***Efficient Embedded Course***

**LAB 9**

**TRUSTZONE EXERCISE:**

**SECURE AND NON-SECURE BLINK**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83369550)

[1.1 Lab overview 1](#_Toc83369551)

[2 Learning Objectives 2](#_Toc83369552)

[3 Requirements 2](#_Toc83369553)

[3.1 Software requirements 2](#_Toc83369554)

[3.2 Hardware requirements 2](#_Toc83369555)

[4 Setting up the project 3](#_Toc83369556)

[4.1 Partitioning the memory into secure and non-secure banks using STM32Cube Programmer 3](#_Toc83369557)

[4.2 Creating the project using STM32Cube MX 5](#_Toc83369558)

[5 Programming in Keil 7](#_Toc83369559)

[5.1 Project\_S 8](#_Toc83369560)

[5.1.1 Define starting address in main.c 8](#_Toc83369561)

[5.2 Project\_NS 9](#_Toc83369562)

[5.2.1 Configuring the settings for Project\_NS 10](#_Toc83369563)

[5.3 Back to Project\_S 12](#_Toc83369564)

[5.3.1 Configuring the settings for Project\_S 12](#_Toc83369565)

[5.4 Building and loading the project 13](#_Toc83369566)

[5.5 Running and understanding the program 14](#_Toc83369567)

[5.5.1 Running Project\_NS 14](#_Toc83369568)

[5.5.2 Running Project\_S 15](#_Toc83369569)

# Introduction

## Lab overview

In this lab, you will implemente a project utilising Arm’s TrustZone feature. TrustZone is a special feature available in newer Arm Cortex M microcontrollers such as the Arm Cortex M33 in the STM32 Nucleo-L552ZEQ board used in this lab. TrustZone enables the user to partition the board ‘Secure’ and ‘Non-Secure’ memory banks and use the memory bank to run different parts of a program on each bank. This reduces the potential of unwanted breaches of sensitive data/program features by isolating them from the rest of the program.

When TrustZone is enabled, program elements from the ‘Secure’ partition are able to call program elements (i.e. function calls, global variables) from the ‘Non-Secure’ partition; However, elements from the ‘Non-Secure’ partition are unable to access program elements from the ‘Secure’ partition.

In this lab, we will test these concept using a simple program that assigns LEDs ‘Secure’ and ‘Non-Secure’ modes and calling them to blink from both the ‘Secure’ and ‘Non-Secure’ memory banks and observe what happens.

The lab is divided into two parts. In Part 1, we will use the tools (STM Programmer and STM CubeMx) provided by STM Microelectronics to do the following:

* Partition the memory into two banks – i.e. secure and non-secure memory banks using STM32Cube Programmer
* Configure the Nucleo-L552ZEQ board using the STM32Cube Programmer
* Set up the project using the STM32Cube MX
* Export the program with all the settings to KEIL MDK using STM32Cube MX

In Part 2, we will use KEIL MDK to:

* Program the board
* Observe the result on the debug window

A picture containing diagram

Description automatically generated

Fig 1: Arm TrustZone project summary

# Learning Objectives

* Partitioning the memory into secure and non-secure banks
* Create secure and non-secure world projects and configure their settings
* Assign security levels to IO peripherals
* Run simple program to call function from secure world to non-secure world

# Requirements

## Software requirements

* **Keil µVision5 MDK IDE**
  + Please check the Getting Started with Keil guide on how to download and install it.
* **STM32Cube Programmer**
  + For more information and download, click [here](https://www.st.com/en/development-tools/stm32cubeprog.html).
* **STM32Cube MX**
  + For more information and download, click [here](https://www.st.com/en/development-tools/stm32cubemx.html).

## Hardware requirements

* **STM32 Nucleo-L552ZE-Q board**
  + For more information, click [here](https://www.st.com/en/evaluation-tools/nucleo-l552ze-q.html).

# Setting up the project

## Partitioning the memory into Secure and Non-secure banks using STM32Cube Programmer

1. Connect the board to your PC and open the STM32Cube Programmer
2. Select the ST-LINK option from the drop-down menu and click connect
3. Click the OB (Option Bytes) tab on the left. TrustZone is disabled at default, so we need to enable it.

A screenshot of a computer

Description automatically generated with medium confidence

**1**

**2**

**3**

Fig 2: STM32Cube Programmer

1. Click User Configuration, and check DBANK to enable Dual Bank mode. This will allow us to partition the memory in 2 banks - one Secure bank, and one Non-secure bank. Click Apply once you have picked the correct options.

A screenshot of a computer

Description automatically generated with medium confidence

Fig 3: Option Bytes Configuration

1. Under Secure Area 1 and Secure Area 2, we need to define the start and the end address of the two banks. In the project, the addresses are chosen as below. Once done, click apply.

* In this lab, Area 1 will be used as the ‘Secure’ memory bank while Area 2 will be used for the ‘Non-secure’ memory bank.

Graphical user interface, application, Word

Description automatically generated

Fig 4: Memory Address Configuration

1. Click Disconnect.

## Creating the project using STM32Cube MX

1. Open STM32Cube MX and click ‘ACCESS TO BOARD SELECTOR’.

Graphical user interface, website

Description automatically generated

Fig 5: STM32Cube MX Home page

1. Under the ‘Board Selector’ tab, search for ‘NUCLEO-L552ZE-Q’. Select the board and click ‘Start Project’.

Graphical user interface, application

Description automatically generated

Fig 6: Board selector settings

1. If asked ‘Initialize all peripherals with their default Mode?’, click ‘Yes’.
2. If asked ‘Do you want to create a new project :’, select the ‘with TrustZone activated ?’ option and click ‘OK’.
3. For this project, we are using three on board LEDs, green Led (PC7) for Secure world, blue Led (PB7) and red Led (PA9) for Non-secure world.

Graphical user interface, text, application

Description automatically generated

Fig 7: Pinout view

1. Under system core, select GPIO, and change Pin Context Assignment for all the 3 LED pins - ‘Cortex-M33 Secure’ for the green LED pin and ‘Cortex-M33 Non-secure’ for the red and blue LED pins.

* This means that the green LED can only be accessed by program elements in the ‘Secure’ memory bank while the blue and red LEDs can be accessed from both the ‘Secure’ and ‘Non-secure’ memory banks.

Graphical user interface, application

Description automatically generated

Fig 8: GPIO settings

1. Navigate to the ‘Project Manager’ tab. Under the ‘Project’ section, select ‘MDK-ARM’ under the ‘Toolchain / IDE’ setting. Name the project and its location as you please.

Graphical user interface, text

Description automatically generated

Fig 9: Project Manager settings

1. Click ‘Generate Code’.
2. Open the project with KEIL MDK.

# Programming in Keil

In Keil, notice that the project is divided into two projects – **project**\_S and **project**\_NS, corresponding to the program in the Secure and Non-secure memory banks respectively. (Note that **project** is just a place holder for the actual name of the project)

Graphical user interface, text, application, email

Description automatically generated

Fig 10: Project overview in Keil

## Project\_S

### Define starting address in main.c

In ‘main.c’ file under **project**\_S, make sure to define the correct starting address for the Non-secure memory bank.

Graphical user interface, text, application, email

Description automatically generated

Fig 11: Non-secure starting address

Notice the function that jumps from the Secure bank to the Non-secure bank as seen in the ***NonSecure\_Init*** function in line 129.

Graphical user interface, text, application, email

Description automatically generated

Fig 12: ‘NonSecure\_Init’ function

## Project\_NS

The main.c file in Project\_NS is for the Non-secure bank. Here the blue and red LEDs are toggled at a certain rate.

Graphical user interface, text, application

Description automatically generated

Fig 12: None-secure ‘main.c’ file

1. Right-click on ‘Project **project**\_ns’ and ‘Set as Active Project’.
2. Click the ‘Options for Target…’ button.

Graphical user interface, text, application

Description automatically generated

Fig 13: ‘Option for Target…’ button

### Configuring the settings for Project\_NS

1. Make sure the IROM1 and IRAM1 addresses are the same as below.

* Notice how the address is covered by the addresses that was defined for ‘Secure Area 2’ in the STM32Cube Programmer is section 4.1.

Graphical user interface

Description automatically generated

Fig 14: IROM1 and IRAM1 Memory Areas

1. Navigate to the ‘Debug’ tab and select the ‘ST-Link Debugger’. Then, click ‘Settings’.

Graphical user interface, text, application

Description automatically generated

Fig 15: Debug configurations

1. Navigate to the ‘Flash Download’ tab.
2. Make sure to add the following ‘On-chip Flash’ and the correct ‘RAM for Algorithm’ start address and size. Click ‘OK’ and ‘OK’ again.

* Again notice how the memory locations are covered by the addresses that were defined for ‘Secure Area 2’ in the STM32Cube Programmer is section 4.1.

Graphical user interface, text, application

Description automatically generated

Fig 16: Flash download configurations

## Back to Project\_S

1. Now set **project**\_s as the active project and click the ‘Options for Target…’ button.

### Configuring the settings for Project\_S

1. Configure IROM2 and IRAM2 as follows.

Graphical user interface

Description automatically generated

Fig 17: IROM2 and IRAM2 Memory Areas for the Secure bank

1. Navigate to the ‘Debug’ tab and select ‘ST-Link Debugger’. Then, click ‘Settings’.

Graphical user interface, text, application

Description automatically generated

Fig 18: Debug configurations

1. Make sure to set the following settings. Once finished, navigate to the ‘Flash Download’ tab.

Graphical user interface

Description automatically generated

Fig 19: Debug window

1. Make sure to add the following ‘On-chip Flash’ and the correct ‘RAM for Algorithm’ start address and size. Click ‘OK’ and ‘OK’ again.

Graphical user interface, text, application

Description automatically generated

Fig 20: Flash download configurations

## Building and loading the project

1. Make **project**\_NS as the active project.
2. Under the ‘Projects’ tab, click Batch Setup and build both projects.

Graphical user interface, text, application

Description automatically generated

Fig 21: Batch Setup

1. Click the ‘Load’ button to flash the board.

## Running and understanding the program

### Running Project\_NS

1. Click the ‘Start/Stop Debug Session’ button.

Graphical user interface, text, application

Description automatically generated

Fig 22: Debug window

1. Click the ‘Run’ button.

* You will notice that only the blue and red LEDs will start to blink but not the green LED. This is because the green LED can only be toggled from the ‘Secure’ memory bank.

Graphical user interface, text, application

Description automatically generated

### Running Project\_S

1. Exit the Debug Session by clicking on the ‘Start/Stop Debug Session’ button again.
2. Set **project**\_S as the active project.
3. Batch rebuild the project.
4. Click the ‘Start/Stop Debug Session’ button to enter the Debug Session.
5. Click the ‘Run’ button to run the program.

* Once you clicked ‘Run’, you notice that all three LEDs are now blinking. This is because the green LED is now being toggled by part of the program in the ‘Secure’ memory partition.