debug for project isis-obc-first-proj
```

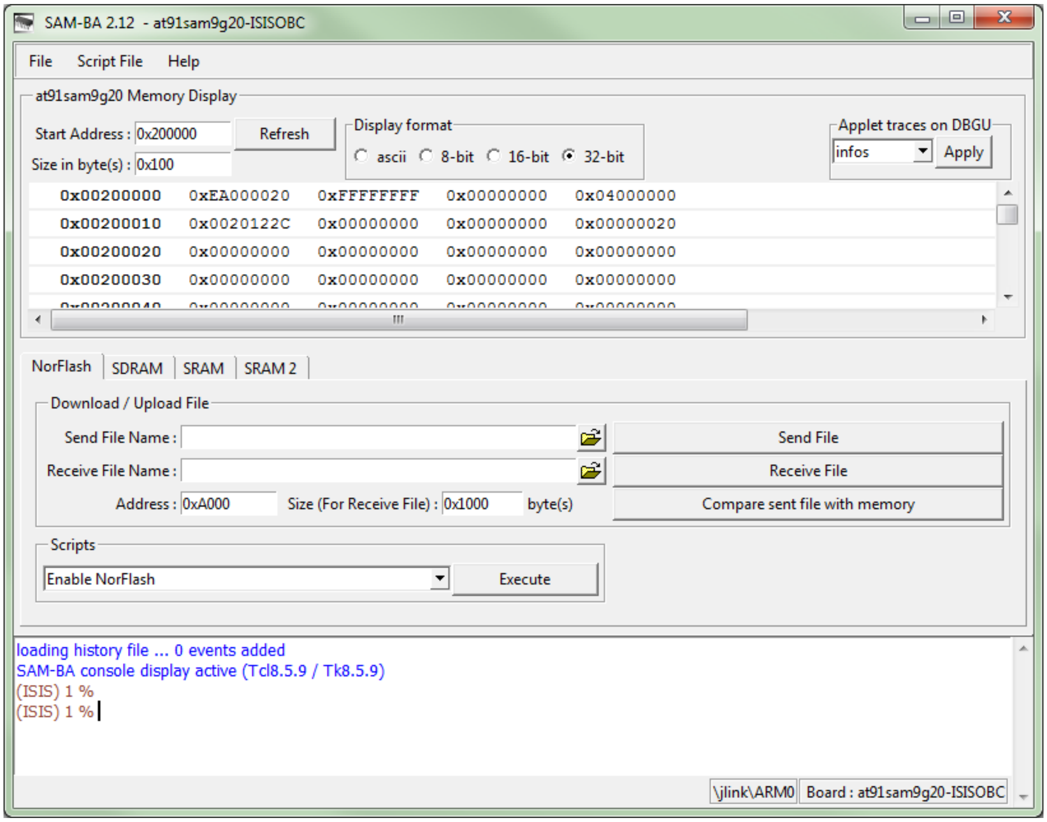
Figure 22: Start Debug session

7.1.4 Debugging the Application

Now you can control the execution of your code by setting breakpoints; clicking play and pause. You can also inspect local variables and add watchpoints for specific variables. You can also see the debug output of your program (such as printf's) using PuTTY.

8 Flashing Your Code

Once you have debugged your code, you can save the code into the Flash memory of the OBC. This is done using the SAM-BA program that is started through Start > All Programs > ISIS-OBC SDK > Atmel SAM-BA.



SAM-BA 2.12 - at91sam9g20-ISISOBC

at91sam9g20 Memory Display

Start Address : 0x200000

Size in byte(s) : 0x100

Display format: ○ ascii ○ 8-bit ○ 16-bit ● 32-bit

Applet traces on DBGU: infos

```
0x00200000   0xEA000020   0xFFFFFFFF   0x00000000   0x04000000
0x00200010   0x0020122C   0x00000000   0x00000000   0x00000020
0x00200020   0x00000000   0x00000000   0x00000000   0x00000000
0x00200030   0x00000000   0x00000000   0x00000000   0x00000000
0x00200040   0x00000000   0x00000000   0x00000000   0x00000000
```

Download / Upload File

Send File Name :

Receive File Name :

Address : 0xA000

Size (For Receive File) : 0x1000 byte(s)

Scripts: Enable NorFlash

```
loading history file ... 0 events added
SAM-BA console display active (Tcl8.5.9 / Tk8.5.9)
(ISIS) 1 %
(ISIS) 1 %
```

\jlink\ARM0 Board : at91sam9g20-ISISOBC

- Make sure you have set up the hardware as indicated earlier.
- Make sure you have closed any running debug session within Eclipse
- Launch SAM-BA
- Select the *at91sam9g20-ISISOBC* board under boards and click "Connect"
- Under the NOR-Flash tab of SAM-BA, click on "Execute" to enable NOR-Flash.
- Browse for a file using the folder icon next to the "Send File" button.
- The Address field is set to 0xA000 which is the default value where the application code should be flashed to. The bootloader occupies address range 0x0000 through 0xA000 and is write-protected when using the *at91sam9g20-ISISOBC* board selection. See Section 8.1 if the bootloader memory needs to be updated.
- Click "Send File" and wait for your program to upload.
- Click "Compare sent file with memory", SAM-BA will read back the contents of the flash and verify that the firmware is flashed correctly. If the process was successful, it will display a message indicating that the sent file and memory match exactly.
- Close SAM-BA, remove power from the OBC and power it up again.
- Your code will now be running on the OBC. You can see the output of your program (such as printf's) using PuTTY.

Note: It is advised to use "Compare sent file with memory" to verify that your firmware was uploaded correctly.

8.1 ISIS-OBC Bootloader

The NOR-Flash of an ISIS-OBC contains a bootloader memory space that starts at address 0 and extends until address 0xA000. The bootloader prepares the board so programs stored from 0xA000 onwards are loaded into SDRAM and started. You should not overwrite the bootloader memory area under regular circumstances. However the bootloader memory can be written to by selecting the *at91sam9g20-ISISOBC-UpdateBootloader* board when starting SAM-BA.

To update the bootloader memory area:

- Launch SAM-BA
- Select the *at91sam9g20-ISISOBC-UpdateBootloader* board under boards and click "Connect"
- Follow the steps for flashing code to NOR-Flash

NOTE: A backup of the standard bootloader binary is provided along with the SDK installer and can be used to update the board when the original bootloader flash memory area is accidently overwritten.

8.2 Backing-Up and Restoring Flashed Code

It is possible to backup code flashed to the ISIS-OBC hardware. Make sure you have set up the hardware as indicated earlier.

- Launch SAM-BA and click "Connect".
- Under the NOR-Flash tab of SAM-BA, click on "Execute" to enable NOR-Flash.
- (option 1) Backup of program excluding bootloader:
  - Enter "0xA000" next to “Address”.
  - Enter "0xF6000" next to "Size (For Receive File)".
- (option 2) Backup of program including bootloader:
  - Enter "0x0" next to “Address”.
  - Enter "0x100000" next to "Size (For Receive File)".
- Browse to the location where you want to save the backup using the folder icon next to the "Receive File" button.
- Click "Receive File" and wait for the operation to finish.

The file you specified now contains a backup of the contents of the NOR-Flash of the OBC.

To restore backups follow the procedure of flashing code described in Section 8.1 for backups taken from start address 0xA000 (option 1), and Section 8.2 for backups taken from start address 0x0 (option 2).

9 Support

In case of issues, please contact ISIS (support@isispace.nl).