SE*cube*™ Development Kit

Getting Started

Release: July the 23th, 2018


HW
SW
Open
Source

Proprietary Notice

The following document offers information, which is subject to the terms and conditions described hereafter.

While care has been taken in preparing this document, some typographical errors, error or omissions may have occurred. We reserve the right to make changes to the content and information described herein or update such information at any time without notice. The opin-ion expressed are in good faith and while every care has been taken in preparing this docu-ment, some typographical errors, error or omissions may have occurred. We reserve the right to make changes to the content and information described herein or update such information at any time without notice. The opinion expressed are in good faith and while every care has been taken in preparing this document.

Authors

Giuseppe AIRÒ FARULLA *(CINI Cyber Security National Lab)* giuseppe.airofarulla@polito.it
Alberto CARELLI *(CINI Cyber Security National Lab)* alberto.carelli@polito.it
Paolo PRINETTO *(President, CINI)* paolo.prinetto@polito.it
Giorgia SOMMA *(Business Development Manager, Blu5 Labs Ltd)* *giorgia.somma@blu5labs.eu*
Antonio VARRIALE *(Managing Director, Blu5 Labs Ltd)* av@blu5labs.eu

Acknowledgment

Authors would like to thank the following persons for their valuable support:
Massimo CASALEGNO
Nicola FERRI
Frederik GOSSEN
Pascal TROTTA

The present work has been partially supported by CISCO and developed within the Project "FilieraSicura: Securing the Supply Chain of Domestic Critical Infrastructures from Cyber Attacks".

Trademarks

Disclaimer

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS AND ITS AUTHORS DISCLAIM ALL WARRANTIES, EXPRESS, OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY TAHT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PURPOSE.

THE SOFTWARE IS PROVIDED TO YOU "AS IS" AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUD-ING, WITHOUT LIMITATION, LOSS REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSI-NESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEREUNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY LIKELIHOOD OF SUCH DAMAGES

Table of content

1. An Introduction to SE*cube*™ *Open Security Platform*

The **SE*cube*™** (Secure Environment cube) *Open Security Platform* is an open source security-oriented hardware and software platform, designed and constructed with ease of integration and service-orientation in mind.

The hardware part of the platform was originally designed by Blu5 Group[1], whereas the software libraries stem from a strong cooperation among five international research institutions, including:

Blu5 Labs Ltd, Blu5 Group, Ta Xbiex, Malta –
Reference: Antonio VARRIALE, av@blu5labs.eu
CINI Cyber Security National Lab, Torino, Italy –
Reference: Paolo PRINETTO, paolo.prinetto@polito.it
Lero, The Irish Software Research Centre, University of Limerick, Limerick, Ireland –
Reference: Tiziana MARGARIA, tiziana.margaria@lero.ie
LIRMM, CNRS, Montpellier, France –
Reference: Giorgio DI NATALE, giorgio.dinatale@lirmm.fr
TU Dortmund, Dortmund, Germany –
Reference: Bernard STEFFEN, bernhard.steffen@tu-do.de

The hardware side of the platform relies on a complete chain of devices, from *chip* to *USB stick,* and is presented in section 2.

The components needed to set up a system and its assembling steps are introduced in sections 3 and 4, respectively.

The software libraries, in conjunction with the above-mentioned design environment, allow developers who are not willing or able to produce the security APIs and protocols themselves to exploit the ready functions provided (currently as APIs and in the near future as services) within the **SE*cube*™** platform and experience the platform as a high-security black box. Con-versely, security experts can enjoy the openness and good documentation to verify, change or rewrite the pre-existing software, starting from basic low-level blocks or even redefine en-tirely the whole system.

All the software is released in source code under GPLv3 license[2].

The components needed to set up a system and its assembling steps are introduced in sections 3 and 4, respectively.

Leveraging the platform thought, we intend to create and nurture over time a community for developers at the different levels of security competence and in different application domains. This will ease sharing project, knowledge, and resource and provide the collectivity of members with specialized support tailored to their needs.

1 www.blu5group.com

2 https://www.gnu.org/licenses/gpl-3.0.en.html

2. The SE*cube*™ Hardware device family

The **SE***cube*™ Hardware device family comprises 3 major members:

The *Chip*, named **SE***cube*™ Chip, or simply **SE***cube*™ (section 2.1)
The *Development Board*, named **SE***cube*™ DevKit (section 2.2)
The USB Stick, named **USE***cube*™ Stick (section 2.3).

2.1. The SE*cube*™ Chip

The core of the **SE***cube*™ Hardware device family is a 3D multi-module SoC (System-on-Chip) (a detail is in Figure 21), integrated in a 9mm x 9mm BGA package (Figure 22). The full **SE***cube*™ Data Sheet is available in Appendix A.

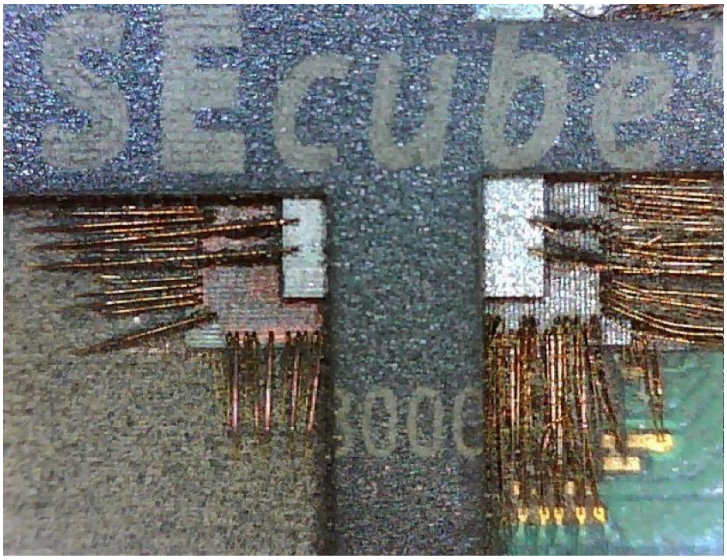


Figure 21 - Detail of the die of the **SE***cube*™ Chip

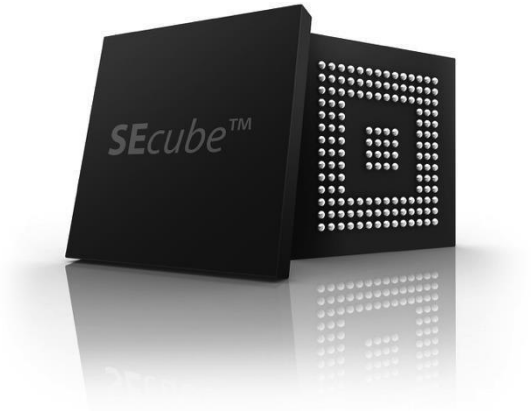


Figure 22 The **SE***cube*™ Chip

The single chip embeds three hardware components: a powerful processor, a flexible FPGA, and an EAL5+ certified smart card (Figure 23).

2.1.1. The Processor

The processor adopted within the **SE*cube*™** is the STM32F4: a high-performance ARM Cortex M4 RISC CPU, produced by ST Microelectronics[3]. It provides the following features:

- 2 MiB[4] of Flash memory
- 256 KiB[5] of SRAM
- 32 bit parallelism
- Operating frequency of 180 MHz Low power consumption

This CPU has been selected among many ARM based micro-controllers, since it offers several features that make it suitable for high-performance and security-oriented solutions. For example, it supports the Cortex CMSIS implementation that provides, among the others, the CMSIS-DSP libraries: a collection with over 60 DSP functions for various data types. The CMSIS-DSP library allows developers to implement complex, real time operations using the embed-ded hardware Floating Point Unit.

In addition, the CPU provides several peripherals such as SPI, UART, USB2.0 and SD/MMC, which ease the hardware integration in diverse devices. For example, a secure USB device can be easily realized using the USB2.0 and the SD card interfaces, respectively.

On the security side, a TRNG (True Random Noise Generator) embedded unit, hardware mech-anisms like MPU (Memory Protection Unit), and privileged execution modes allow implement-ing the security strategies required by a certified secure controller (e.g., privileged memory areas, key generation, etc.).

For programming, debug, and testing operations, the CPU provides a standard JTAG interface that can be permanently disabled once the development cycle is over, protecting all the sen-sitive information through a physical hardware lock.

[3] http://www.st.com/en/microcontrollers/stm32f4-series.html?querycriteria=productId=SS1577

[4] For the purpose of this document 1 MiB = 2^{20} Bytes

[5] For the purpose of this document 1 KiB = 2^{10} Bytes

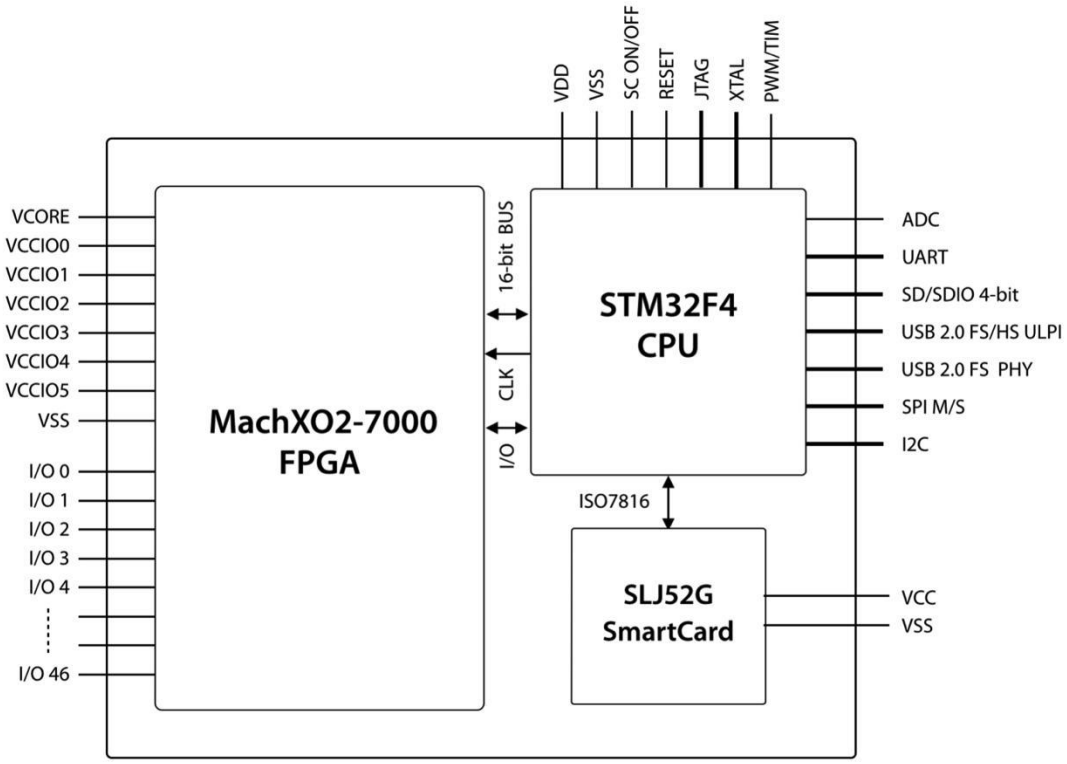


Figure 23 **SE*cube*™** *Hardware Architecture*

2.1.2. The FPGA

The FPGA element, a Lattice MachXO2-7000 device[6], is based on a fast, non-volatile logic array providing the following main features:

- 7,000 LUTs
- 240 Kib[7] embedded block RAM
- 256 Kib user flash memory
- Ultra low-power device

The FPGA exposes 47 general purpose I/O which may be used as a 32-bit external bus able to transfer data at 3.2 Gib/s.

As outlined in Figure 23, within the **SE*cube*™ Chip** the FPGA is connected to the CPU through a 16-bit internal BUS, providing a data transfer rate of up to 1.6 Gib/s.

A CPU-FPGA clock line is provided to simplify the clock domains synchronization.

To limit the number of pins and the BGA package size, the FPGA JTAG is connected just to the embedded CPU, which manages both the debug and the programming operations. Therefore, the FPGA configuration can be implemented by means of customized, high-security tech-

[6] Lattice Semiconductor, www.latticesemi.com - "MachXO2™ Family Data Sheet – DS1035

[7] For the purpose of this document 1 Kib = 2^{10} bits

niques. For example, the programming stream can be encrypted and signed through dedicated algorithms. The CPU and/or the smartcard elements can then be used to decrypt and verify it before its injection into the FPGA.

2.1.3. The Smartcard

The third component of the **SE*cube*™ Chip** is an EAL5+ certified security controller, hereafter named smartcard[8], based on a secure chip produced by Infineon, that provides the following features:

ISO7816 interface

JavaCard Platform, Global Platform
2.2 128 KiB Flash

EC, ECDH up to 521 bit (HW accelerator)
RSA up to 2 Kib (HW accelerator)

AES128/192/256 (HW accelerator)

As outlined in Figure 23, within the **SE*cube*™ Chip**, the smartcard is connected to the CPU via a standard ISO7816 interface.

The smartcard does not expose any interface outside the chip. This architectural solution pro-vides high-grade and certified security functionalities behind a simpler and easy-to-use appli-cation interface.

8 Infineon Chip Card & Security ICs Portfolio -

2.2. The SE*cube*™ DevKit

The **SE*cube*™ DevKit** is an open development board (Figure 24) designed to support developers to integrate the **SE*cube*™ Chip** in their hardware and software projects.

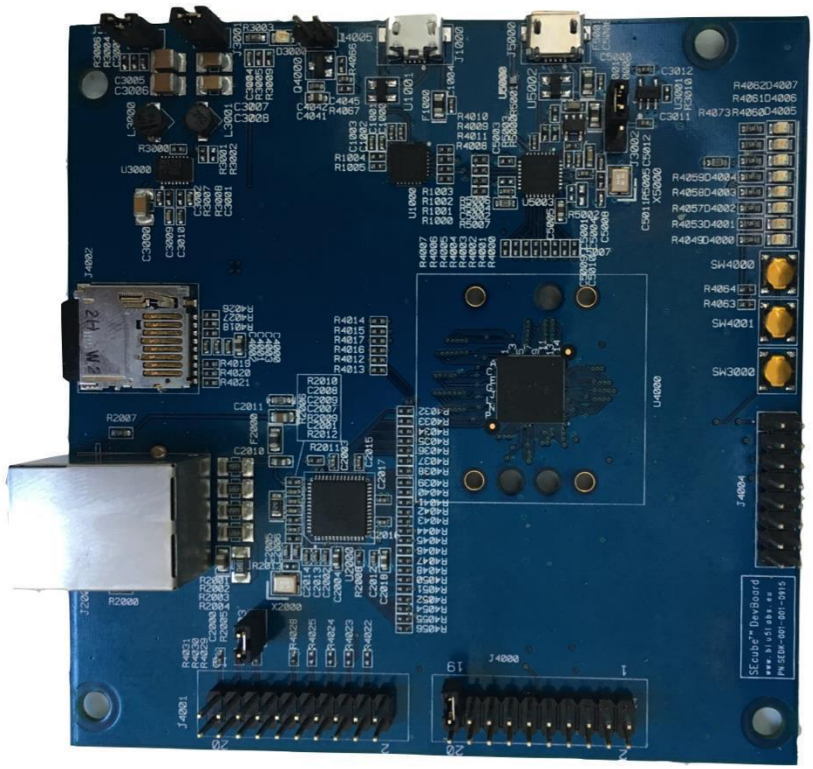


Figure 24 The **SE*cube*™ DevKit** board

The board, whose schematic is in Figure 24, is equipped with several interfaces and peripherals, including:

- USB 2.0 High Speed (J5000)
- USB 2.0 to UART (J1000)
- microSD card (J4002)
- Ethernet 10/100 socket (J2000)
- Switches and Led (SW4000, SW4001, SW3000, LED0, …)
- SEcube™ embedded FPGA and CPU GPIOs (J4004,
- J4000) SEcube embedded CPU JTAG (J4001)

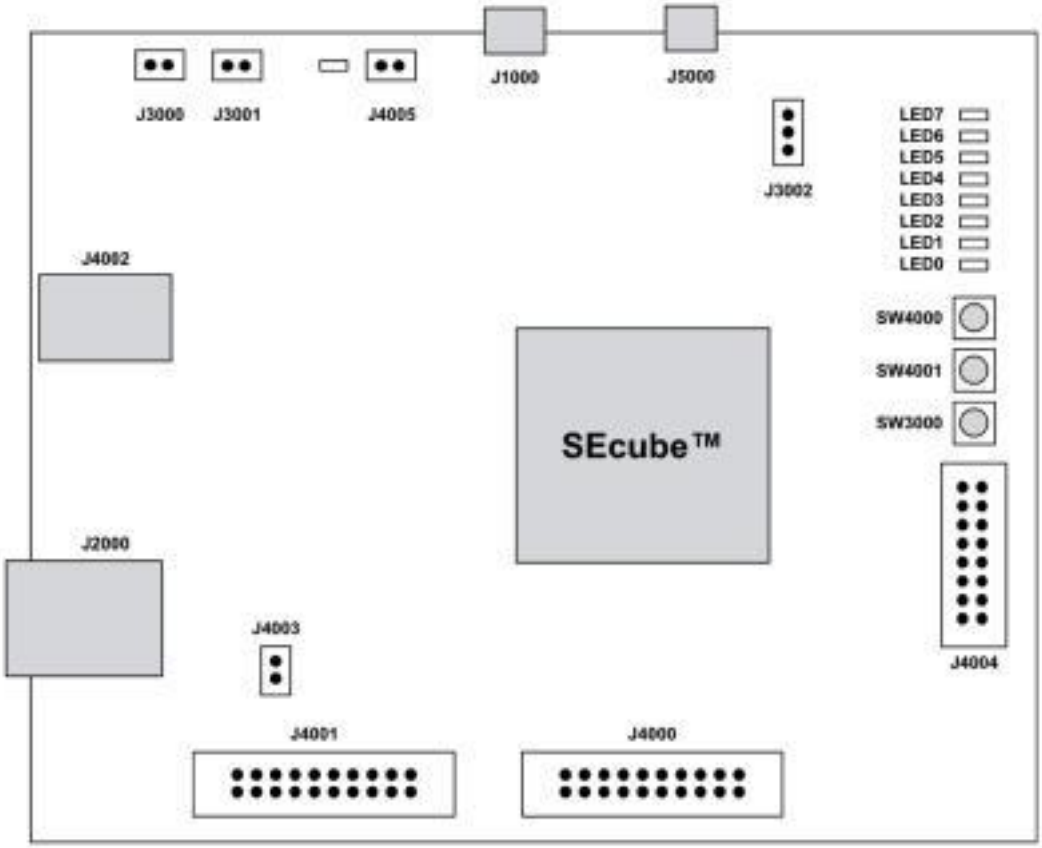


Figure 25 - **SE*cube*™ DevKit**

The board is directly powered by one of the 2 micro USB connectors. The jumper 3002 selects the connector to be used to power the board (pins 1-2 select J5000, pins 2-3 select J1000). The **SE*cube*™** DevKit allows connecting two power supply lines and measuring the related power consumption, through the following jumpers:

J3000: 1V2 power supply line

J3001: 3V3 power supply line

The power supply of the smartcard embedded into **SE*cube*™** is controlled by a dedicated pin. Nevertheless, the jumper J4005 allows to bypass this control and power the embedded smart-card permanently.

The jumper J4003 allows a direct control of the **SE*cube*™** reset pin via the JTAG interface.

2.3. The USE*cube*™ Stick

As shown in Figure 26, the **USE*cube*™** Stick is an USB form factor based on the **SE*cube*™** Chip, which proves to be an amazing way to deploy the **SE*cube*™** functionalities through a USB 2.0 High-Speed interface.

Figure 26 The **USE*cube*™** Stick

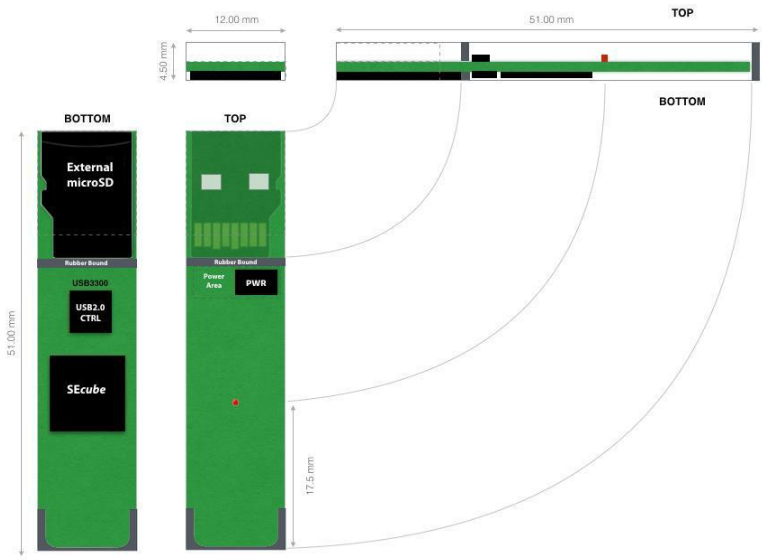


Figure 27 **USE*cube*™** Stick Internal structural details

From the hardware point of view, the **USE*cube*™** Stick is designed as part of the **SE*cube*™** Dev-Kit. This allows the developers to migrate in a very fast way from the development board to the Stick and be ready for a market deployment.

Figure 28 The **USE***cube*™ Stick plugged in a laptop

The **USE***cube*™ Stick is compatible with any Operating System and the **SE***cube™* functionalities are easily exposed to applications and services without installing any driver.

Since the **USE***cube*™ Stick storage capability is based on a microSD card, both the size and the speed can be tuned per the user requirement and can be changed at any time, just replacing the microSD, without buying a new **USE***cube*™ Stick.

The microSD card socket is embedded in the USB connector. As shown in Figure 27, this solu-tion allows to save space making the **USE***cube*™ Stick very compact and, at the same time dust and water-resistant.

Since the **USE***cube*™ Stick is not provided with the JTAG interface, to inject the firmware previously developed and tested on the **SE***cube*™ DevKit, all the devices come with an embedded secure boot loader.

More details on the **USE***cube*™ Stick firmware injection can be found in Session 10.

3. The SE*cube*™ System Setup

In this Section, we outline the set of both hardware and software resources you need to set up the **SE*cube*™** DevKit**.**

At the end of this Section you will have acquired a clear overview of the prerequisites to set up the environment.

3.1. Hardware resources

The following hardware resources are needed (detailed in the following Paragraphs):

1. PC
2. STM SDK
3. The **SE*cube*™** DevKit
4. The **USE*cube*™** Stick

3.1.1. PC

You do not need a particularly new or powerful PC to get started with the **SE*cube*™** DevKit. Minimal requirements include:

2+ GiB[9] of RAM

10+ GiB of empty/available space on
HDD 2 USB ports

3.1.2. STM SDK

To program the STM32F4 processor available on the **SE*cube*™** DevKit you can follow two alternatives, resorting to:

an in-circuit programmer and debugger, and particularly to the *ST-Link/v2*[10],

one board such as the *STM Discovery* or *STM Nucleo*, equipped with a ST-Link/v2 pro-grammer,

respectively.

3.1.3. ST-Link/v2

Description

The ST-LINK/V2 is an in-circuit debugger and programmer for the STM8 and STM32 microcontroller families. Its JTAG/serial wire debugging-programming (SWD) interface is used to communicate with the STM32 microcontroller comprised within the **SE*cube*™** DevKit.

This programmer requires 5V power supplied by a standard USB connector (A to Mini B cable) compatible with the USB 2.0 interface.

9 For the purpose of this document 1 GiB = 2^{30} Bytes

10 http://www.st.com/en/development-tools/st-link-v2.html

How to get it

We suggest to get the programmer through RS[11], at a price of 19.19 €.

What you should have got

Your purchase should comprise the following items (Figure 31):

- The St-Link/v2 programmer
- USB 2.0 A to Mini B cable
- JTAG to SWD cable
- SWIM cable (not needed to program the **SE*cube*™** DevKit)

Figure 31 – Components purchased with the ST-Link/v2 programmer

3.1.4. ST Discovery or ST Nucleo board

Description

The *ST Discovery* and *ST Nucleo* boards represent an affordable and flexible way for users to build project with a microcontroller from the STM32 family, choosing from the various combinations of performance, power consumption and features.

These boards do not require any separate probe as they both integrate a ST-Link/V2 program-mer/debugger.

The STM32 Nucleo board comes with the STM32 comprehensive software HAL library to-gether with various packaged software examples, as well as direct access to online resources.

11 http://it.rs-online.com/web/p/kit-di-sviluppo-per-processori-e-microcontrollori/7141701/?sra=pmpn

How to get it

We suggest to get the boards through RS. It is important to clarify that you do not need to buy them both: you can buy only one board, and your purchase will in any case represent a valid alternative to the ST-Link/v2 programmer.

The recommended Discovery[12] and Nucleo[13] boards can both be bought through RS.

What you should have got

In both cases, you should get the board with a USB 2.0 A to Mini B cable.

3.1.5. The SE*cube*™ DevKit

Description

See section 2.2.

How to get it

The **SE*cube*™** DevKit can be ordered online[14].

What you should have got

Your purchase should comprise the following items, depicted in Figure 32:

The **SE*cube*™** DevKit; MicroSD;
USB 2.0 A to Mini- B cable

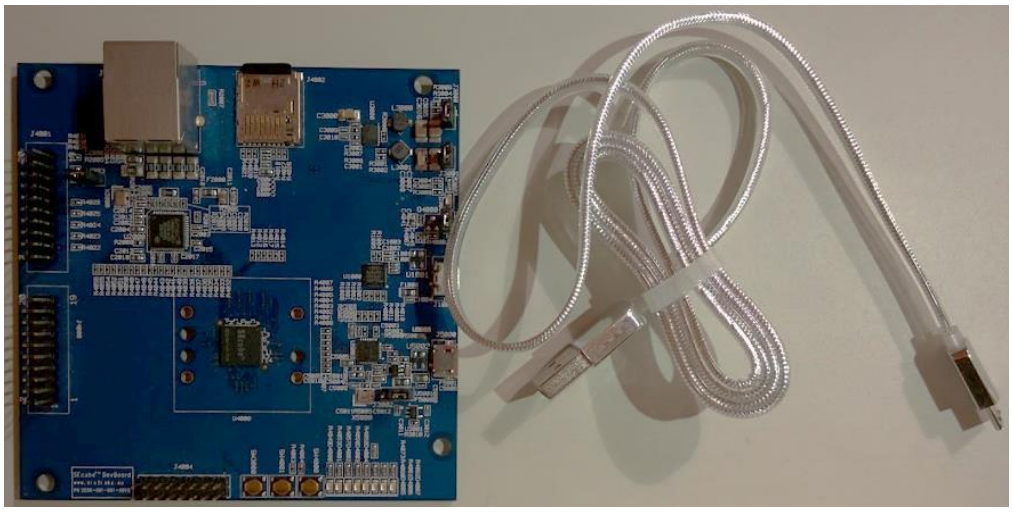


Figure 32 - Components purchased with the SEcube™ DevKit

12 http://it.rs-online.com/web/p/kit-di-sviluppo-per-processori-e-microcontrollori/9107951/

13 http://it.rs-online.com/web/p/kit-di-sviluppo-per-processori-e-microcontrollori/8029425/

14 http://www.secube.eu

3.1.6. The USE*cube*™ Stick

Description

See section 2.3

How to get it

The **USE***cube*™ Stick can be ordered online[15].

What you should have got

Your purchase should comprise various items, depending on which purchase option you choose.

3.2. Software resources

You need the following tools:

1. Operating System
2. Java Runtime Environment
3. Eclipse
4. AC6 Tools: GNU ARM Embedded Toolchain
5. STM32CubeMX - STM32Cube initialization code generator
6. Lattice Diamond Software
7. ST-Link/v2 drivers
8. ST-Link Utility
9. Open Source SDK

3.2.1. Operating System

Two Operating Systems are currently supported:

Windows 7 (or later)[16]

Linux with Kernel 2.6 (or later)[17]

15 http://www.secube.eu

16 This procedure has been tested with Windows 7 Professional x64

17 This procedure has been tested with both Linux Chakra kernel 4.5.7-1 x64 and Linux Ubuntu 14.04LTS x64

3.2.2. Java Runtime Environment

Description

The Java Runtime Environment (JRE) is a software package that contains what is required to run a Java program. It includes a Java Virtual Machine implementation together with an implementation of the Java Class Library. The Oracle Corporation, which owns the Java trade-mark, distributes a Java Runtime environment with their Java Virtual Machine called HotSpot.

Version required

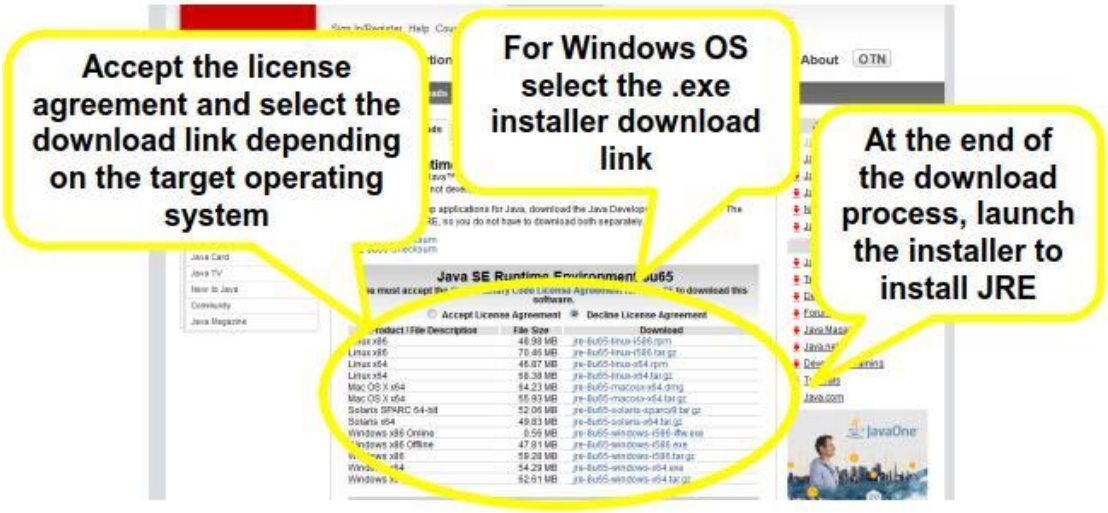
Version 8u111 (or later)

How to get it

The program is available free of charge from the Oracle website[18].

Installation hints

Visit the download link and follow the instructions as in the following screenshot:



What it's going to be used for

The Java Runtime Environment is required for Eclipse to work properly.

3.2.3. Eclipse

Description

Eclipse is of the most widely used free and open-source integrated development environment (IDE) in computer programming.

It contains a base workspace and an extensible plug-in system for customizing the environment.

18 http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may be used to develop applications in other programming languages as well, resorting to dedicated plugins.

Version required

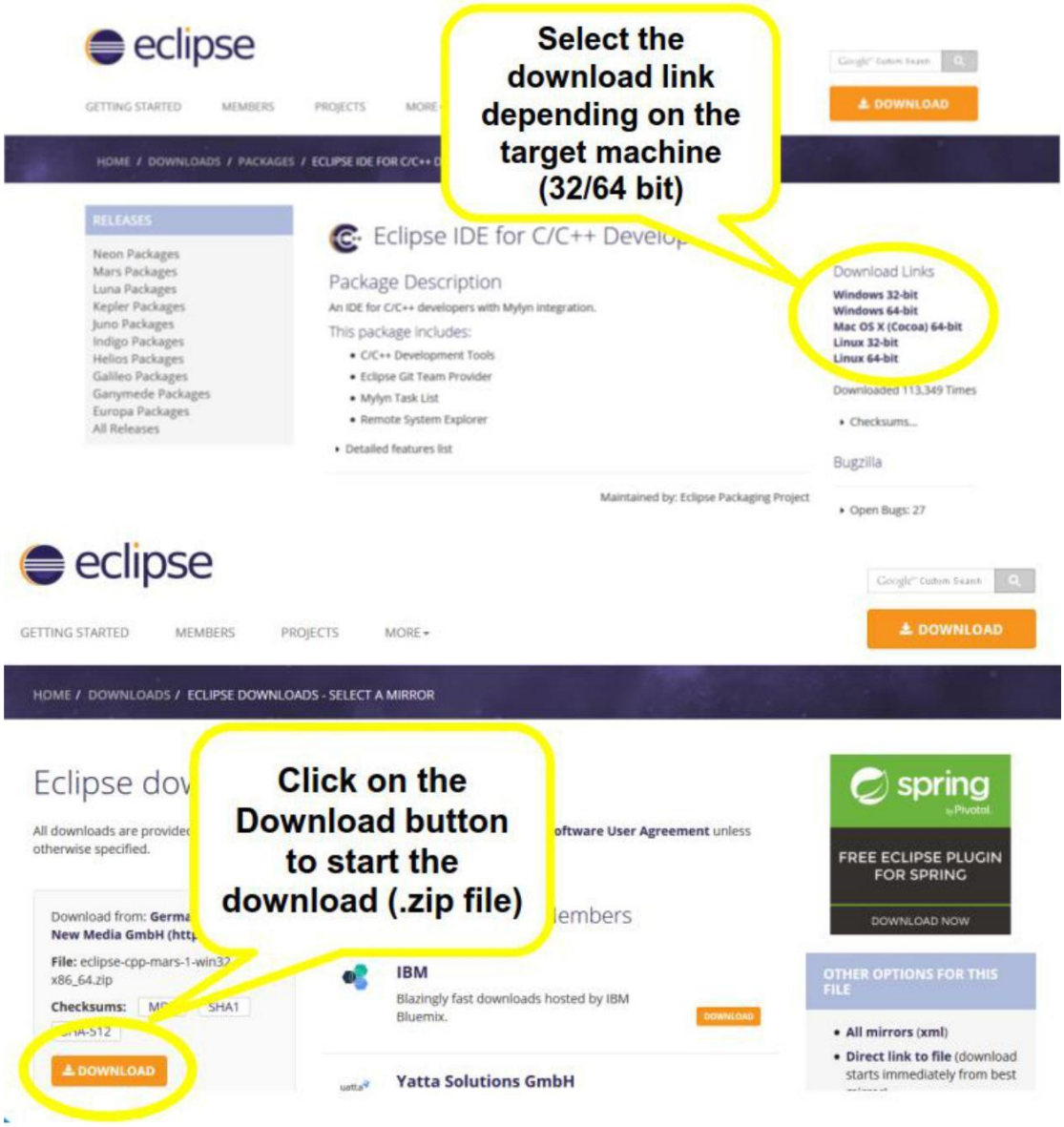
Version 4.6 Neon (or later)

How to get it

You need to download the Eclipse IDE for C/C++ Developers [19].

Installation hints

Visit the download link and follow the instructions as in the following screenshots:



19 http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/neon1a

Pay attention to choose the same architecture (32- or 64- bit) for both Eclipse and the Java Virtual Machine in your PC. You can verify which version of Java is present in your machine by launching the command “java -version” in a Command Prompt: its outcome would clearly state if the Java version within your PC is a 64-bit architecture (otherwise you should assume that it is a 32-bit architecture).

If the two architectures do not match it is possible that Eclipse will show this error on start-up: “Can't start Eclipse - Java was started but returned exit code=13”.

What it’s going to be used for

Eclipse is the recommended IDE to develop applications that will run on the STM32F4 proces-sor of the **SE*cube*™** DevKit.

3.2.4. AC6 Tools

Description

The AC6 Tool will install the GNU Embedded Toolchain for ARM, which is a ready-to-use, open source suite of tools for C, C++ and Assembly programming targeting ARM Cortex-M and Cor-tex-R family of processors. It includes the GNU Compiler (GCC) and is available free of charge directly from ARM for embedded software development on both Windows and Linux operat-ing systems.

The reference platform for this document is the System Workbench for STM32 (SW4STM32) Eclipse plugin.

SW4STM32 is an integrated environment that includes:

- Building tools (GCC-based ARM cross compiler, assembler and linker);
- OpenOCD and GDB debugging tools;
- Flash programming tools

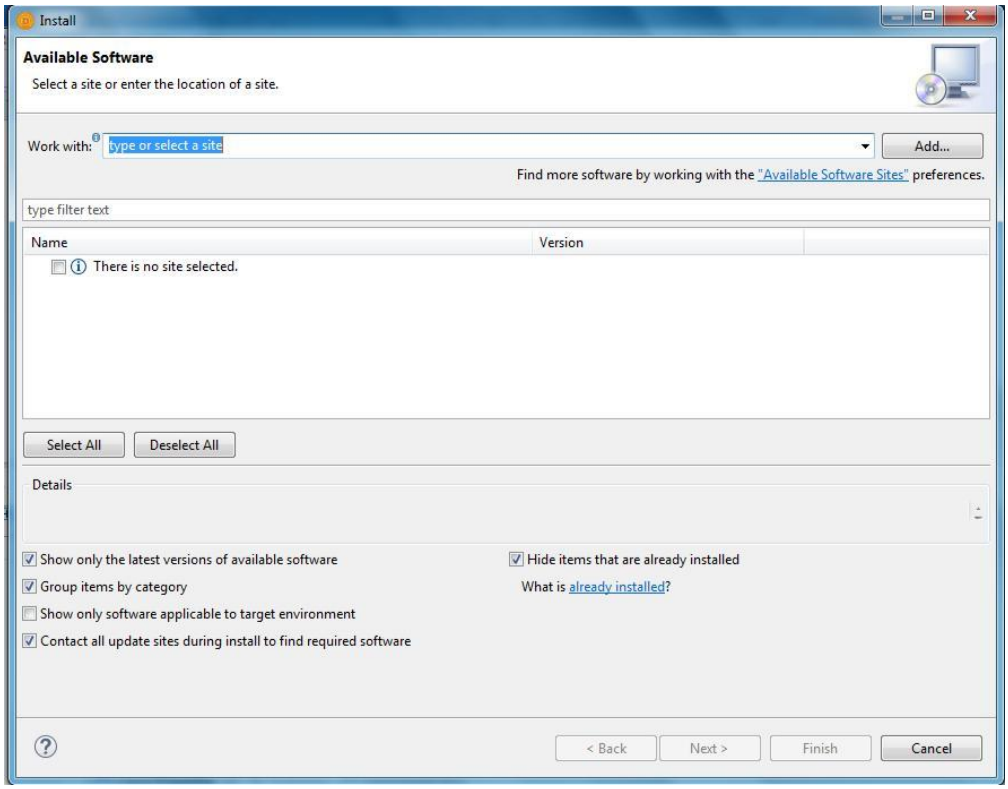
Version required

Version 5.0 (or later).

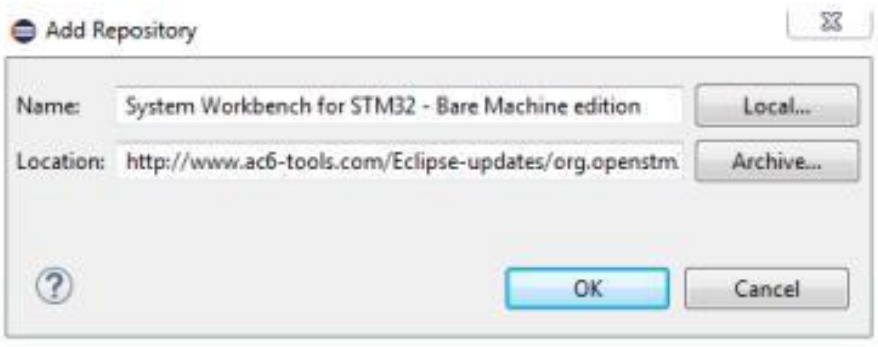
How to get it

To install SW4STM32 as an Eclipse plugin:

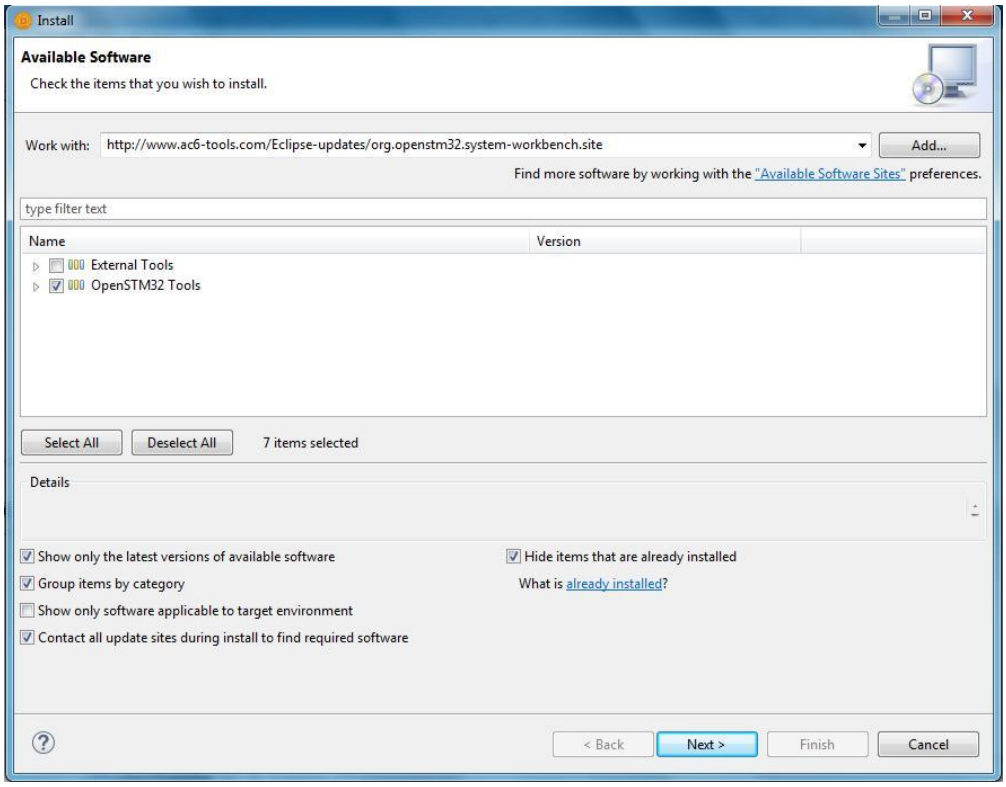
1. launch Eclipse IDE
2. on the toolbar, click «Help >> Install New Software…»
3. in the Available Software window, click «Add»



4. in the Add Repository window, set Name and Location fields as follows, and then click «OK»
    a. Name: System Workbench for STM32 - Bare Machine edition
    b. Location: http://www.ac6-tools.com/Eclipse-updates/org.openstm32.system-workbench.site



5. select OpenSTM32 Tools and click «Next»



6. accept the license agreement and click «Finish» to start the plugin installation, continue the installation also if a warning for incompatible or unsigned components is prompted
7. restart Eclipse

What it's going to be used for

The toolchain will be used to create, build, debug and in general to manage project that will be executed from the STM32 microcontroller comprised within the **SE*cube*™** DevKit.

3.2.5. STM32CubeMX - STM32Cube initialization code generator

Description

STM32CubeMX is a graphical software configuration tool that allows generating C initialization code using graphical wizards. It also embeds a comprehensive software platform, delivered per series. This platform includes the STM32Cube HAL (an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio), plus a consistent set of middleware components (RTOS, USB, TCP/IP and graphics). All embedded software util-ities come with a full set of examples.

STM32CubeMX is an extension of the existing MicroXplorer tool. It is a graphical tool that allows configuring STM32 microcontrollers very easily and generating the corresponding initialization C code through a step-by-step process.

The reference platform for this document is the STM32CubeMX Eclipse plugin.

Version required

Version 4.0 (or later)

How to get it

The software is downloadable free of charge online[20].
After having registered to the website, it will be possible to download a .zip file containing the STM32CubeMX Eclipse plugin; to install it then follow these steps:

1. launch Eclipse IDE
2. on the toolbar, click «Help >> Install New Software…»
3. in the Available Software window, click «Add»
4. in the Add Repository window click on «Archive», select the downloaded ZIP file, and click «OK»
5. check the box corresponding to STM32CubeMX plugin and click «Next»
6. accept the license agreement to install the plugin, continue the installation also if a warning for incompatible or unsigned components is prompted
7. restart Eclipse

What it's going to be used for

STM32CubeMX eases system development providing:
C code generation covering initialization code for standard toolchains

Embedded software libraries and middleware components (e.g., Open-source TCP/IP stack, USB drivers, open-source FAT file system, open source RTOS) with related ex-amples

3.2.6. Lattice Diamond Software

Description

Lattice Diamond® software is the leading-edge software design environment for cost- sensitive, low-power Lattice FPGA architectures. Lattice Diamond's integrated tool environment provides a modern, comprehensive user interface for controlling the Lattice Semiconductor FPGA implementation process.

Version required

Version 3.5 (or later)

20 http://www.st.com/en/development-tools/stsw-stm32095.html
As an alternative (not recommended), it is possible to install the software as a standalone by downloading and extracting the .zip file downloadable from http://www.st.com/en/development-tools/stm32cubemx.html.
If you work under Windows, you can execute directly the .exe executable; if you work under Linux, you have the launch the following command from the command prompt "sudo java -jar filename.exe" (substituting "filename" with the actual file-name of the executable) and to insert your user password if required.

How to get it

The software is downloadable free of charge online[21].

Installation hints

When downloading the software, it is possible to choose the free license.

What it's going to be used for

The software will be used for controlling the implementation process of the Lattice Semiconductor FPGA comprised within the **SE*cube*™** DevKit.

3.2.7. ST-Link/v2 drivers

Description

USB Driver for the ST-Link/v2 programmer.

Version required

Version 4.0.0 (or later)

How to get it

The software is downloadable free of charge online[22].

Installation hints

During the installation procedure, it is possible to receive warnings from the Operating System if drivers are not properly signed; however, the installation procedure should not be inter-rupted.

What it's going to be used for

To allow the usage of the ST-Link/v2 programmer.

3.2.8. ST-Link Utility

Description

The STM32 ST-LINK utility software facilitates fast in-system programming of the STM32 microcontroller families in development environments via the ST-LINK and ST-LINK/V2 tools.

Version required

Version 4.0.0 (or later)

21 http://www.latticesemi.com/en/Products/DesignSoftwareAndIP/FPGAandLDS/LatticeDiamond.aspx?gclid=Cj0KEQjw4_DABRC1tuP-

22 http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link009.html

How to get it

The software is downloadable free of charge online for Windows users[23].

Linux user, instead, can resort to the Open On-Chip Debugger (OpenOCD); this software is downloadable free of charge online[24].

What it's going to be used for

To speed up the usage of the ST-Link/v2 programmer.

3.2.9. Open Source SDK

Description

A Software Development Kit (SDK or "devkit") is typically a set of software development tools that allows the creation of applications for a given system. To exploit all the functionalities of your **SE*cube*™** DevKit, we provide a free and open-source SDK which are commented in this document.

Of relevance within this SDK is a configuration file (*SEcubeDevBoard.ioc*) which stores the con-figuration of microprocessor integrated in the **SE*cube*™** DevKit.

How to get it

This file is available as part of the Open Source SDK which can be downloaded from the following link: http://www.secube.eu/resources.

Once downloaded, extract the content into a known location and browse to "SEcube_SDK\De-velopment\" to find *SEcubeDevBoard.ioc*.

What it's going to be used for

The configuration file is used to generate automatically software driver and or custom config-urations tailored for the microprocessor integrated in the **SE*cube*™** DevKit.

23 http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link004.html

24 https://sourceforge.net/projects/openocd/files/openocd/0.9.0/

4. Assembling the System

In this Section, we list the instructions you need to follow to properly connect the **SE*cube*™** DevKit to the Host PC and to Programmer/debugger, as shown in Figure 41**.**

At the end of this Section you will have acquired a clear overview of the procedures to follow to start using the environment.

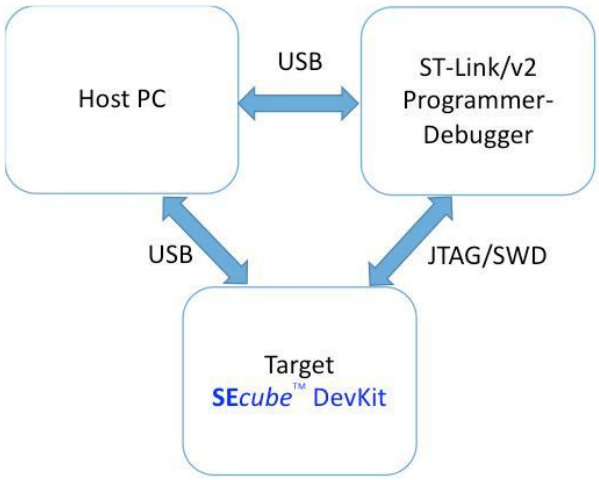


Figure 41 – System Architecture

4.1. Assembling Steps

4.1.1. Using the ST-Link/v2

If you decide to use the ST-Link/v2 programmer, assembling is composed of the following two steps:

1. Connect the **SE*cube*™** DevKit with the programmer by means of the JTAG/SWD cable: the cable should be inserted on the JTAG docks on both the programmer (in this case the orientation of the plug is forced from the dock) and the DevKit (in this case you must pay attention in inserting the plug on top of both lines of connectors and with its protrusion oriented towards the inner side of the DevKit)
2. Connect the ST-Link/v2 with the PC by means of the USB cable

The system assembled is shown in Figure 43, while a close-up on the JTAG connection is in Figure 44.

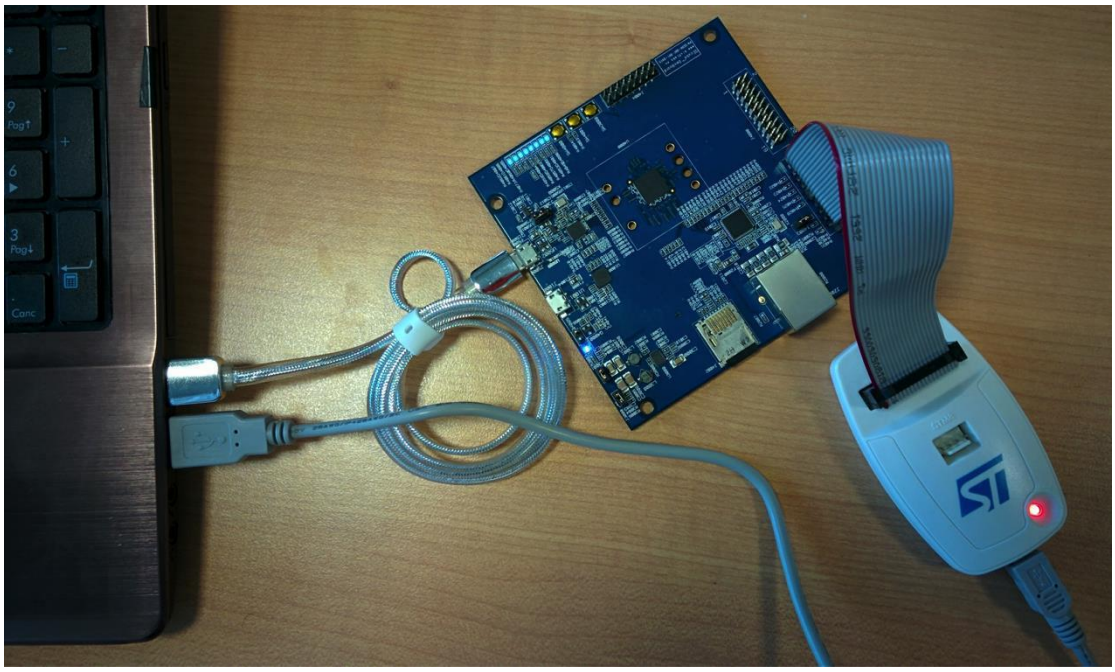


Figure 42 - Connection between the STLink/v2 programmer and the **SE*cube*™** DevKit

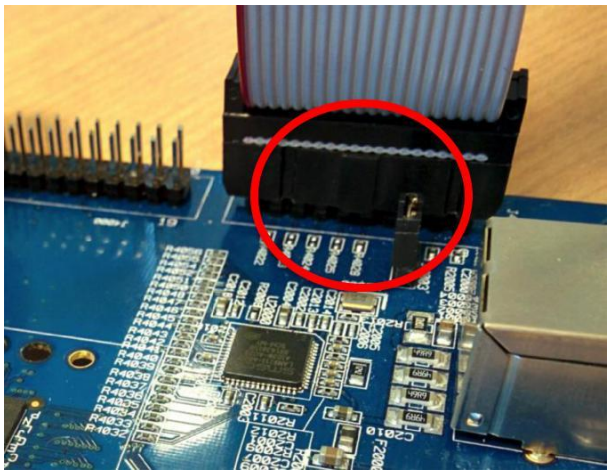


Figure 43 - Connection between the STLink/v2 programmer and the **SE*cube*™** DevKit, close-up (highlighted in red) on the JTAG connector orientation

4.1.2. Using a ST Discovery or Nucleo board

If you decide to use the ST-Link programmer comprised within a Discovery or Nucleo board, assembling requires the following three steps:

1. Isolate the programmer from the rest of the board by moving the jumpers to reach the configuration shown in Figure 44 and described in the STM32 Nucleo-64 boards user manual[25]
2. Connect the **SE*cube*™** DevKit with the programmer by means of the JTAG/SWD cable: the cable should be inserted on the JTAG docks of the DevKit (you must pay attention in inserting the plug with its protrusion oriented towards the inner side of the DevKit) and on the SWD dock of the board, accordingly to the schema in Table 1
3. Connect the ST-Link/v2 with the PC by means of the USB cable

Pin	**CN4**	**Designation**
1	VDD	Target VDD from application
2	SWCLK	SWD clock
3	GND	Ground
4	SWDIO	SWD data I/O
5	NRST	Reset of target MCU
6	SWO	Reserved

Table 1 – Programmer SWD pins schema

25 Available at the following link, please refer to page 18 of the user manual: http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf

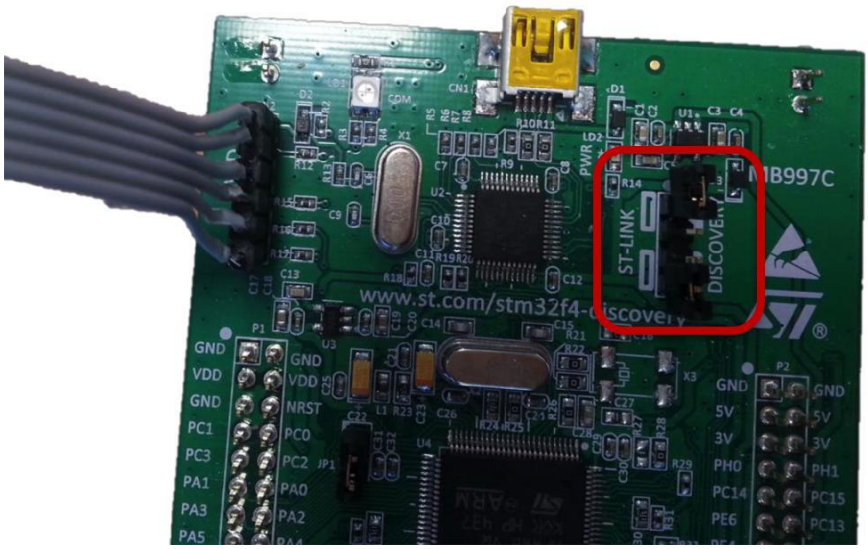


Figure 44 – Jumpers configuration to isolate the ST-Link programmer on a Discovery board (highlighted in red, the same applies to Nucleo boards)

The result is shown in Figure 45.

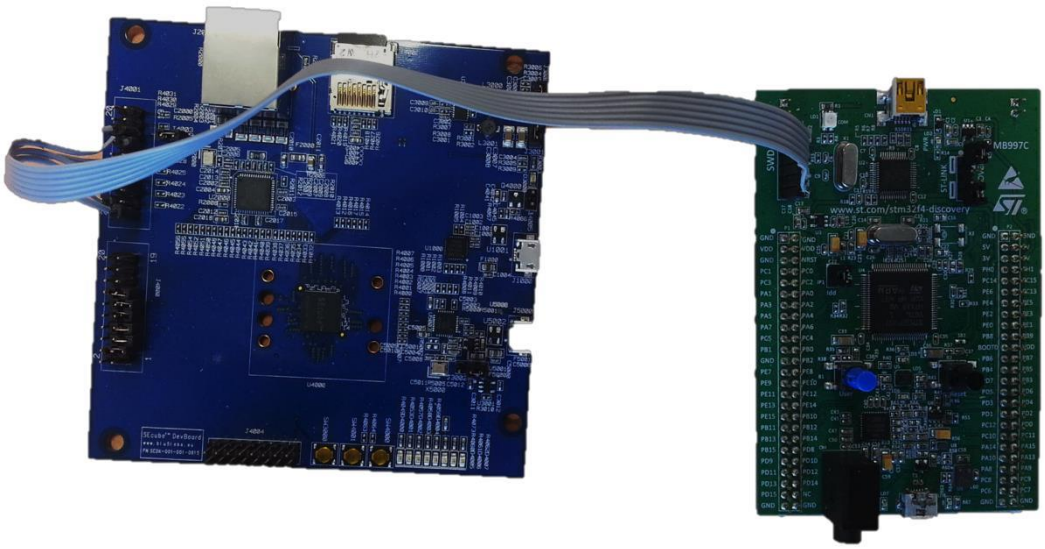


Figure 45 – Connection between the Discovery board and the DevKit (the same applies to Nucleo boards)

4.2. What it should happen

After having properly connected the programmer through the USB interface its signaling led should turn on; after having properly connected the DevKit through the USB interface all its led should turn on.

5. The SE*cube*™ Open Source Software Libraries

This section provides a global overview of the **SE***cube*™ Open Source Software Libraries, then hereinafter will be often referred to as the **SE***cube*™ SDK.

5.1. Introduction and Libraries Architecture

The **SE***cube*™ SDK consists in a set of multi-level and open source C libraries, collectively referred to as APIs in the sequel.

The SDK will be later complemented with a service-oriented approach to provision and execution, and with the ability to use such services in models, enabling a model driven development of secure applications[26] in the CINCO metamodeling platform[27].

From the user/developer point of view, the APIs have been implemented targeting two nested environments depending on where physically the code runs:

Device-Side
Host-Side

The *Device-Side* relates to software running on the embedded processor of the **SE***cube*™-based hardware device (e.g., the **USE***cube*™) and it practically exports APIs that are used by the *Host-Side* software. This, in turns, runs on the appliance hosting the device (e.g., the Laptop), and exploits the Device-Side functionalities through an open source secure RCP (Remote Call Procedure) protocol, encapsulated in the SDK.

In this scenario, **USE***cube*™ acts as a powerful coprocessor providing a secure and fully controlled execution environment.

From the architectural point of view, the libraries have been implemented targeting hierarchical abstraction levels.

At each level, each component (but the lowest one) represents a "service" for the upper level and relies on "services" provided by the next lower level, only.

5.2. *Device-Side* Libraries

The *Device-Side* software includes the libraries of basic functionalities that are executed on the embedded processor of the **SE***cube*™-based hardware device.

From the architectural point of view, the libraries have been implemented targeting by defining 3 different main cores, providing the following APIs:

- Communication Protocol and Provisioning APIs: for these purpose, **Communication Core** and the main **Core** are used;
- Basic Security APIs: **Dispatcher Core** provides these functionalities;

[26] A Carelli, G Di Natale, P Trotta, T Margaria Towards Model Driven Design of Crypto Primitives and Processes. Proc. Int. Conf. on Security and Management (SAM), part of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), July 2016

[27] S Naujokat, J Neubauer, T Margaria, B Steffen Meta-level reuse for mastering domain specialisation. International Symposium on Leveraging tions of Formal Methods, 218-237

5.2.1. *Communication Protocol* and *Provisioning* APIs (Communcation Core, SE3 Core)

Closest to the device, these cores mainly include *software drivers*, which offer basic functionalities to manage and access:

- internal peripheral (e.g., TRNG, internal Flash memory, timers, SD card, …), managed by the main SE3 Core;
- external communication interfaces, namely USB, UART, SPI, and GPIOs, managed by the Communication Core;

This level is entitled to discover possible **SE*cube*™** devices connected to the host system and create a bidirectional communication channel.

These functionalities are exposed through a simple set of APIs summarized in Appendix D.1.

5.2.2. *Basic Security* APIs (Dispatcher Core)

This Core provides the basic APIs for implementing secure applications, including basic cryptographic algorithms and various utilities, like the SW and HW crypto cores management (FPGA, smartcard, microprocessor) and the interface to the new component called SEkey.

In addition, it is used to secure the communication channel after a successful login, which can be easily implemented as a multi-factor authentication thanks to the 3 hardware elements inside the chip and the password provided by the user.

The services exposed in this core includes several APIs to manage both the device provisioning and the user/admin operational processes:

- Pin initialization
- Primary keys initialization
- Login & logout
- Information retrieval
- Encryption/decryption/signature verification
- Key management

These functionalities are exposed through a simple set of APIs summarized in Appendix D.2.

5.2.2.1. A new key manager: SEKey

The previously mentioned operations regarding the key managment are made possible by the usage of SEkey device, a key management system able to create, erase, administer, update the communication keys of the user groups; every time a group has to be created, this new component has to provide to the administrator the necessary mechanisms to create it. Moreover, information about one group must be exclusively read or written by the users of the group itself. When creating a group, the administrator has to choose the algorithms to create and exchange the keys between the users (as DES, AES, DH, RSA, ...), the group policies, and the components of the group. All these operations will be performed by using SEkey.

5.3. *Host-Side* Libraries

On the host side, the software is tailored for existing devices (e.g., laptops or Desktop PC) that see the **SE***cube*™ hardware as an external peripheral which exposes the services described above (section 5.2), properly organized in 2 abstraction levels.

The **SE***cube*™ device is thus seen by the host as a closed black box providing services. The host starts the service request by sending the related command and the optional data packets, through a proper interface, per a custom protocol.

The *Host-Side* Libraries are designed to be scalable, i.e., for dealing with multiple devices, and portable on different Operating Systems, thus limiting the usage of and isolating platform-dependent modules. They practically run on top of the host OS, directly relying on the OS System calls. To improve portability and migrations, the libraries are organized in such a way that all the OS-dependent sub-modules (e.g., communication interface, file system, etc.) be easily identifiable.

From the architectural point of view, the *Host-Side* Libraries have been implemented targeting 4 hierarchical abstraction levels, and namely:

- *Communication Protocol* and *Provisioning* APIs (Level0 Host-Side – L0)
- *Basic Security* APIs (Level1 Host-Side – L1)
- *Intermediate Security* APIs (Level2 – L2)
- *Advanced Security* APIs (Level3 – L3).

5.3.1. *Communication Protocol* and *Provisioning* APIs (Level0 Host-Side)

This level implements the basic functionalities to communicate with the **SE***cube*™, resorting to the Level0 Device-Side (L0_dev_) services (Figure 51).

The APIs that this level provides to the upper level include, among the others:

- sending/receiving command and data packets from/to the device
- segmentation of raw data streams into protocol-compliant packets
- functions implementing standard cryptographic algorithm
- low-level error management functions
- commodities functions, such as low-level data manipulation in dealing with possible endianness mismatches between the host side and the **SE***cube*™ embedded processor

All of these functionalities are exposed through a simple set of APIs summarized in Appendix E.1.

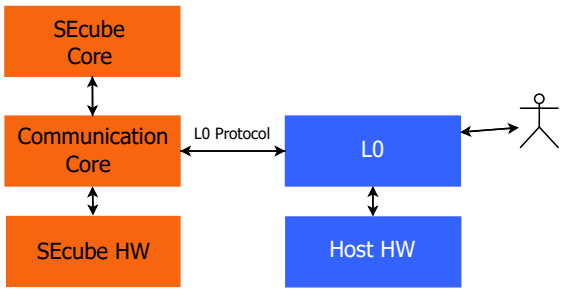


Figure 51 – Communication Protocol

5.3.2. *Basic Security* APIs (Level1 Host-Side)

This level relies on L0 services to provide the basic APIs for implementing secure applications (Figure 52), including multi-factor login, secure communication channel, cryptographic algorithms and key management.

The L1 includes 16 functions, of which:

6 concern login/logout

4 concern the key management

4 concern the cryptography

2 concern utility functions

In addition, L1 allows developers to manage several SEcube™ devices concurrently, providing dedicated operation control flows (one command/response session per communication channel), which allow encoding and decoding commands for the individual SEcube™ target, being able to communicate to the SEcube Core that the Dispatcher Core is needed.

These functionalities are exposed through a simple set of APIs summarized n Appendix E.2.

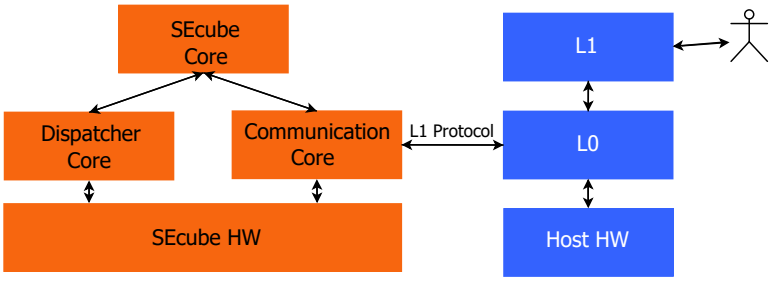


Figure 52 – L1 layer

5.3.3. *Intermediate Security* APIs *(Level2 – L2)*

This level relies on L1 services to provide the APIs for implementing more abstract secure functionalities (Figure 53). Typical examples include APIs for the protection of data both at-rest and in-motion, without being forced to understand in detail all the low-level hardware and security mechanisms.

The set of APIs available at L2 are outside the scope of this document.

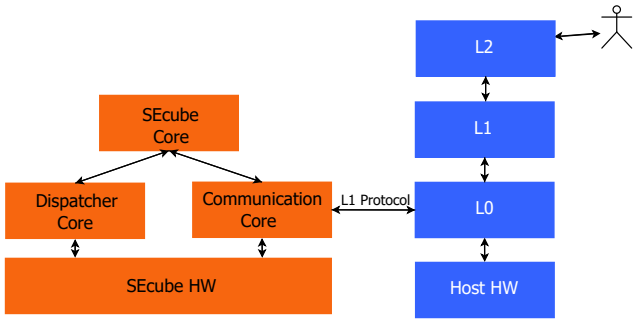


Figure 53 – L2 level

5.3.4. *Advanced Security* APIs *(Level3 – L3)*

This level relies on L2 services (Figure 54) to develop secure applications, such as Secure File Systems, Key management Systems, etc.
The set of APIs available at L3 are outside the scope of this document.

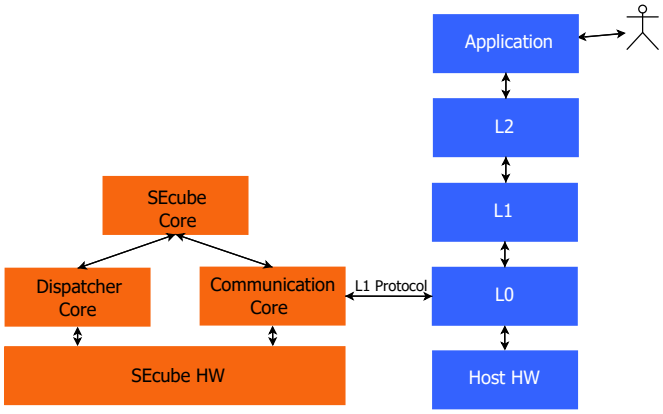


Figure 54 - L3 level Application

6. Installing the SE*cube*™ OpenSource Software Libraries

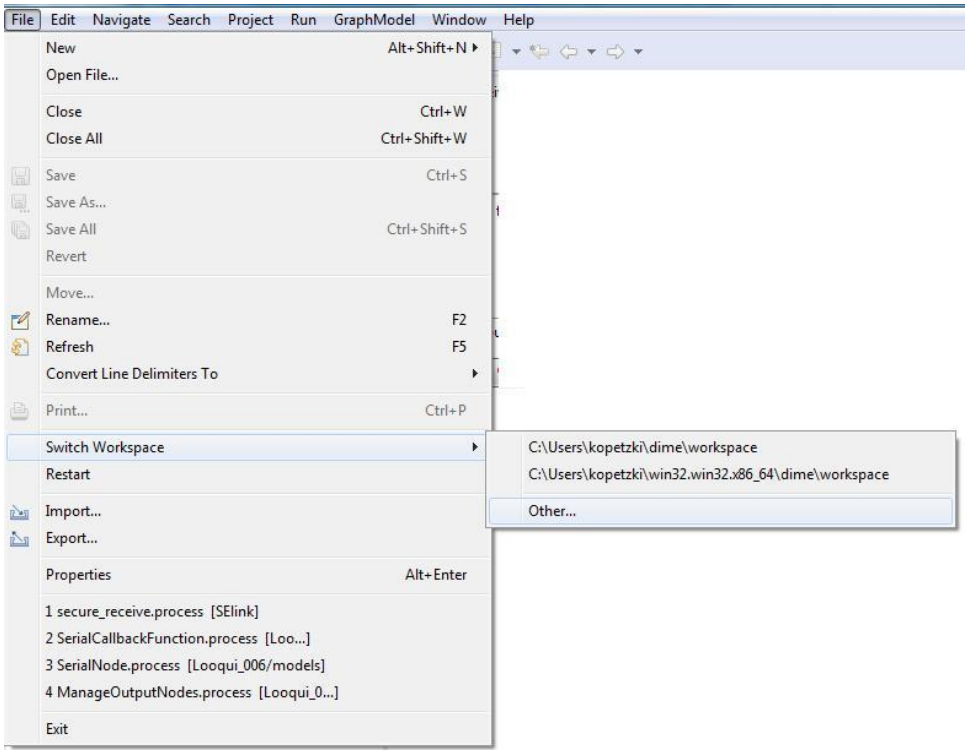
In this Section, we list the instructions you need to follow to import the **SE*cube*™** software libraries within an Eclipse project**.**

At the end of this Section you will have acquired a clear overview of the procedures to follow to import the libraries and use them to foster the development of your application.

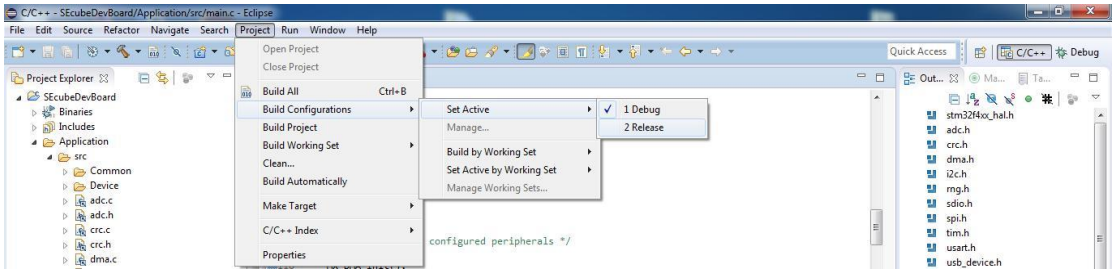
6.1. SE*cube*™ Open Source Software Libraries – Device side

Hereby is listed a step-by-step guide to create the binaries files that will be executed on the **SE*cube*™** DevKit:

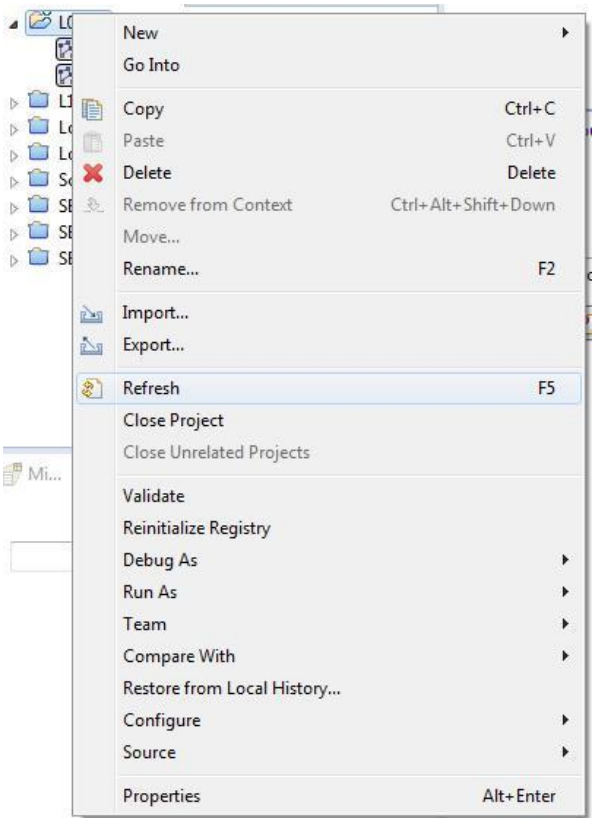
1. Download the "Environment.zip" (the file is inside the SDK package) file containing the project from the following link: http://www.secube.eu/resources/
2. Extract the .zip file to a known location
3. Launch Eclipse
4. Change Eclipse perspective to «C/C++ selecting Window >> Perspective >> Open Perspective >> Other… >> C/C++»
5. Switch the Eclipse workspace to «File >> Switch Workspace >> Other…» and select the "ws" folder contained within the "Environment" folder previously extracted from the .zip file



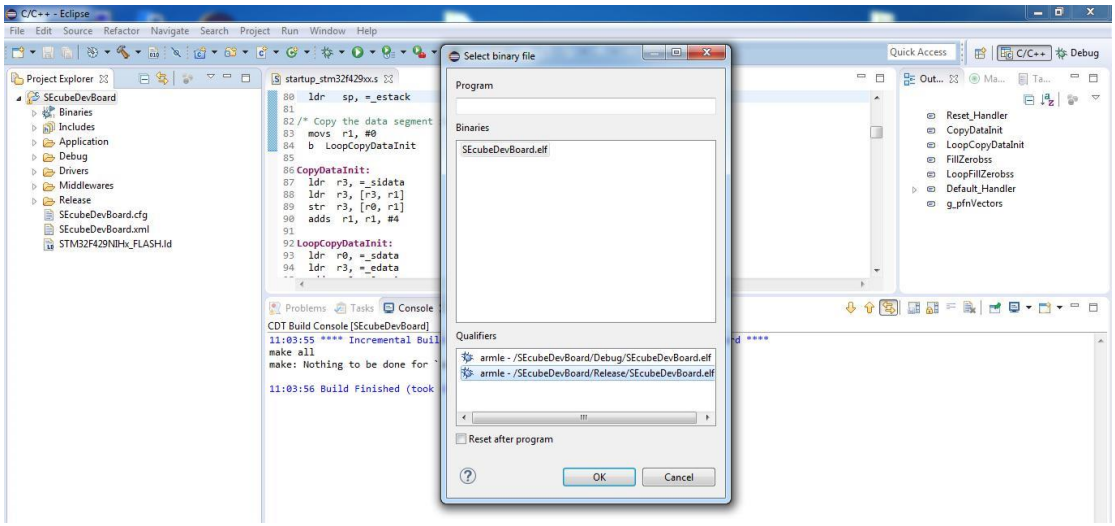
6. Set the Debug configuration from «Project >> Build Configuration >> Set Active»



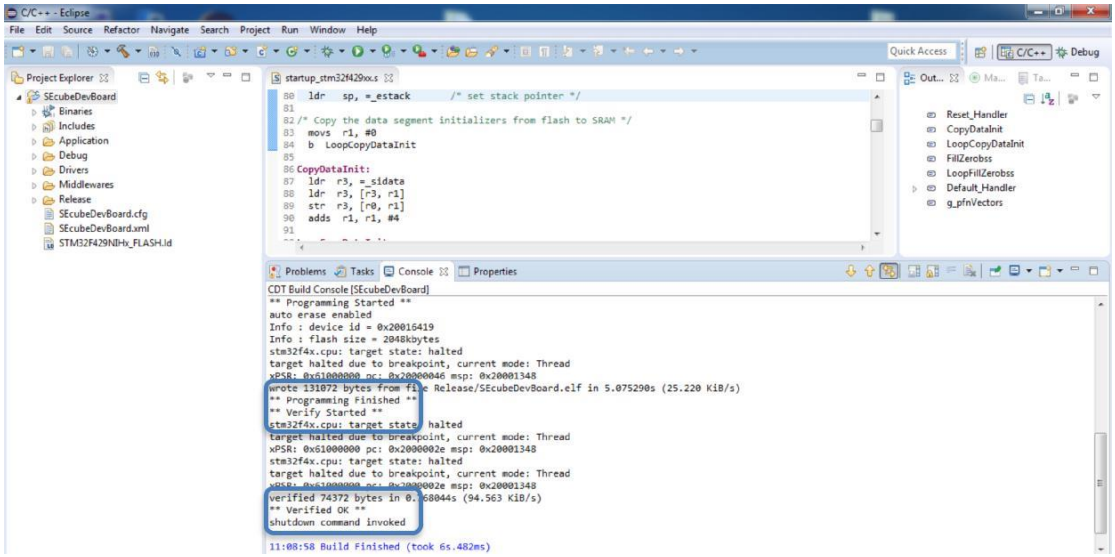
7. Build the project in Debug mode from «Project >> Build All»
8. Set the Release configuration from «Project >> Build Configuration >> Set Active»
9. Build the project in Release mode from «Project >> Build All»
10. Right-click on the project in the Project Explorer and select «Refresh»



11. Connect the DevKit as described in Section 4.2
12. Run the project by right-clicking on it in the Project Explorer and selecting the Release binary under «Target >> Program Chip» (i.e., select the label containing the string "/Release")



13. Wait until the debugger shows the messages “Programming Finished” and “Verified OK”



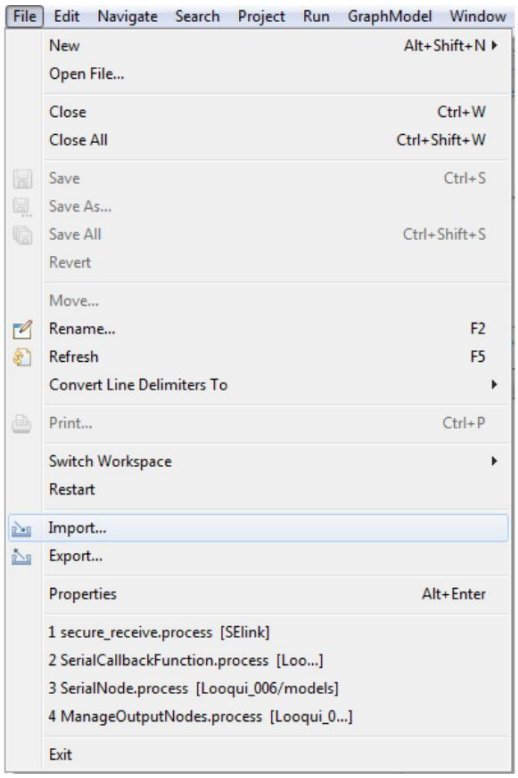
Now your DevKit is fully configured and the STM32 microprocessor is ready to be used to develop your security applications.

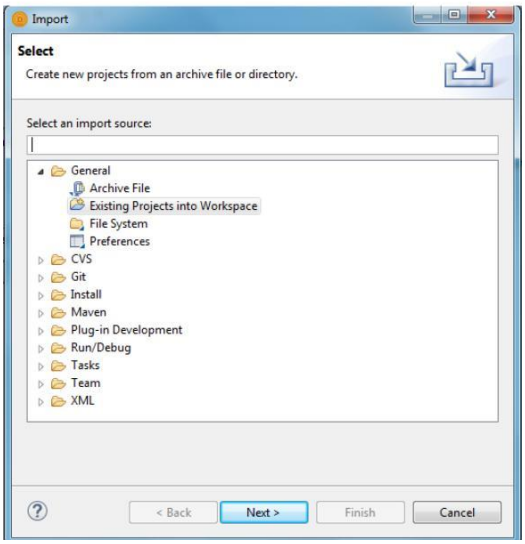
6.2. SE*cube*™ OpenSource Software Libraries – Host side

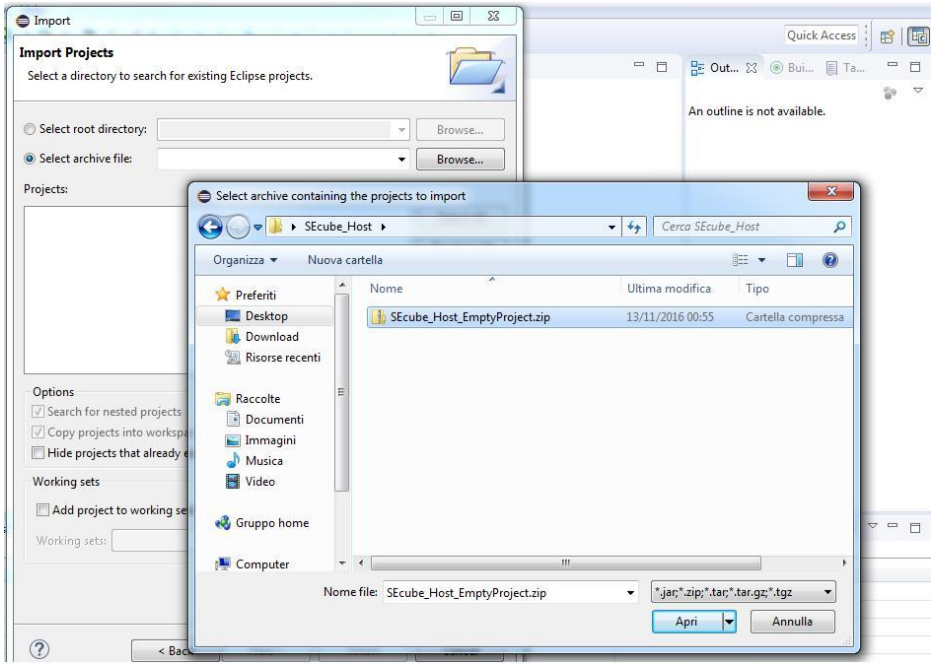
Hereby is listed a step-by-step guide to create a C project to use leverage on the capabilities of the DevKit:

1. Download the “SEcube_Host_EmptyProject.zip” file (the file is inside the SDK package) containing the empty project from the following link: http://www.secube.eu/resources/
2. Launch Eclipse

3. Change Eclipse perspective to «C/C++ selecting Window >> Perspective >> Open Perspective >> Other… >> C/C++»
4. Import the project «File >> Import…», select “Existing file into workspace” and press “Next”, Select archive file, browse to the file downloaded and press “Finish”







5. Set your preferred configuration from «Project >> Build Configuration >> Set Active»
6. Build the project from «Project >> Build All»
7. Connect the **SE*cube*™** DevKit with the USB cable
8. Now you are ready to write your code within the "main.c" file in the Project Explorer, compile it and run it. Some examples are reported in the following.

WARNING! Before executing any kind of developed "main.c" file, the first time, after the chip programming operation, it will be necessary to run the device initialization main program, that will be included in the downlodable development kit inside the directory "SEcube_SDK\Libraries\Examples\Device_Initialisation\"; so, please use the following procedure:

1. Launch Eclipse
2. Import the project following section 6.2
3. Open the file named "main.c"
4. Replace its content with the code contained in the SDK under the folder "SEcube_SDK\Libraries\Examples\Device_Initialisation\"
5. To correct an inclusion problem, substitute `#include "../secube-host/L1.h"` with `#include "./secube-host/L1.h"`;

Now, you are ready to run your first program with SEcube.

6.3. Running your first program: Hello World (host-side)

Typically, the first program you run in a new environment is a basic one (e.g., one printing a string such as "Hello World" on the screen) intended to let you test that the environment is properly configured and to familiarize with the configurations needed to make the program code executable.

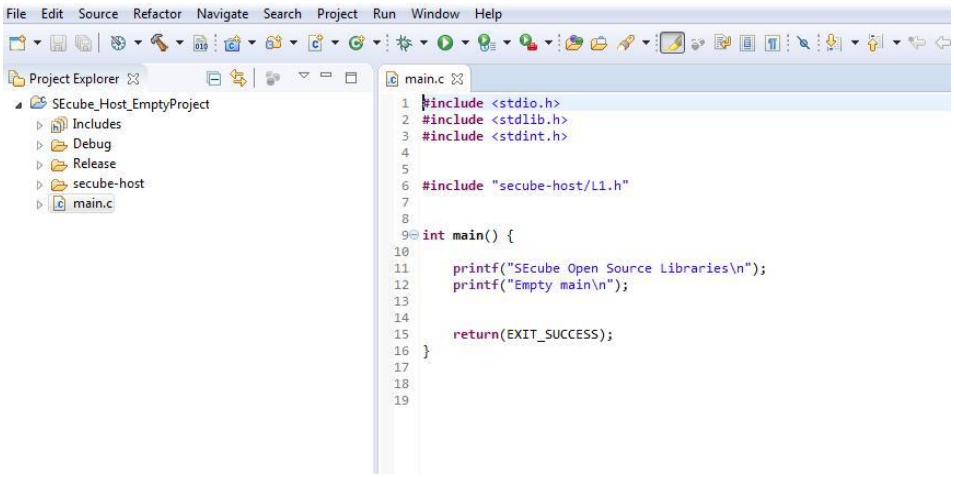
The code shown in Appendix C is a first example of how to use the Open Source Libraries; it resorts only on the STM32 microprocessor as it triest to log in and out from your **SE*cube*™** DevKit.

Hereby we list a step-by-step guide to run your first program on the **SE*cube*™** DevKit.

Note: to run this software you must have already initialized your device.

To use it follow this step-by-step guide:

6. Launch Eclipse
7. Import the project following section 6.2
8. Open the file named "main.c"
9. Replace its content with the code shown in Appendix C (it can also be found in the SDK under the folder "SEcube_SDK\Libraries\Examples\HelloWorld\")



5. Connect the DevKit with USB cable
6. Set the Release configuration from «Project >> Build Configuration >> Set Active»
7. Build the project from «Project >> Build All»
8. Run the project

When setting up a project with the provided files for the API, a directory structure is implicitly assumed. As SEfile includes "se3/L0.h", "se3/L1.h" etc., a folder "se3" must be present to contain the expected parts of the API. The tool for setting up the environment could serve as such an example project[28]

28 Freely available on-line (https://bitbucket.org/ede1/secube-command-line-tools.git).

The directory structure of this project is the following:

```
├── build
├── CMakeLists.txt
├── include
│   └── main.h
├── secube
│   ├── se3
│   │   ├── aes256.c
│   │   ├── aes256.h
│   │   ├── crc16.c
│   │   ├── crc16.h
│   │   ├── L0.c
│   │   ├── L0.h
│   │   ├── L1.c
│   │   ├── L1.h
│   │   ├── pbkdf2.c
│   │   ├── pbkdf2.h
│   │   ├── se3c0def.h
│   │   ├── se3c1def.h
│   │   ├── se3comm.c
│   │   ├── se3comm.h
│   │   ├── se3_common.c
│   │   ├── se3_common.h
│   │   ├── sha256.c
│   │   └── sha256.h
│   ├── SEfile.c
```

| └── SEfile.h

└── source

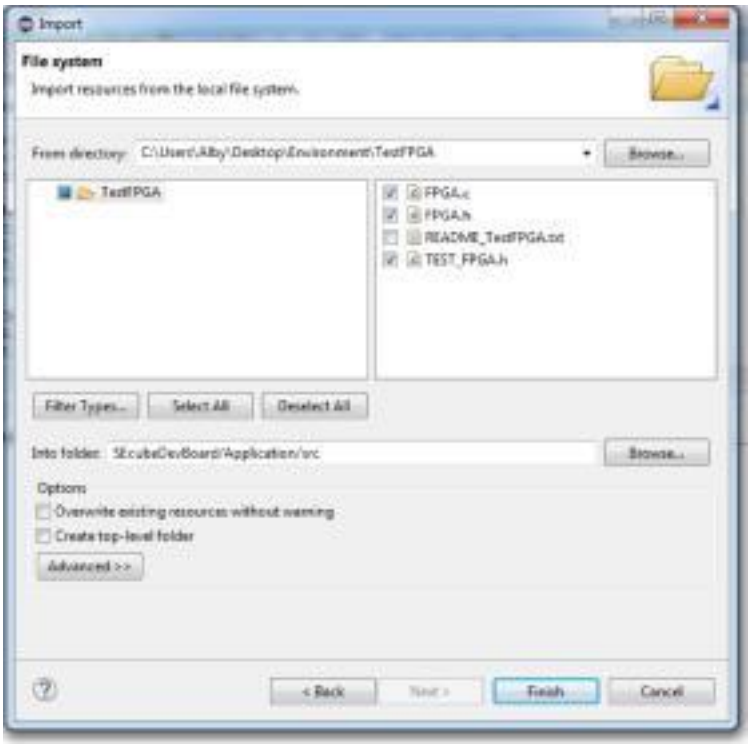
└── main.c

6.4. Running your first program: FPGA_LED (device-side)

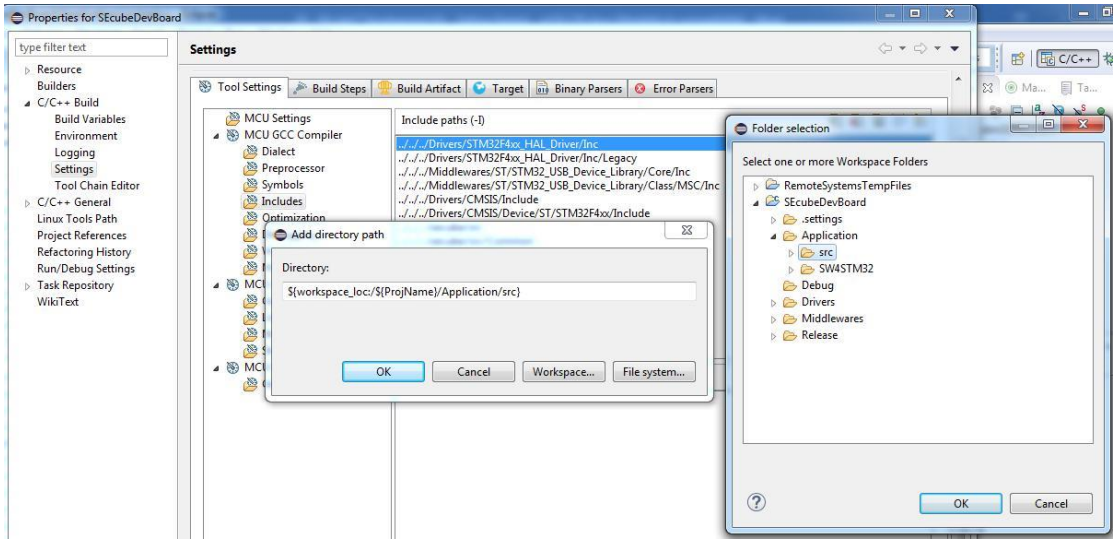
The procedure shown in this paragraph guides you to a first example of how to use the Open Source Libraries with the FPGA; it programs the FPGA embedded in the **SE*cube*™** chip to make the led blink.

Hereby we list a step-by-step guide to run this program:

1. Download the OpenSource SDK from http://www.secube.eu/resources/
2. Follow the steps in Section 6.1
3. Import the project as described in section 6.2
4. Import the necessary «File >> Import…», select "Filesystem" and press "Next"
5. Browse to the directory where the SDK has been downloaded and then to the path "SEcube_SDK\Libraries\Examples\TestFPGA\"
6. Select the files in that folder (FPGA.c, FPGA.h and TEST_FPGA.h); you might want to set also "Destination Folder" to "SEcubeDevBoard/Application/src" and then press "Finish"



7. Configure both “Debug” and “Release” configurations from «Project >> Properties >> C/C++ Build >> MCU GCC Compiler >> Includes» and add the “Destination folder”



8. Now edit the code in “main.c” file including the header file “FPGA.h”
9. Also in “main.c” add a call to “B5_FPGA_Programming()” function
10. Open the file named “gpio.c” and add the following lines to the function “MX_GPIO_Init()”:

```
/*Configure GPIO pin : PE2 */
GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/*Configure GPIO pins : PE3 PE4 PE5 PE6 */
GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
```

11. Save the changes to all files and build the project
12. Connect the DevKit as described in Section 4.2
13. Run the project by right-clicking on it in the Project Explorer and selecting the Release binary under «Target >> Program Chip» (i.e., select the label containing the string “/Release”)

Notice that the after turned on, flashing of the FPGA might require some time (up to two minutes) to be completed. After that, you should see that the led on the board start blinking at different frequencies.

7. Running the provided Functional Test

In this Section, we list the instructions you need to follow to run the provided functional test with the **SE***cube*™ DevKit**.**

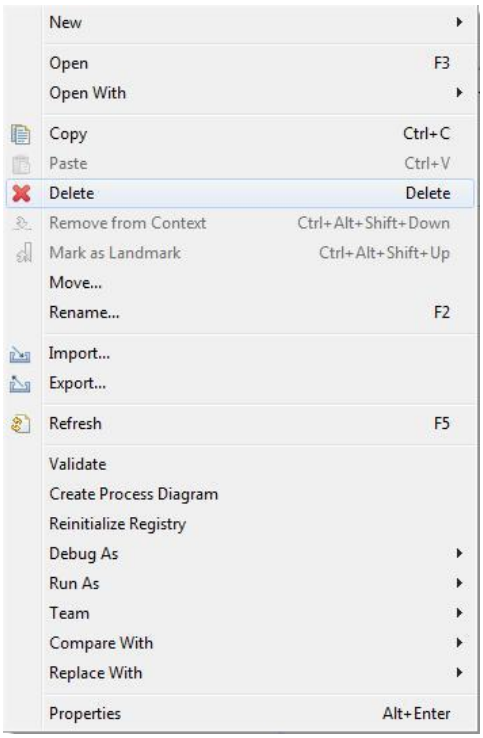
At the end of this Section you will have acquired a clear overview of the procedures to follow to run programs on the environment.

Notice that to run this program your **SE***cube*™ DevKit must be already initialized.

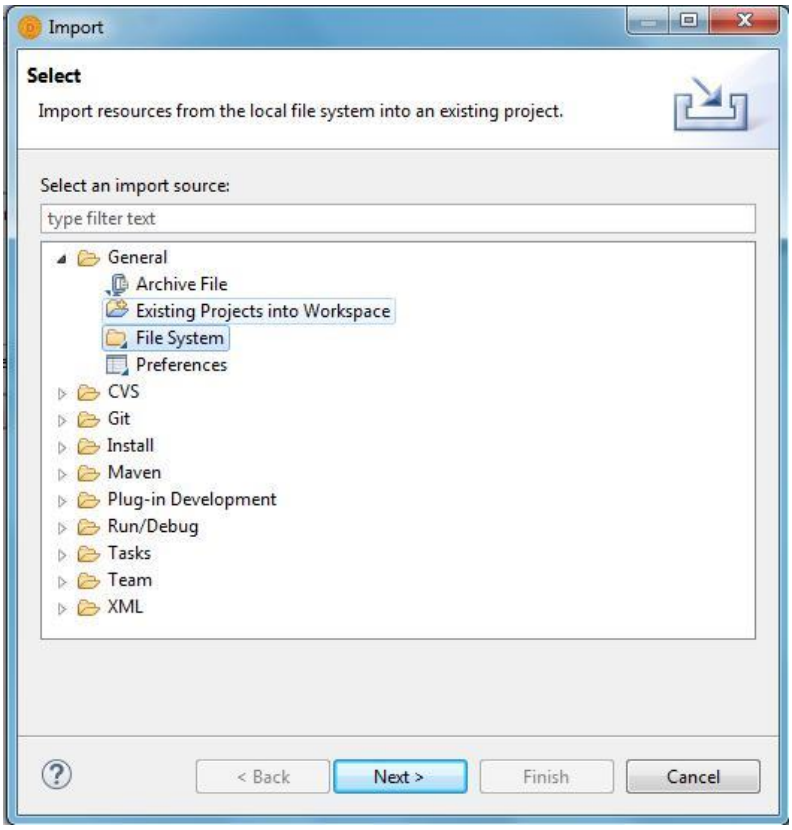
7.1. Step

Hereby is listed a step-by-step guide to run the provided functional test:

1. Download the “SEcube_SDK.zip” file containing the project from the following link: http://www.secube.eu/resources/
2. Extract the .zip file to a known location
3. Launch Eclipse
4. Locate the project presented in section 7.2 in the folder “SEcube_SDK\Examples\FunctionalTest”
5. Locate the “funct_test.c” file in the Project Explorer
6. Right-click on it and select “Delete”



7. Import the new “main.c” file from «File >> Import…», select “File System” and press “Next”, browse the file “SEcube_SDK\Examples\FunctionalTest” folder within the folder previously extracted, select the “funct_test.c” file and press “Finish”



8. Set the Debug configuration from «Project >> Build Configuration >> Set Active»
9. Build the project in Debug mode from «Project >> Build All»
10. Set the Release configuration from «Project >> Build Configuration >> Set Active»
11. Build the project in Release mode from «Project >> Build All»
12. Right-click on the project in the Project Explorer and select «Refresh»
13. Connect the **SE*cube*™** DevKit with USB cable
14. Run the project by right-clicking on it in the Project Explorer and selecting the Release binary under «Run as >> Local C/C++ Application»

7.2. Comments

The provided functional test exploits capabilities offered by the Open Source SDK to show the basic common operations typically used with a **SE*cube*™** device; to provide an in-depth un-derstanding of it, its code is hereby decomposed and commented.

The functional test can be found in the SDK in under the folder "\SEcube_SDK\Libraries\Examples\FunctionalTest\". Starting from the includes instructions

```
#include <stdint.h>
#include <stdbool.h>
#include "../secube-host/L1.h"
```

It is possible to see how the SDK level "L1", offering high-level functions, is being used.
The global declarations

```
static uint8_t pin[32] = {
 'a','b','c','d',  0,0,0,0, 0,0,0,0, 0,0,0,0,
 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0
};
static uint8_t pin0[32] = {
 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0
};
```

contain the PIN used to access the device. Each PIN is a generic value of 32-byte. Note that the value of `pin0` is used to login the first time just after the device is initialized.

Skipping to the main, there is a common snippet used to list all the **SE*cube*™** device con-nected to the host

```
L0_discover_init(&it);
while (L0_discover_next(&it)) {
      printf("SEcube found!\nInfo:\n");
      printf("Path:\t %ls\n",
                  it.device_info.path);
      printf("Serial Number: ");
      print_sn(it.device_info.serialno);
      printf("\n\n");
}
```

Here it is shown how you can retrieve information about the device (i.e., its path and its serial number) from the structure `it.device_info`.

Next step allows to open the device (if it is correctly found).

```
L0_discover_init(&it);
if (!L0_discover_next(&it)) {
      return_value = L0_open(&dev,
      &(it.de-vice_info), SE3_TIMEOUT);
}
if (return_value != SE3_OK) {
      printf("Error opening device\nAbort
                  Tests");
      return(0);
}
else
{
      printf("Open Device success\n");
}
```

Without knowing which **SE*cube*™** device is of interest, here the first available device is opened with `L0_open()` function from level "L0". It should be noted that every function in the SDK returns a value stating whether the operations has been completed successfully (`SE3_OK`) or not (in this case the value returned is bigger than 1). So, it is good practice to always check for the expected return value after calling a function of the SDK.

When a device is opened, it does not offer any facilities but the echo service. This service can be used to test if the link between the host and the device side is working and the peripherals can communicate.

```
if (test_echo(&dev)) {
        printf("Echo success\n");
  }
  else {
        printf("Error Echo\n");
  }
```

The function `test_echo()` does not belong directly to the SDK; it requires as only parame-ter a pointer to the device where to test the echo service. In this function, the buffer of ran-dom data to be sent is generated and the other to be received is allocated. The test will send by default 1 MiB of data.

```
enum {
        TEST_SIZE = 1 * 1024 * 1024
};

  […]
test_randbuf(TEST_SIZE, &sendbuf);
recvbuf = (uint8_t*)malloc(TEST_SIZE);
```

The test is repeated for NRUN times. The data sent can be at most SE3_REQ_MAX_DATA bytes.

```
  for (rep = 0; rep < NRUN; rep++) {
        printf("Echo - Run %d... ", rep);
        sp = sendbuf;
        rp = recvbuf;
        n = TEST_SIZE / SE3_REQ_MAX_DATA;
        for (i = 0; i < n; i++) {
              r = L0_echo(dev, sp,
                          SE3_REQ_MAX_DATA, rp);
              if (SE3_OK != r) goto cleanup;
              sp += SE3_REQ_MAX_DATA;
              rp += SE3_REQ_MAX_DATA;
        }
  […]
```

Finally, the data received in `recvbuff` is compared with the one sent `sendbuf`:

```
if (memcmp(sendbuf, recvbuf, TEST_SIZE))
      goto cleanup;
printf("   -> OK\n");
```

To use any service offered by the **SE*cube*™** device other than the echo service, the login pro-cedure is required. The function `test_login()` shows how to perform log in operations with the device. Several login operations are completed with different PINs:

Log in as User with default pin (all zeroes) - It should fail when initialized

```
return_value = L1_login(&s, dev, init_pin,
                                  SE3_ACCESS_USER);
```

Log in as User with correct PIN

```
return_value = L1_login(&s, dev, pin,
                                  SE3_ACCESS_USER);
```

Log in as Admin with correct PIN

```
return_value = L1_login(&s, dev, pin,
                                  SE3_ACCESS_ADMIN);
```

Log in as User with wrong PIN

```
return_value = L1_login(&s, dev, pin_aaaa,
                                  SE3_ACCESS_USER);
```

To pass this test, both User and Admin PIN must be set to "*test*" (otherwise you should change accordingly the content of the `pin` arrays).

The last operation in the function `test_algorithm()` shows how to get the list of encryption algorithms available. Also for accessing to this service, the login is required. First a suitable array is created:

```
#define MAX_ALG 10
[…]
se3_algo alg_array[MAX_ALG];
```

Then, the "L1" function requesting the algorithm list is called

```
printf("Retrieving algorithm list...\n");
return_value = L1_get_algorithms(&s, 0,
                  MAX_ALG, alg_array, &count);
if (return_value != SE3_OK) {
      printf("Error retrieving algorithm
                                    list");
      return(!SE3_OK);
}
```

Finally, for each algorithm received (i.e., available on the device), its characterizing information (e.g., its name) are printed out with the following code:

```
for (i = 0; i < count; i++) {
     memcpy(buff_name, alg_array[i].name,
     SE3_CMD1_CRYPTO_ALGOINFO_NAME_SIZE);

     printf("Algorithm name: %s\n",
                    buff_name);

          printf("Algorithm type: %u\n",
        (unsigned int)alg_array[i].type);

     printf("Algorithm block size: %u\n",
      (unsigned int)alg_array[i].block_size);

     printf("Algorithm key size: %u\n\n\n",
     (unsigned int)alg_array[i].key_size);
}
```

8. From the SE*cube* ™ DevKit to the USE*cube*™ Stick

To migrate your project from the **SE***cube*™ DevKit to the **USE***cube*™ Stick, you do not need any programmer. You have just to use the boot loader embedded in the **USE***cube*™ Stick and the SEcube™ USB Loader Free Software available online[29].

This software allows injecting your binary image file into the internal **SE***cube*™ flash, starting from the address 0x08020040. Once you decided the starting address, be sure that the same address is configured in the linker.

In order to guarantee the maximum security level, the bootloader allows the final image to take the full control of the hardware. On this purpose your image shall remap the interrupts vector table as soon as possible, as shown in the following code:

```
/* new vector table in RAM */
uint32_t vectorTable_RAM[256] __attribute__(( aligned
(0x200ul) ));
/* vector table ROM  */
extern uint32_t  __Vectors[];

int main(void)
{
      // Hardware Abstraction Layer
        Initialisation HAL_Init();

      // Configure the system clock
        SystemClock_Config();

      // Remapping Interrupt Vector (overload the USB Loader
                                      Interrupt Vector)
      uint32_t i;
      for (i = 0; i < 256; i++) {
          vectorTable_RAM[i] = __Vectors[i];
                      /* copy vector table to RAM */
      }

      // Interrupt Remapping
        __disable_irq();
      SCB->VTOR =
      (uint32_t)&vectorTable_RAM; __DSB();
      __enable_irq();

      SystemCoreClockUpdate();
…
}
```

As shown in Figure 81, the image injection can be executed through the USB Loader Tool, which comes together with the **USE***cube*™ Stick.

29 http://www.secube.eu

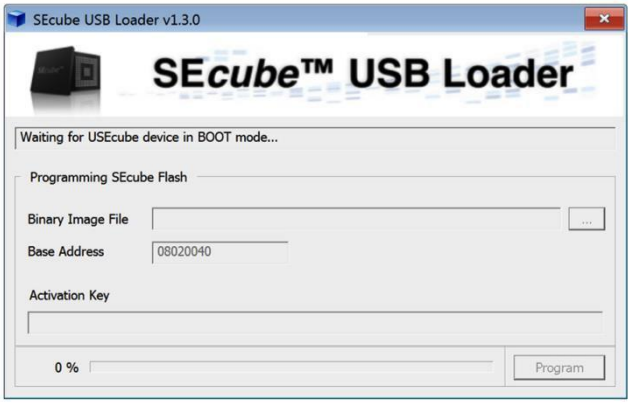


Figure 81 **SE*cube™*** USB Loader

The USB Loader Tool recognizes the **USE*cube*™** Stick in Boot Mode, which means that the firmware image has not been yet injected, and allows developers/users to inject a custom binary image in the internal flash. The base address is parametric and, as stated before, must be coherent with the linker settings.

In order to prevent unauthorized people to inject firmware in your **USE*cube*™** Stick, a unique Activation Key is delivered when the device is acquired.

It is very important that your binary image is well tested to run properly on the final device. It is also suitable for your firmware to support an in-line programming functionality in order to allows users to update the **USE*cube*™** Stick firmware in the future.

APPENDIX A - SE*cube*™ Data Sheet

SE*cube*™ Data Sheet Introduction

August 2015 Data Sheet DUI 15082DS

General Description

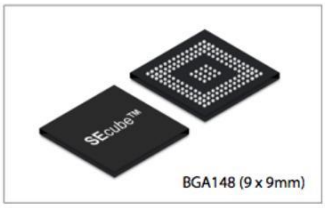
The SE*cube*™ (Secure Environment cube) is a powerful chip which integrates three key security elements in a single package. A fast floating-point Cortex-M4 CPU, a high-performance FPGA and an EAL5+ certified Security Controller (Smart Card).

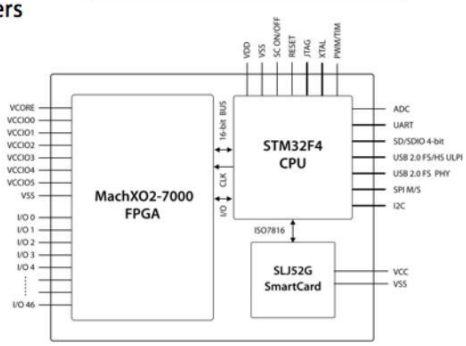
The result of this innovative combination gives an extremely versatile secure environment in a single SoC, in which developers can rapidly implement complex applications and appliances.

The SE*cube*™ chip has multiple embedded communication interfaces. In addition, the internal FPGA provides up to 47 fast I/O (100 MHz) for custom high-speed interface implementations.

This allows fast integration of the SE*cube*™ into any hardware design, while drastically reducing the final BOM.

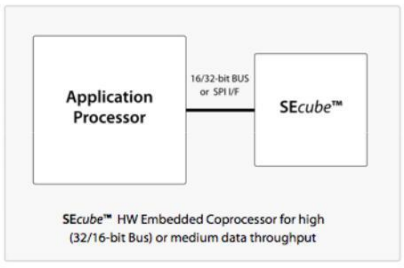
The SE*cube*™ is the ultimate solution for high-end design, delivering integration of a flexible, configurable and certified secure element.



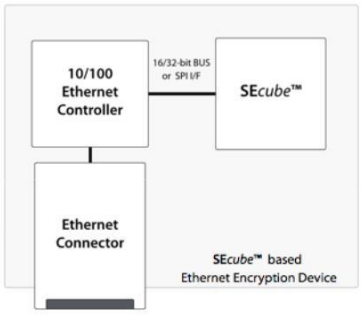


SE*cube*™ Block Diagram

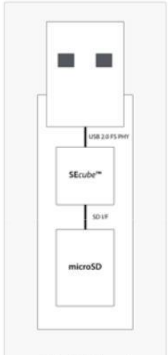
TYPICAL APPLICATION DIAGRAMS



SE*cube*™ HW Embedded Coprocessor for high (32/16-bit Bus) or medium data throughput



SE*cube*™ based Ethernet Encryption Device



SE*cube*™ based USB 2.0 Secure Token

SE*cube*™ Data Sheet Introduction

August 2015 Data Sheet DUI 15082DS

Main Features

Three powerful elements in one chip

■Embedded STM32F4 CPU

- Core: ARM® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 180 MHz, MPU, 225 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories:
  - 2 MB of Flash memory organised into two banks allowing read-while-write
  - 256+4 KB of SRAM including 64-KB of CCM (core coupled memory) data RAM
- Clock, reset and supply management:
  - 1.7 V to 3.6 V application supply and I/Os POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - Internal 32 kHz RC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - VBAT supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- JTAG interface
- 1×12-bit, 2.4 MSPS ADC, 7.2 MSPS in triple interleaved mode
- Up to 17 timers: up to twelve 16-bit and two 32- bit timers up to 180 MHz, 1 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
- 1xSPI (45 Mbits/s) Master/Slave configurable
- 1USART (11.25 Mbit/s, CTS,RTS RS232)
- 1 × I2C interface (SMBus/PMBus)
- 1 x SD/SDIO interface up to 48MHz (SD v4.2, SDIO v2.0), 1bit-4bit modes supported
- True random number generator
- CRC calculation unit
- RTC: sub-second accuracy, hardware calendar
- 96-bit unique ID
- USB Connectivity:
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
- Connections to SmartCard:
  - ISO7816 interface with Clock
  - 1 x GPIO to control external power supply
- Connections to FPGA:
  - 16-bit data, 6-bit address, 100MHz bus SRAM/PRAM mode, 2 x chip selects
  - Master Oscillator pin, up to 90 MHz
  - 5xGPIOs connected to the FGPA JTAG interface for bit-bending programming operations
  - 2xGPIOs for status/polling/interrupt signalling

■ Embedded MachXO2-7000 FPGA

- 6864 LUTs and 47 I/Os
- Ultra Low Power Device (65 nm process, 19 µW standby power, programmable low swing differential I/Os, Stand-by mode and other power saving options)
- Embedded and distributed memory
  - 240 Kbits SysMEM™ embedded blocks RAM
  - 54 Kbits distributed RAM
  - Dedicated FIFO control logic
- 256 Kbits On-Chip User Flash Memory
- Flexible I/O Buffers:
  - (LVCMOS 3.3/2.5/1.8/1.5/1.2, LVTTL, PCI, LVDS, Bus-LVDS, MLVDS, RSDS, LVPECL, SSTL 25/18, HSTL 18,Schmitt trigger input up to 0.5 V hysteresis, etc.)
  - On-chip differential terminations
- Wide Frequency range (10 MHz to 400 MHz)
- Non-Volatile infinitely reconfigurable
- In-field logic configuration while system operates

■ Embedded SLJ52G SECURITY CONTROLLER - SMART CARD

- JavaCard Platform, including ePassport and eSign applets
- ISO7816 Interface
- Supported standards: JC 3.0, GP 2.2, ICAO BAC, SAC, AA, BSI-TR03110 v1.11 EAC, ISO 18013 BAP, EAP config 1-4
- 128 KByte EEPROM
- DES, 3DES, AES up to 256-bit
- RSA up to 2048-bit, ECC up to 521-bit
- Certified Common Criteria CC EAL5+ high

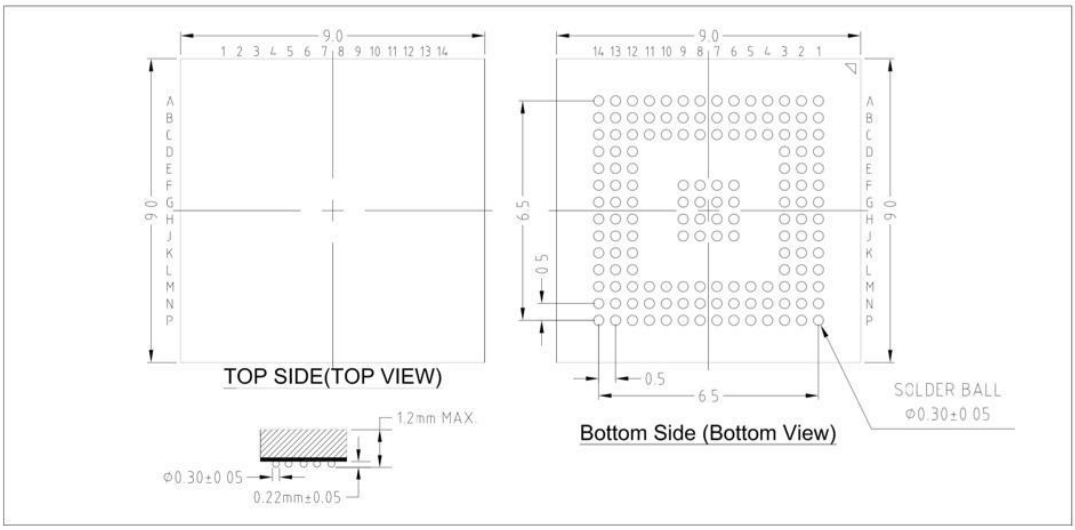
SE*cube*™ Data Sheet Introduction

August 2015 Data Sheet DUI 15082DS

Pinout and Packaging

SE*cube*™ Pinout Table

A1	FPGA_IO_D12	B12	FPGA_IO_D4	E3	CPU_JTAG_TMS	H2	CPU_SDIO_D2	L1	FPGA_IO_CTRL1	N6	FPGA_VCORE
A2	FPGA_IO_CTRL4	B13	FPGA_IO_GP14	E12	CPU_USB_ULPI_D6	H3	CPU_SDIO_CLK	L2	CPU_SC_PWR	N7	CPU_SPI_SSN
A3	FPGA_IO_CTRL2	B14	FPGA_VCORE	E13	CPU_USB_ULPI_D5	H6	VSS	L3	CPU_UART_TX	N8	CPU_USB_ULPI_NXT
A4	FPGA_IO_D9	C1	FPGA_VCCIO0	E14	FPGA_IO_D0	H7	VSS	L12	CPU_USB_ULPI_CLK	N9	CPU_GP1
A5	FPGA_IO_D14	C2	CPU_VCAP2	F1	FPGA_IO_D15	H8	VSS	L13	FPGA_VCORE	N10	CPU_USB_ULPI_STP
A6	FPGA_IO_D7	C3	CPU_VDD	F2	CPU_JTAG_TDI	H9	VSS	L14	FPGA_VCORE	N11	CPU_I2C_SCL
A7	FPGA_IO_GP0	C4	CPU_VDD	F3	CPU_UART_CTS	H12	CPU_VDD	M1	FPGA_IO_CTRL3	N12	CPU_I2C_SDA
A8	FPGA_IO_D6	C5	CPU_UART_RX	F6	VSS	H13	FPGA_IO_GP13	M2	CPU_JTAG_TRST	N13	CPU_USB_ULPI_D0
A9	FPGA_IO_GP6	C6	FPGA_IO_GP3	F7	VSS	H14	FPGA_IO_D1	M3	VSS	N14	VSS
A10	FPGA_IO_GP5	C7	CPU_SDIO_D1	F8	VSS	J1	CPU_USB_ULPI_D7	M4	CPU_VDD	P1	FPGA_VCCIO0
A11	FPGA_IO_GP9	C8	CPU_SDIO_D0	F9	VSS	J2	CPU_VDD	M5	CPU_SPI_CLK	P2	VSS
A12	FPGA_IO_D5	C9	CPU_GP0	F12	CPU_VCAP1	J3	CPU_VDD	M6	CPU_XTAL_IN	P3	FPGA_VCCIO5
A13	FPGA_IO_GP15	C10	FPGA_VCCIO1	F13	CPU_USB_ULPI_D4	J6	VSS	M7	CPU_SPI_MOSI	P4	FPGA_IO_CTRL6
A14	FPGA_VCCIO1	C11	CPU_VDD	F14	FPGA_IO_GP11	J7	VSS	M8	CPU_RSTN	P5	CPU_USB_ULPI_DIR
B1	FPGA_VCCIO1	C12	CPU_VDD	G1	FPGA_VCCIO0	J8	VSS	M9	CPU_ADC	P6	FPGA_VCORE
B2	FPGA_IO_D10	C13	FPGA_VCORE	G2	CPU_JTAG_TCK	J9	VSS	M10	CPU_WKUP	P7	FPGA_VCCIO4
B3	FPGA_IO_CTRL0	C14	FPGA_VCCIO2	G3	CPU_SDIO_D3	J12	CPU_VDD	M11	CPU_VDD	P8	CPU_SPI_MISO
B4	FPGA_IO_D8	D1	FPGA_IO_CTRL13	G6	VSS	J13	FPGA_IO_D2	M12	VSS	P9	FPGA_IO_CTRL8
B5	FPGA_IO_D13	D2	FPGA_VCORE	G7	VSS	J14	FPGA_IO_D3	M13	VSS	P10	FPGA_IO_CTRL9
B6	FPGA_IO_D11	D3	FPGA_VCORE	G8	VSS	K1	FPGA_VCORE	M14	FPGA_VCCIO2	P11	FPGA_IO_CTRL10
B7	FPGA_IO_GP1	D12	CPU_USB_DM	G9	VSS	K2	FPGA_VCORE	N1	SC_VCC	P12	FPGA_IO_CTRL11
B8	FPGA_IO_GP2	D13	CPU_TIMER_PWM	G12	CPU_USB_DP	K3	CPU_JTAG_TDO	N2	FPGA_IO_CTRL5	P13	FPGA_IO_CTRL12
B9	FPGA_IO_GP4	D14	FPGA_IO_GP10	G13	CPU_USB_ULPI_D3	K12	CPU_USB_ULPI_D2	N3	VSS	P14	FPGA_VCCIO3
B10	FPGA_IO_GP8	E1	FPGA_IO_CTRL14	G14	FPGA_IO_GP12	K13	CPU_USB_ULPI_D1	N4	FPGA_IO_CTRL7		
B11	FPGA_IO_GP7	E2	CPU_UART_RTS	H1	CPU_SDIO_CMD	K14	FPGA_VCCIO2	N5	CPU_XTAL_OUT		



SE*cube*™ Packaging information

SE*cube*™ Data Sheet Introduction

August 2015 — Data Sheet DUI 15082DS

Embedded Components Cross-Connections

Refer to the specific component's data sheets for further details.
Power supply signals are directly connected to the relating components.

SE*cube*™ Internal Connections

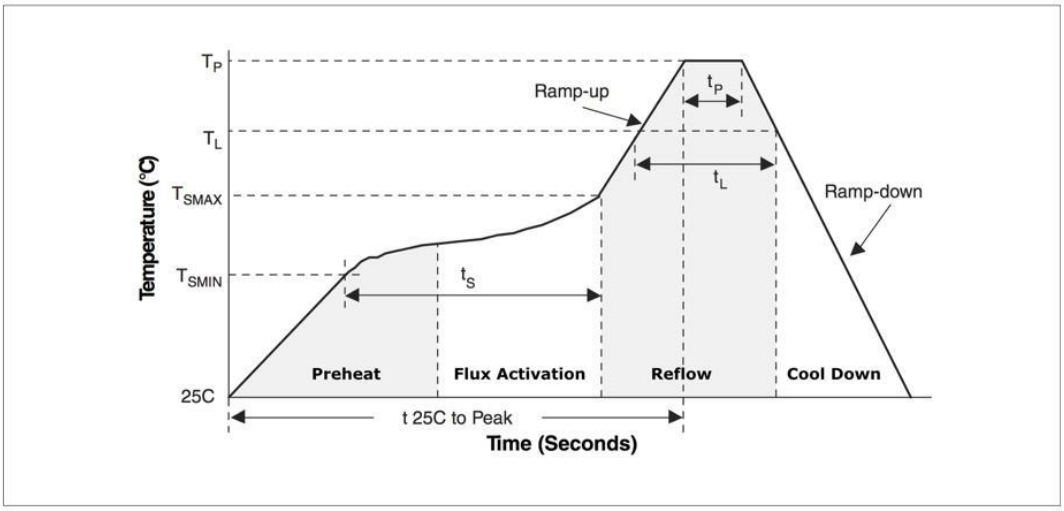
SE*cube*™ Data Sheet Introduction

August 2015 Data Sheet DUI 15082DS

Reflow Profiles

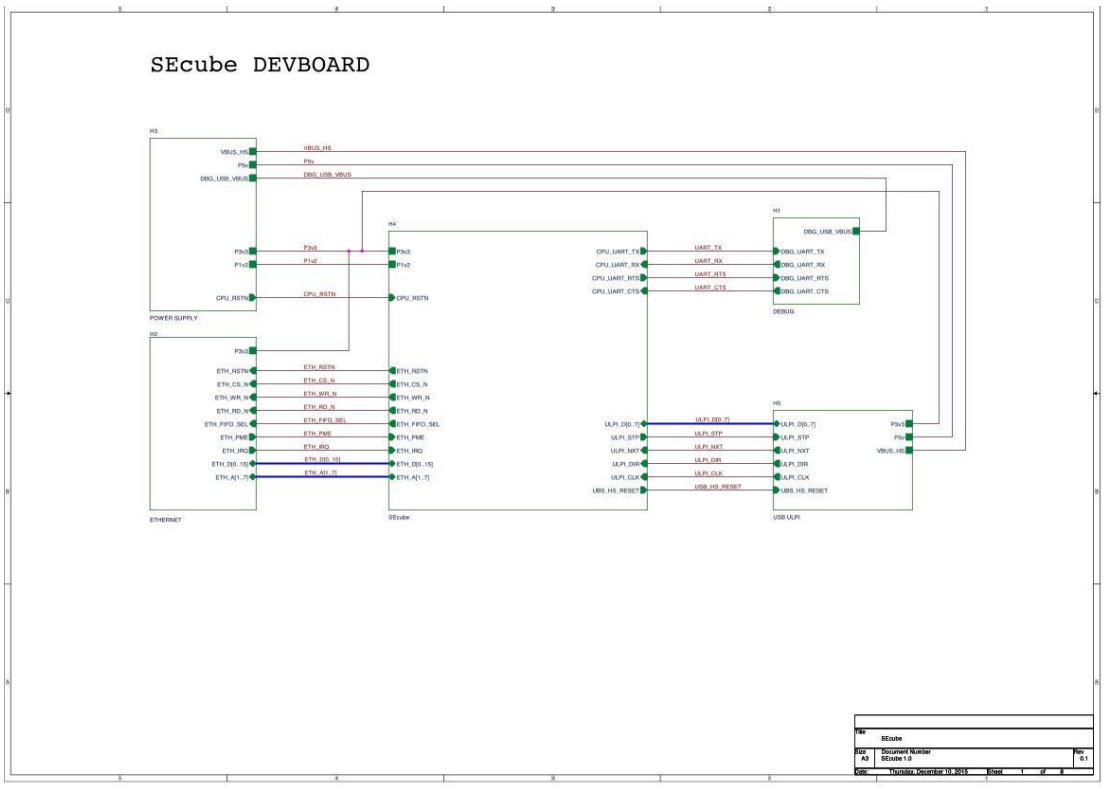
SE*cube*™ Reflow Table

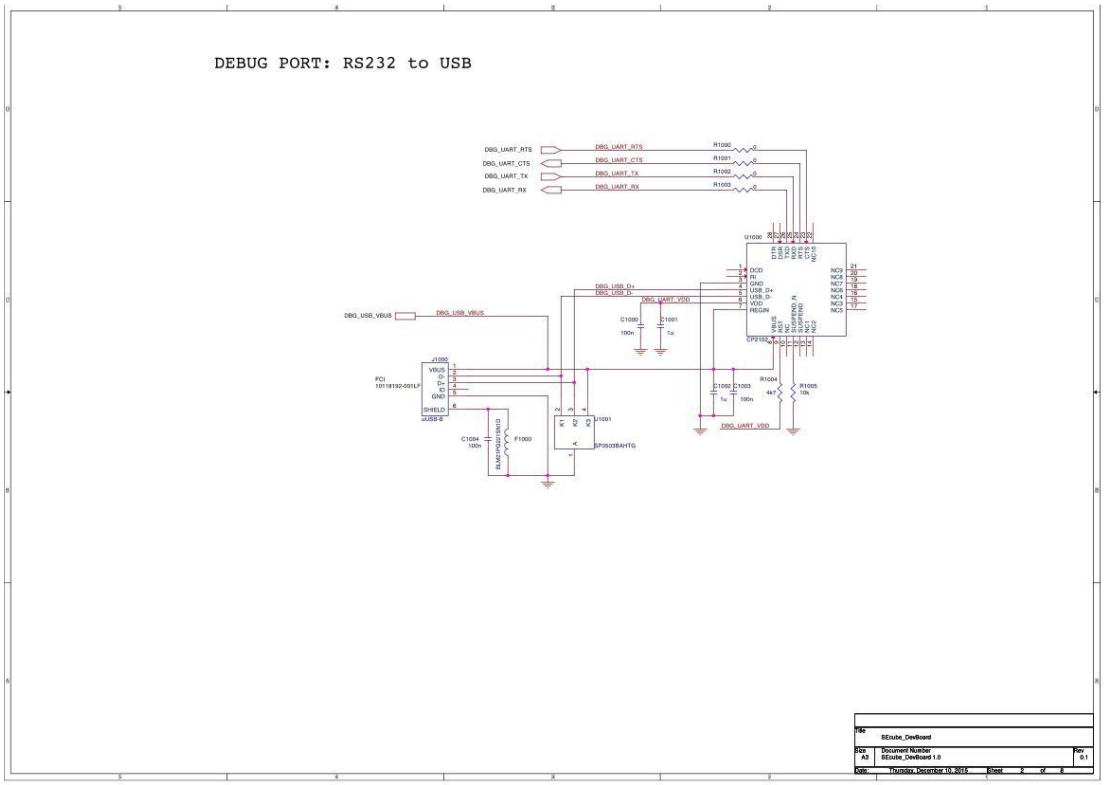
Parameter	Description	Pb-Free and Halogen-Free Packages
Ramp-Up	Average Ramp-Up Rate (T_{SMAX} to T_P)	3 °C/second max.
T_{SMIN}	Preheat Peak Min. Temperature	150 °C
T_{SMAX}	Preheat Peak Max. Temperature	200 °C
t_S	Time between T_{SMIN} and T_{SMAX}	60 seconds–120 seconds
T_L	Solder Melting Point	217 °C
t_L	Time Maintained above T_L	60 seconds–150 seconds
t_p	Time within 5 °C of Peak Temperature	30 seconds
Ramp-Down	Ramp-Down Rate	6 °C/second max.
t 25 °C to T_p	Time from 25 °C to Peak Temperature	8 minutes max.

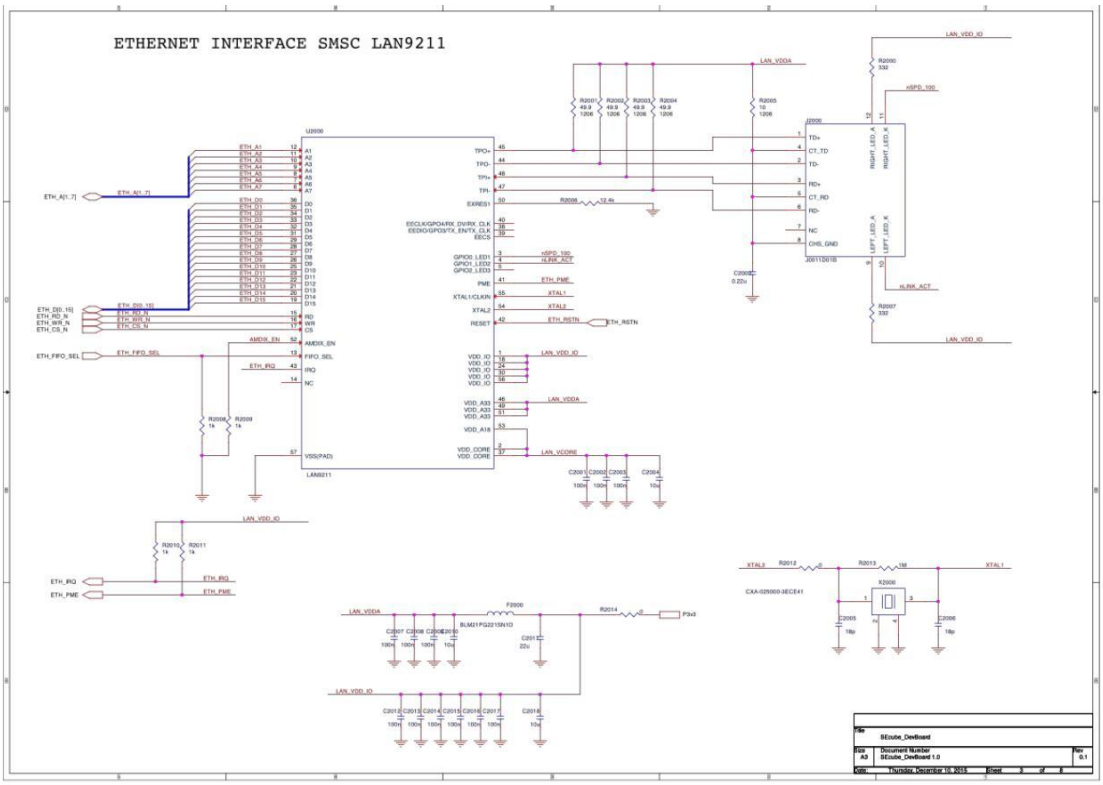


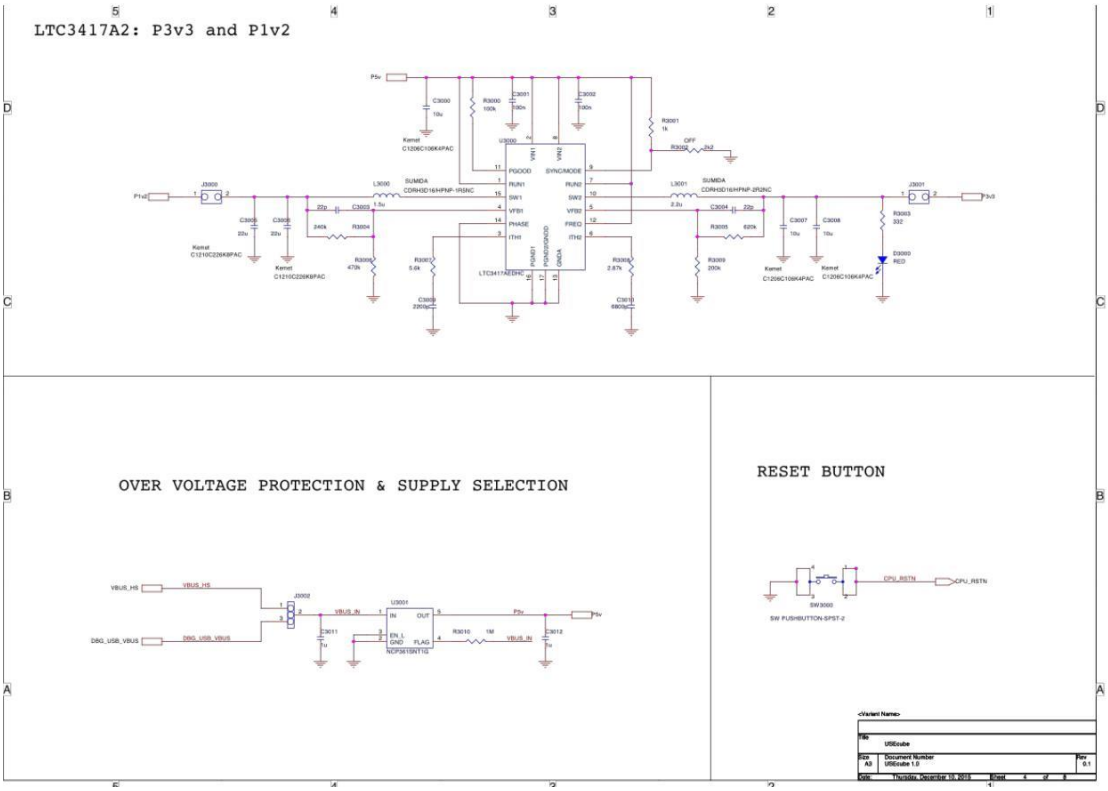
SE*cube*™ Termal Reflow Profile

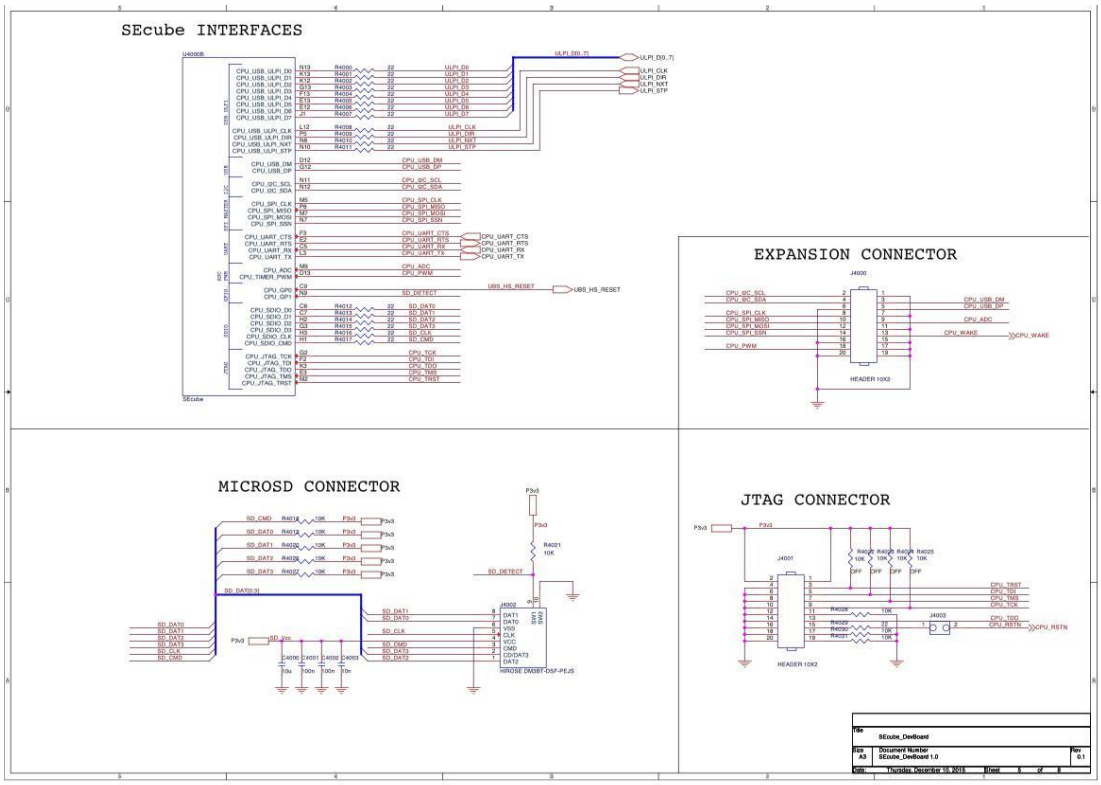
APPENDIX B - SE*cube*™ DevKit Schematics

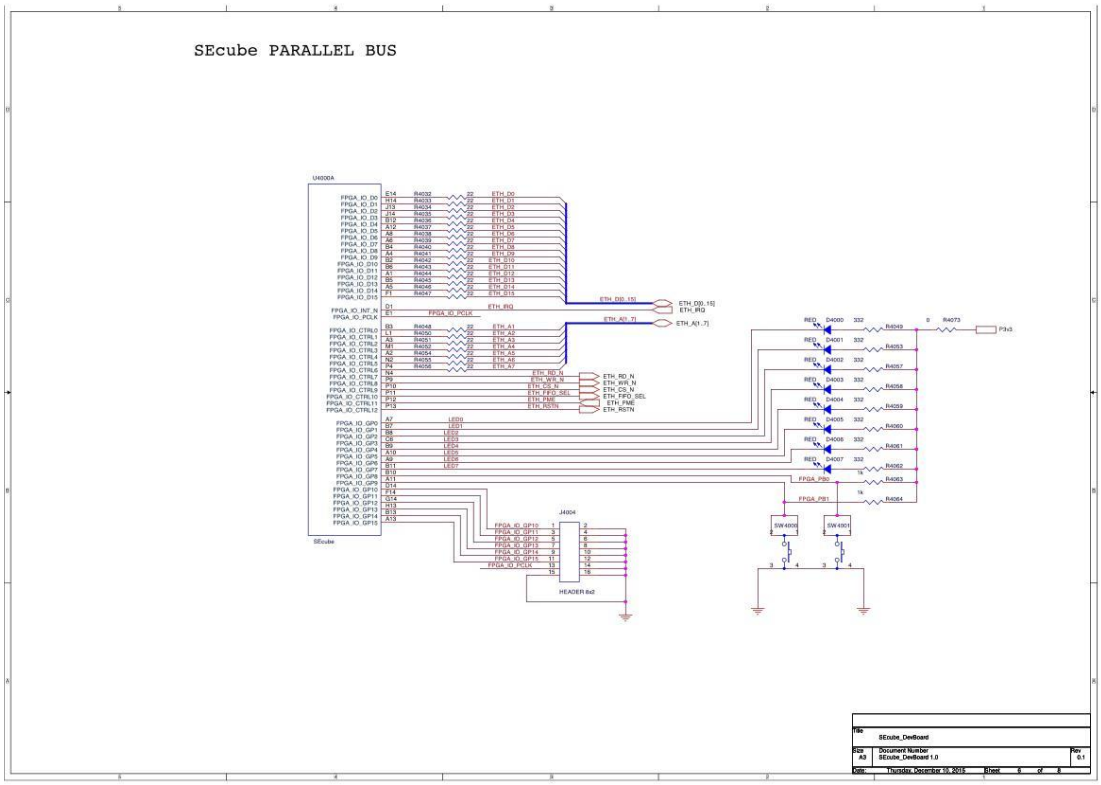


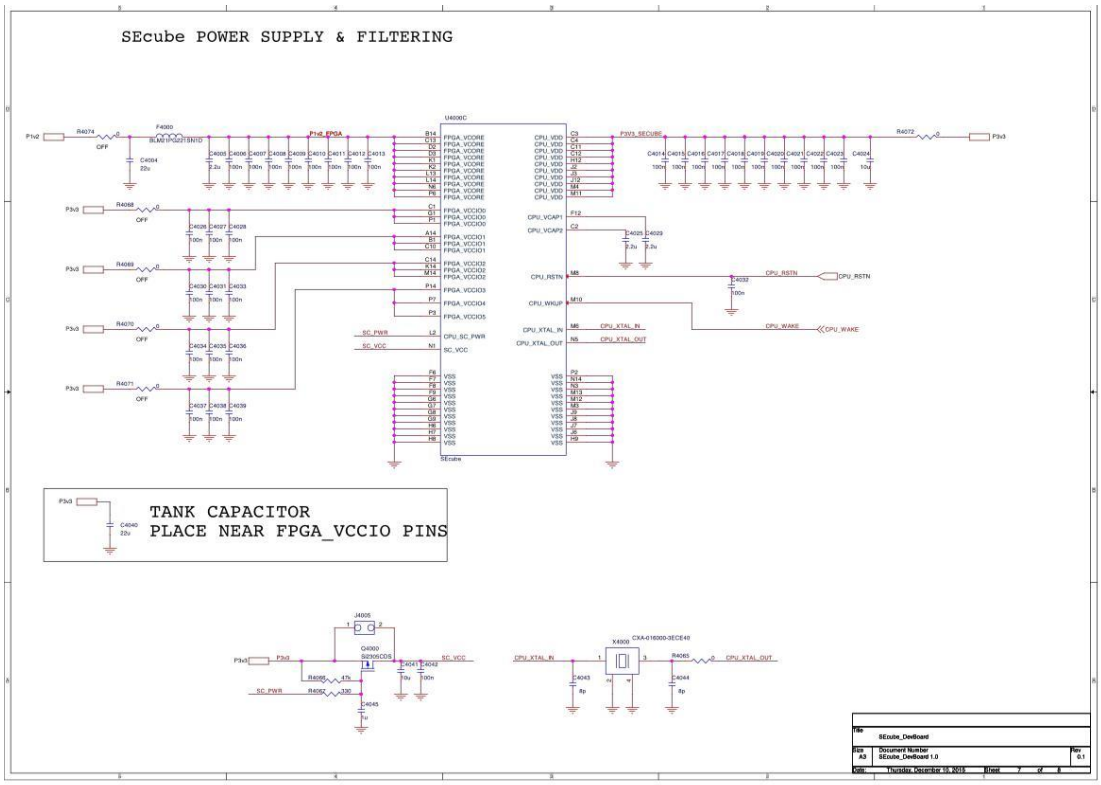


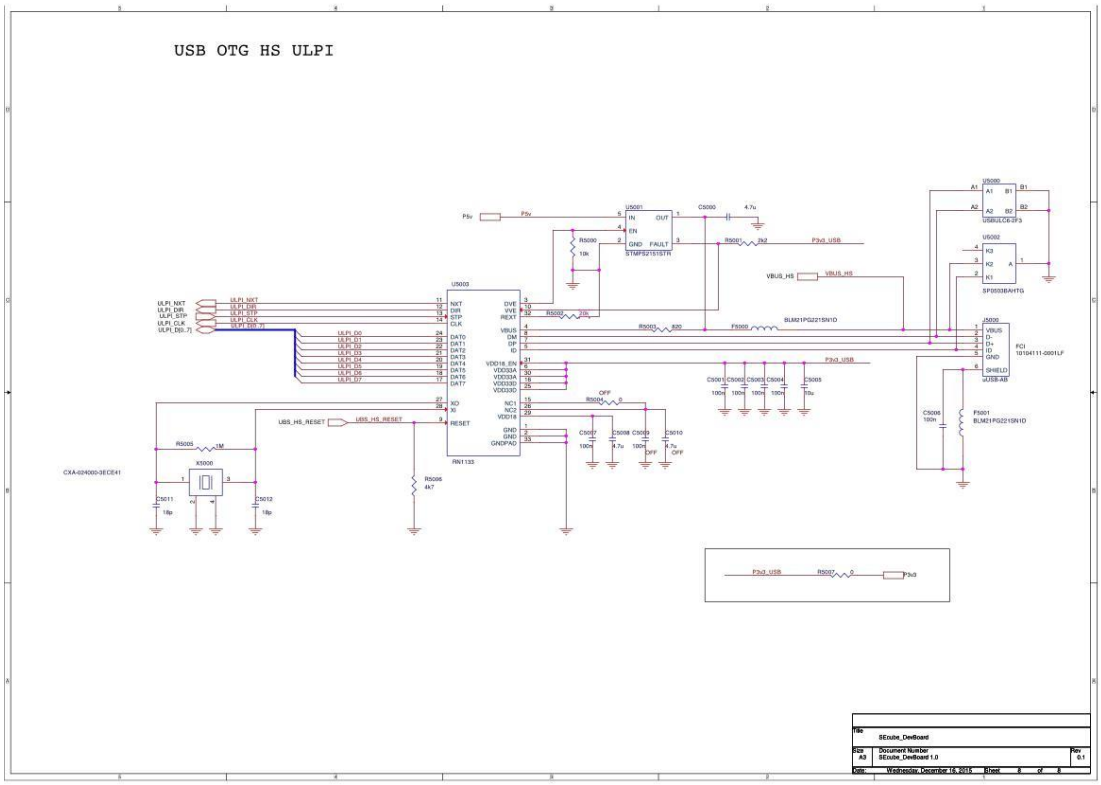

ETHERNET INTERFACE SMSC LAN9211


LTC3417A2: P3v3 and P1v2
OVER VOLTAGE PROTECTION & SUPPLY SELECTION
RESET BUTTON


SEcube INTERFACES
EXPANSION CONNECTOR
MICROSD CONNECTOR
JTAG CONNECTOR


SEcube PARALLEL BUS


SEcube POWER SUPPLY & FILTERING
TANK CAPACITOR
PLACE NEAR FPGA_VCCIO PINS


USB OTG HS ULPI

APPENDIX C – Source code "main.c" for the HelloWorld application

```
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>

#include "secube-host/L1.h"

static uint8_t pin_admin[32] = {
      'a','d','m','i', 'n',0,0,0, 0,0,0,0, 0,0,0,0,
      0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0
};

int main() {
      se3_disco_it it;
      se3_device dev;
      se3_session s;
      uint16_t return_value = 0;
      bool logged_in = false;

      printf("Welcome to the SEcube Hello World application!");
      printf("\n\n\n");
      Sleep(1000);

      printf("Looking for SEcube
      devices...\n"); Sleep(3000);

      L0_discover_init(&it);
      if (L0_discover_next(&it)) {
            printf("SEcube device found!");
            return_value = L0_open(&dev, &(it.device_info),
                                                SE3_TIMEOUT);
      }
      if (return_value != SE3_OK) {
            printf("Error opening device.\nPlease check board
                                              connection.\n\n");
            Sleep(3000);
            return(1);
      }
      else {
            printf("Open Device success\n");
      }

      /* Log in */
      printf("Logging in as
      admin...\n"); Sleep(3000);
      return_value = L1_login(&s, &dev, pin_admin,
                                              SE3_ACCESS_ADMIN);
      if (return_value != SE3_OK) {
```

```
            printf("Error, login failed.\nPlease check security
                                                    pin.\n\n");
            Sleep(3000);
            return(2);
      }
      else {
            printf("Login success\n");
            logged_in = true;
      }
      /* */


      if (logged_in)
            printf("You are logged in SEcube device! Hello
                                                    World!!!");


      /* Log out */
      printf("Logging out…\n");
      Sleep(3000);
      return_value = L1_logout(&s);
      if (return_value != SE3_OK) {
            printf("Error, logout failed.\nPlease check board
                                              connection.\n\n");
            Sleep(3000);
            return(3);
      }
      else {
            printf("Logout success\n");
            printf("\n\nConnection to SEcube device was
                                              successful\n");
            printf("Press [ENTER] to finish\n");
      }
      /* */

      getchar();
      return (0);
}
```

APPENDIX D – Device-Side Libraries

This appendix lists the full set of device-side APIs, provided by the 3 main developed cores (Communication Core, SE3 Core, Dispatcher Core). For more in-depth details please refer to the Doxygen-based documentation available online[30].

Communication Protocol and Provisioning APIs

```
uint16_t echo(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

ECHO command handler. Send back data it receives.

```
uint16_t factory_init(uint16_t req_size, const uint8_t*
req, uint16_t* resp_size, uint8_t* resp);
```

FACTORY_INIT command handler. Initialize device's serial number.

Basic Security APIs

```
uint16_t config(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

Get or set a configuration record. Used to set an User/Admin PIN

```
uint16_t challenge(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

CHALLENGE command handler. Get a login challenge from the device.

```
uint16_t login(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

LOGIN command handler. Respond to challenge and complete the login.

```
uint16_t logout(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

LOGOUT command handler. Log out and release resources.

```
uint16_t crypto_init(uint16_t req_size, const uint8_t*
req, uint16_t* resp_size, uint8_t* resp);
```

CRYPTO_INIT handler. Initialize a cryptographic context.

```
uint16_t crypto_update(uint16_t req_size, const uint8_t*
req, uint16_t* resp_size, uint8_t* resp);
```

CRYPTO_UPDATE handler. Use a cryptographic context.

```
uint16_t crypto_set_time(uint16_t req_size, const uint8_t*
req, uint16_t* resp_size, uint8_t* resp);
```

30 http://www.secube.eu/resources/

CRYPTO_SET_TIME handler. Set device time for key validity.

```
uint16_t crypto_list(uint16_t req_size, const uint8_t*
req, uint16_t* resp_size, uint8_t* resp);
```

Get list of available algorithms supported by the device.

```
uint16_t key_edit(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

Insert, delete or update a key.

```
uint16_t key_list(uint16_t req_size, const uint8_t* req,
uint16_t* resp_size, uint8_t* resp);
```

Get a list of keys available in the device.

APPENDIX E – Host-Side Software

This appendix lists the full set of host-side L0 and L1 APIs. For more in-depth details please refer to the Doxygen-based documentation available online[31].

Communication Protocol and Provisioning APIs (Level0 – L0)

```
uint16_t L0_TXRX(se3_device* device, uint16_t req_cmd,
uint16_t req_cmdflags, uint16_t req_len, const uint8_t*
req_data, uint16_t* resp_status, uint16_t* resp_len, uint8_t*
resp_data);
```

Main function for communicating with SEcube(tm) device. It sends a packet of data containing a command and read the response written by the **SE*cube*™**.

```
uint16_t L0_echo(se3_device* device, const uint8_t*
data_in, uint16_t data_in_len, uint8_t* data_out);
```

Echo service. Send a packet of data to the device which should reply with the same data. Login is not required.

```
uint16_t L0_factoryinit(se3_device* device, const uint8_t*
se-rialno);
```

Initialise the **SE*cube*™** with the Serial Number to be set on it. Before using the **SE*cube*™** de-vice, this function must be called and can be used just once.

```
uint16_t L0_open(se3_device* dev, se3_device_info*
dev_info, uint32_t timeout);
```

Open the **SE*cube*™**.

```
void L0_close(se3_device* dev);
```

Close the **SE*cube*™**.

```
bool L0_discover_serialno(uint8_t* serialno, se3_device_info*
device);
```

Discover Serial Number information of the **SE*cube*™**.

```
void L0_discover_init(se3_disco_it*
it);
```
Initialize discovery iterator.

```
bool L0_discover_next(se3_disco_it*
it);
```
Increment discovery iterator.

31 http://www.secube.eu/resources/

Basic Security APIs (Level1 – L1)

```
uint16_t L1_login(se3_session* s, se3_device* dev, const
uint8_t* pin, uint16_t access);
```

This function is used to let a user/admin login on the **SE*cube*™**. Before issuing any command to the **SE*cube*™**, login is required. Default user/admin PIN are set to all zeroes.

```
uint16_t L1_set_admin_PIN(se3_session* s, uint8_t* pin);
```

This function is used to change the current admin pin.

```
uint16_t L1_set_user_PIN(se3_session* s, uint8_t*
pin);
```

This function is used to change the current user pin.

```
uint16_t L1_logout(se3_session* s);
```

This function is used to logout from the **SE*cube*™** device.

```
uint16_t L1_key_list(se3_session* s, uint16_t skip,
uint16_t max_keys, se3_key* key_array, uint16_t* count);
```

This function is used get the list of the keys inserted into the **SE*cube*™** device.

```
uint16_t L1_key_edit(se3_session* s, uint16_t op, se3_key* k);
```

This function is used to edit the keys data on the **SE*cube*™** device. Depending on op, it can insert/modify a key or delete it.

```
bool L1_find_key(se3_session* s, uint32_t key_id);
```

Find a key into the **SE*cube*™** device. It can be used to check if a key is present or not.

```
uint16_t L1_crypto_init(se3_session* s, uint16_t algorithm,
uint16_t mode, uint32_t key_id, uint32_t* sess_id);
```

Initialise a crypto session.

WARNING: data length for input and output must be a multiple of the crypto block size. The length of the input buffer is computed with the macro ENC_SIZE(buffer_len). The output buffer is allocated with that length but the actual length is an output of the function.

```
uint16_t L1_crypto_update(se3_session* s, uint32_t sess_id,
uint16_t flags, uint16_t data1_len, uint8_t* data1,
uint16_t data2_len, uint8_t* data2, uint16_t* dataout_len,
uint8_t* data_out);
```

Update a crypto session.

WARNING: data length for input and output must be a multiple of the crypto block size. The length of the input buffer is computed with the macro ENC_SIZE(buffer_len). The output buffer is allocated with that length but the actual length is an output of the function.

```
uint16_t L1_crypto_set_time(se3_session* s, uint32_t
devtime);
```

Set time for a crypto session.

WARNING: This function must be invoked right after any login operation, otherwise all of the subsequent operations cyphering will fail.

```
uint16_t L1_encrypt(se3_session* s, uint16_t algorithm,
uint16_t mode, uint32_t key_id, size_t datain_len, int8_t*
data_in, size_t* dataout_len, uint8_t* data_out);
```

This function is used to encrypt a buffer of data given the algorithm, the encryption mode, the buffer size. The encrypt data is returned.

```
uint16_t L1_decrypt(se3_session* s, uint16_t algorithm,
uint16_t mode, uint32_t key_id, size_t datain_len, int8_t*
data_in, size_t* dataout_len, uint8_t* data_out);
```

This function is used to decrypt a buffer of data given the algorithm, the decryption mode, the buffer size. The decrypted data is returned.

```
uint16_t L1_digest(se3_session* s, uint16_t algorithm, size_t
datain_len, int8_t* data_in, size_t* dataout_len, uint8_t*
data_out);
```

This function is used to compute the digest or sign a buffer of data.

```
uint16_t L1_get_algorithms(se3_session* s, uint16_t skip,
uint16_t max_algorithms, se3_algo* algorithms_array,
uint16_t* count);
```

This function is used to retrieve a list algorithms supported by the **SE*cube*™**.