Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Getting Started with NUCLEO-N657X0-Q AI From Scratch

Lukáš Kohout, Zdeněk Pohl
kohoutl@utia.cas.cz, zdenek.pohl@utia.cas.cz

Revision history

Rev.	Date	Author	Description
0	31.08.2026	L. Kohout	Initial draft
1	08.10.2026	L. Kohout	XSPI HSLV
2			

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved.

Contents

1 Introduction 1

2 Sources 1

3 Required Tools 1

4 AI Model 2

5 Creating a Project for STM32CubeIDE 5

6 Modifying the source code of the STM32CubeIDE project 10

7 Project Compilation in STM32CubeIDE 10

8 Project Debugging in STM32CubeIDE 13

8.1 Create the *NUCLEO-N657X0-Q*_FSBL Debug Configuration 13

8.2 Configure Application Loading and Symbols 14

8.3 Start the Debugging Session 16

8.4 Troubleshooting 17

9 Programming NUCLEO-N657X0-Q 18

10 XSPI High Speed Mode 19

10.1 Setting up HSLV with the Complete Process 20

10.2 Setting up HSLV from the Precompiled Executable Binary 22

10.3 Performance Comparison 22

11 Package contents 23

Acknowledgement

This project has received funding from the European Union's Horizon Europe research and innovation programme under the HORIZON-JU-Chips-2024-1-IA grant agreement No 101194172. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or CHIPS JU. Neither the European Union nor the granting authority can be held responsible for them.

The project is also supported by its members France, Italy, Czech Republic, Germany, Sweden, Danmark, Greece, Spain, Netherland, Portugal, Turkey, Switzerland.

National funding in the Czech Republic:

This project has also received national funding from the Ministry of Education, Youth and Sports of the Czech Republic (MEYS) under grant agreement No 9A25004.





© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

1 Introduction

This application note provides a complete, reproducible workflow for building, deploying, debugging, and programming an AI application on the NUCLEO-N657X0-Q from scratch. Using a quantized INT8 neural network that approximates the sine function, it covers the required software tools and tested versions, model creation and conversion to TensorFlow Lite, project generation in STM32Cube AI Studio, source-code integration in STM32CubeIDE, NPU execution verification, signed FSBL and application binaries, debugger configuration, troubleshooting, and final board programming on Linux and Windows. The document emphasizes practical configuration details, known version constraints, reproducible file paths and commands, and validation of the resulting inference on the target hardware.

2 Sources

1. https://community.st.com/t5/stm32-mcus/how-to-build-an-ai-application-from-scratch-on-the-stm32n6570-dk/ta-p/825591
2. https://github.com/STMicroelectronics/stm32ai-modelzoo-services
3. https://www.st.com/en/development-tools/stm32n6-ai.html

3 Required Tools

List of all tools needed to build an AI application on the NUCLEO-N657X0-Q from scratch.

- **STM32CubeIDE**, tested version 2.2.0,

  https://www.st.com/en/development-tools/stm32cubeide.html

  Editing, compiling and debugging projects for ST components.

- **STM32CubeMX**, tested version 6.17.0,

  https://www.st.com/content/st_com/en/stm32cubemx.html

  STM32CubeMX allows programmer to create a project for the **STM32CubeIDE** tool for a specific ST component, user can enable and configure individual peripherals, IO pins, clock generators, etc. in detail. In this guide the tool is used internally by the **STM32Cube AI Studio**.

  NOTE: version 6.17.0 is not the latest version as of June 30, 2026. The latest version as of that date is 6.18.0, but the procedure described here does not work for it, the example does not run on the board, there is some problem with the drivers in FSBL.

- **STEdgeAI-Core**, tested version 4.0.1,

  https://www.st.com/en/development-tools/stedgeai-core.html

  AI library management tool from ST.

- **STM32CubeProgrammer**, tested version 2.23.0,

  https://www.st.com/en/development-tools/stm32cubeprog.html

  ST component programmer.

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

- **STM32Cube AI Studio**, tested version 1.3.0,

  https://www.st.com/content/st_com/en/campaigns/edge-ai-toolchain-for-mcus-z14.html

  Creating a project for **STM32CubeIDE** from an AI model. It uses all the above tools, if their version changes, by update for example, the path to the new tool can be set in the tool settings.

- **Python,** tested version 3.12.3 (Ubuntu 24.04)

  Creating (definition, training, testing, compilation … tflite, onnx, …) an AI model. It is necessary to adhere to Python version 3.12 due to support for the *tensorflow* library.

  - Linux: In the tested version of Ubuntu 24.04, no action is required, Python in the required version is an integral part of the distribution here.
  - Windows: Go to Microsoft Store, search for and install Python version 3.12. You will also need to enable long paths on Windows, in PowerShell type the following command:

```
New-ItemProperty `
-Path "HKLM:SYSTEM\CurrentControlSet\Control\FileSystem" `
-Name "LongPathsEnabled" -Value 1 -PropertyType DWORD -Force
```

- **srec_cat**, tested version 1.64.D001 (Ubuntu 24.04)
  - Linux terminal:

```
sudo apt install srecord
```

  - Windows:

    https://sourceforge.net/projects/srecord/files/srecord-win32/1.65/

  For the STM32N6 component, the FSBL, application, and model weights are typically distributed across multiple binary files. Using **srec_cat** is strongly recommended to simplify programming by concatenating them into a single programming file.

4 AI Model

The AI model is created using Python, i.e. its definition, training, quantization and translation into *tflite* format. For simplicity, the sine function approximation was chosen. The necessary Python libraries will be installed in the local environment so that they do not affect the system in any way.

1. Create a *sin_model* folder in our working directory *$your_work* and go to it.
2. Set up the local environment for Python, Linux terminal / Windows cmd.

```
python3 -m venv tf-env
```

   This step is only performed the first time, it creates the *tf-env* directory, which will contain all local settings and installations of necessary libraries.

3. Activate local Python environment.
   - Linux terminal:

```
source tf-env/bin/activate
```

   - Window cmd:

```
tf-env\Scripts\activate.bat
```

   Successful activation is indicated by the command line starting with the name of the selected environment (tv-env).

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

4. Install all necessary libraries. This step only needs to be done once for the given local environment, the libraries will be installed into the *tf-env* directory and will remain there.

```
pip3 install --upgrade pip
pip3 install numpy
pip3 install tensorflow
```

5. Create Python code to implement the model. The sine-function approximation was chosen for simplicity. The number of inputs and fully connected layers was intentionally set higher than necessary to prevent the model analyzer from automatically switching the implementation from the NPU to the CPU. We will put the following code in the file *sin_model_int8.py*. The file is also available in the attached package.

```
import numpy as np
import tensorflow as tf

# -----------------------------------------------------------
# training data
# -----------------------------------------------------------
NUM_SAMPLES = 10000

x = np.random.uniform(
    -3.14,
    3.14,
    size=(NUM_SAMPLES, 8)
).astype(np.float32)

# The net learns the sine of the first element
y = np.sin(x[:, 0]).astype(np.float32)

# -----------------------------------------------------------
# model definition
# -----------------------------------------------------------
model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(8,)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(1)
])

model.compile(
    optimizer='adam',
    loss='mse'
)

# -----------------------------------------------------------
# run training
# -----------------------------------------------------------
model.fit(
    x,
    y,
    epochs=50,
    batch_size=128,
    verbose=1
)

# -----------------------------------------------------------
# Test
# -----------------------------------------------------------
test = np.array([
    [1.0, 0, 0, 0, 0, 0, 0, 0]
], dtype=np.float32)
```


© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
pred = model.predict(test, verbose=0)

print("sin(1.0) =", np.sin(1.0))
print("NN       =", pred[0][0])

# ------------------------------------------------------------
# Representative dataset for INT8 calibration
# ------------------------------------------------------------
def representative_dataset():
    for i in range(1000):
        yield [x[i:i+1]]

# ------------------------------------------------------------
# INT8 quantization
# ------------------------------------------------------------
converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations             = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset    = representative_dataset
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type      = tf.int8
converter.inference_output_type     = tf.int8

tflite_model = converter.convert()

with open("sin_model_int8.tflite", "wb") as f:
    f.write(tflite_model)

print("Save sin_model_int8.tflite")

# ------------------------------------------------------------
# Checking the resulting model
# ------------------------------------------------------------
interpreter = tf.lite.Interpreter(
    model_path="sin_model_int8.tflite"
)

interpreter.allocate_tensors()

print("\nINPUT:")
print(interpreter.get_input_details())

print("\nOUTPUT:")
print(interpreter.get_output_details())
```

6. Model Compilation
   - Linux terminal:

     ```
     python3 ./sin_model_int8.py
     ```

   - Windows cmd:

     ```
     python sin_model_int8.py
     ```

   This step creates a quantized model and saves it to the file *sin_mode_int8.tflite*.

7. Exit the local Python environment

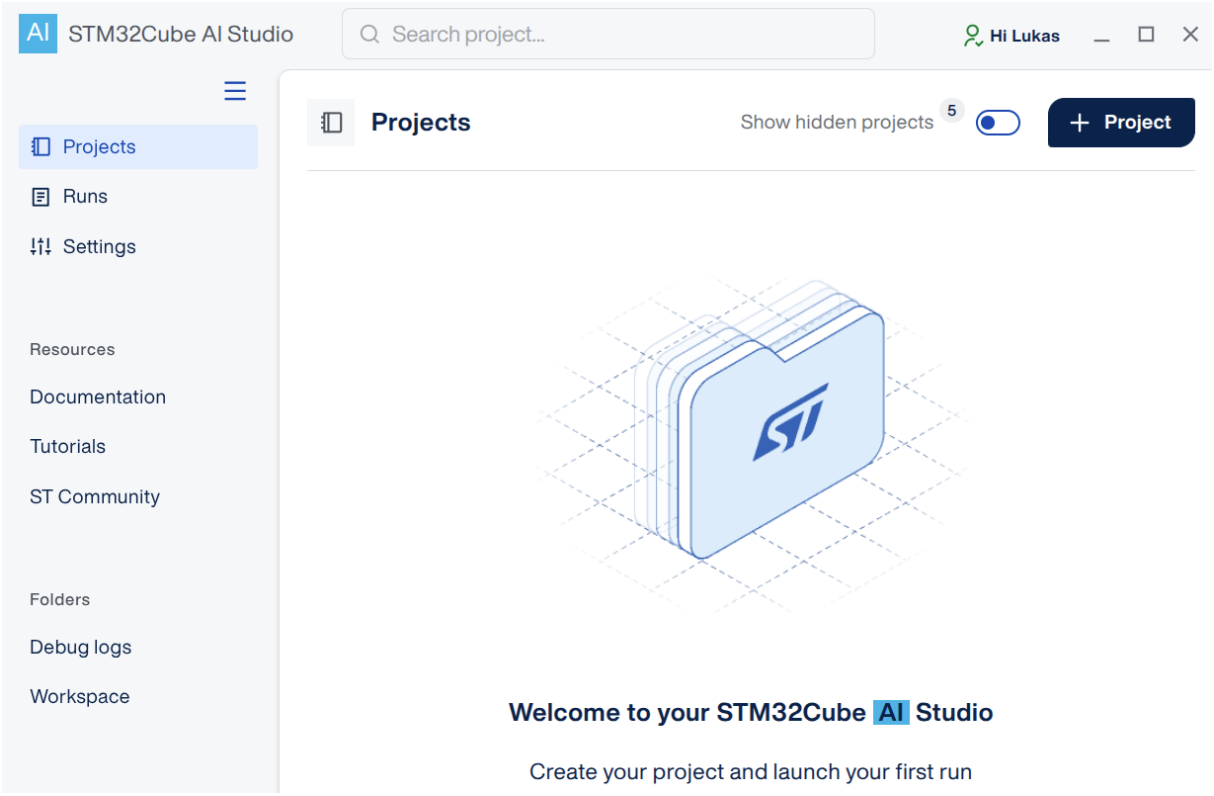
   ```
   deactivate
   ```


© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

5 Creating a Project for STM32CubeIDE

The project for **STM32CubeIDE** is created using the **STM32Cube AI Studio** tool. To prepare the project follow steps bellow.

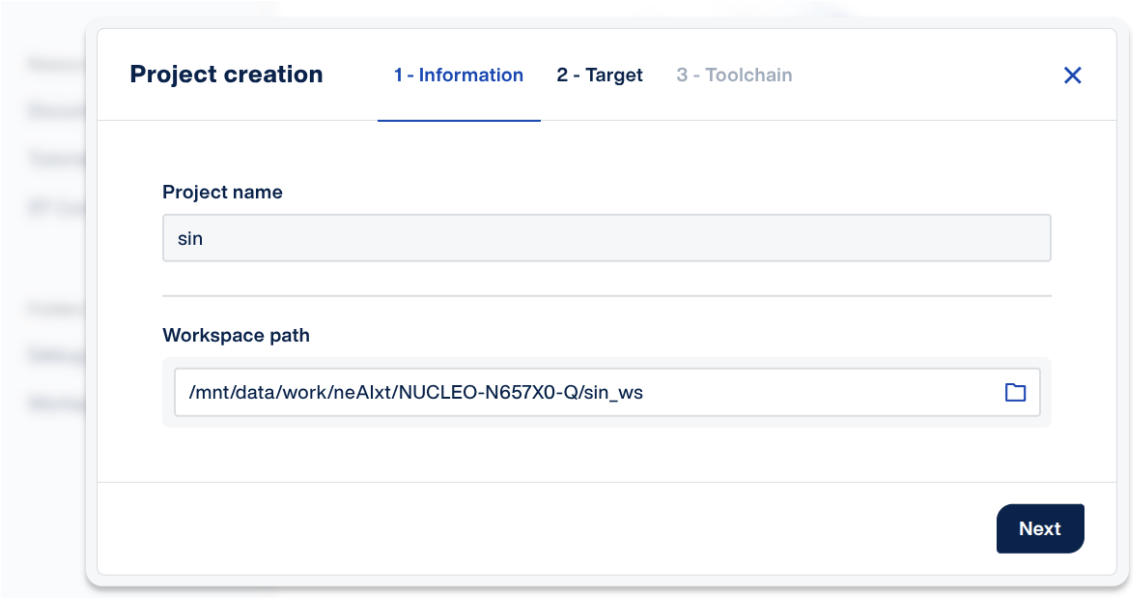
1. Start the **STM32Cube AI Studio** tool and create the project by clicking *+Project* button.



2. Set name and location of the project, in the *1-Information* dialog box set

   Project name → **sin**

   Workspace path → **$your_work/sin_ws**

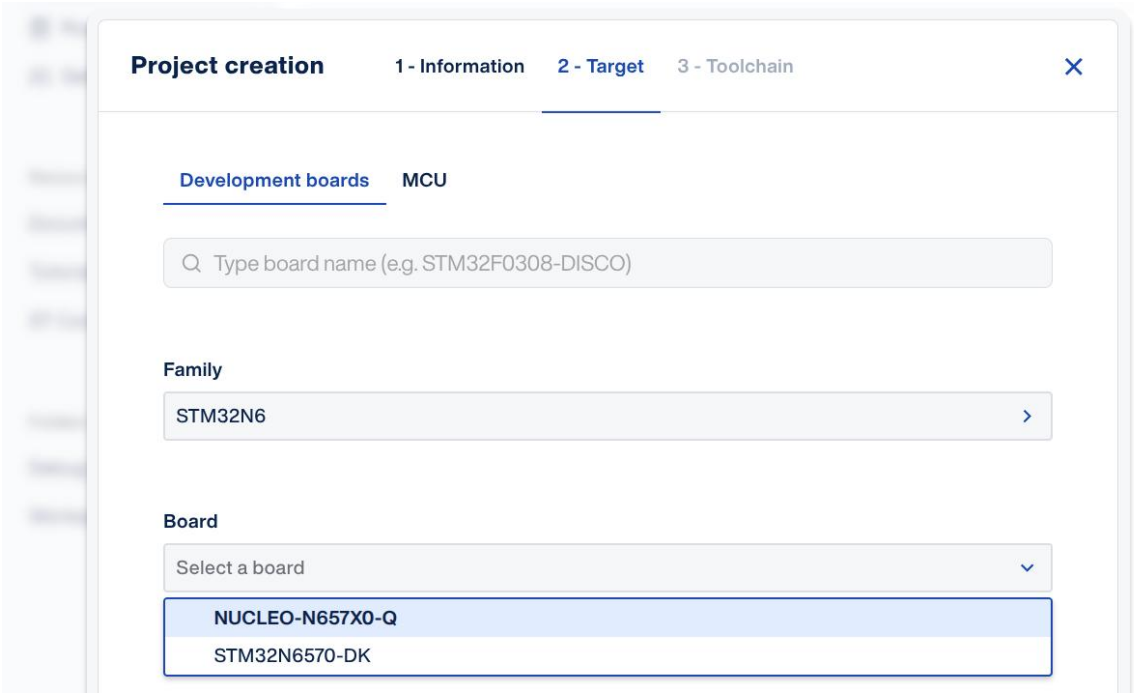


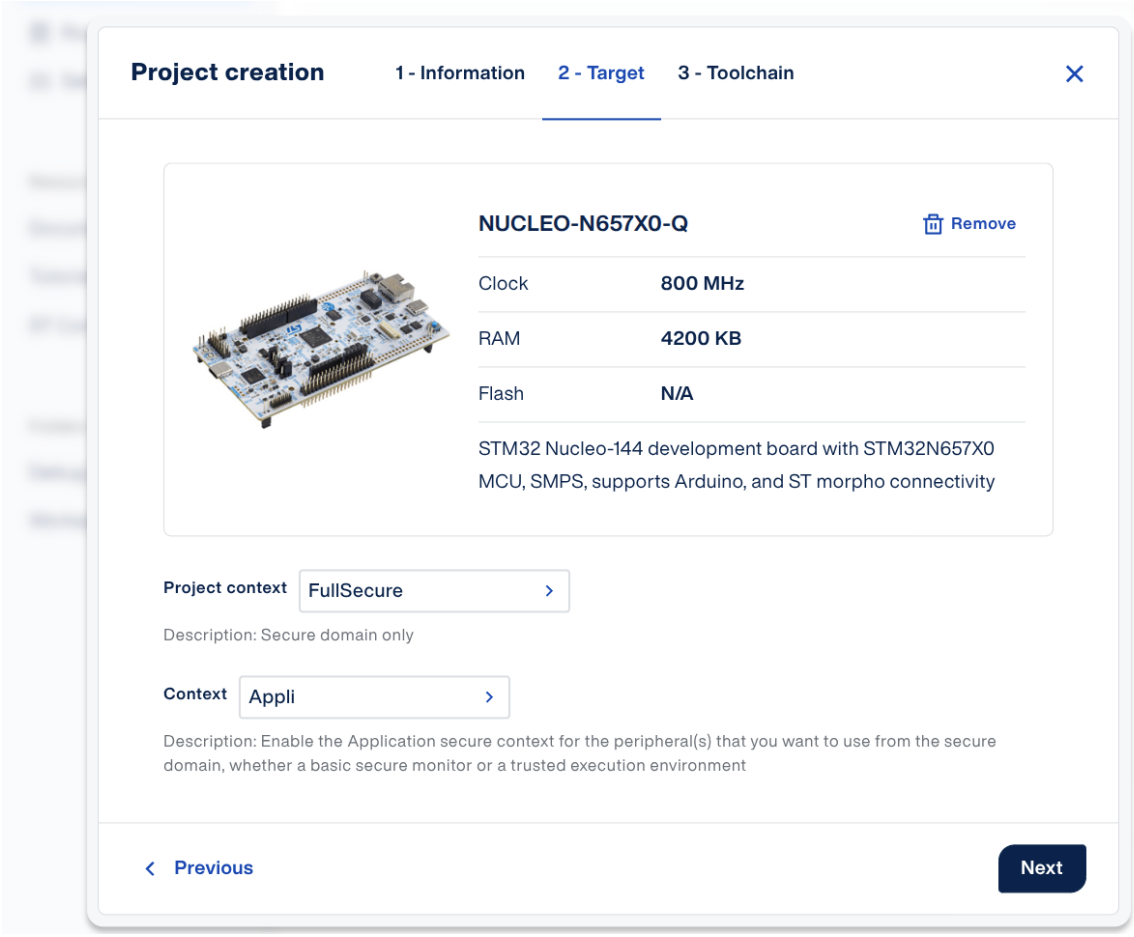
© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

3. Select NUCLEO-N657X0-Q board as a target, in the 2-*Target* dialog box select

   Family → **STM32N6**

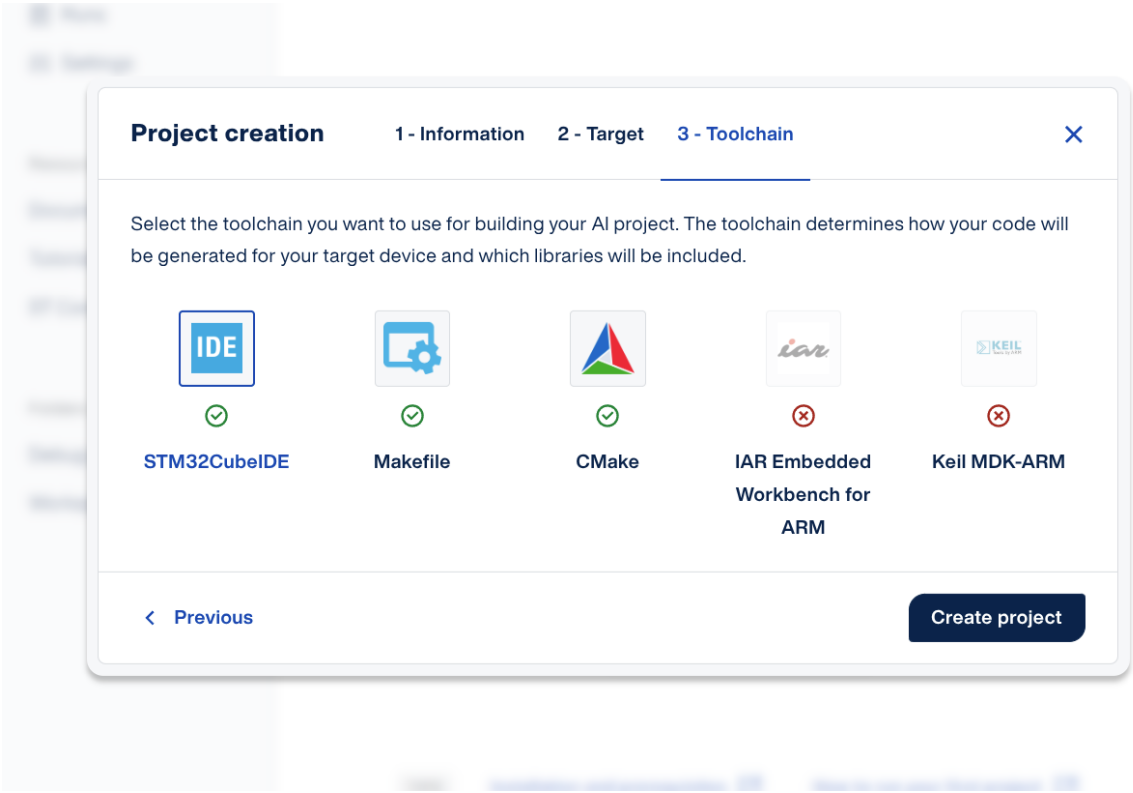
   Board → **NUCLEO-N657X0-Q**





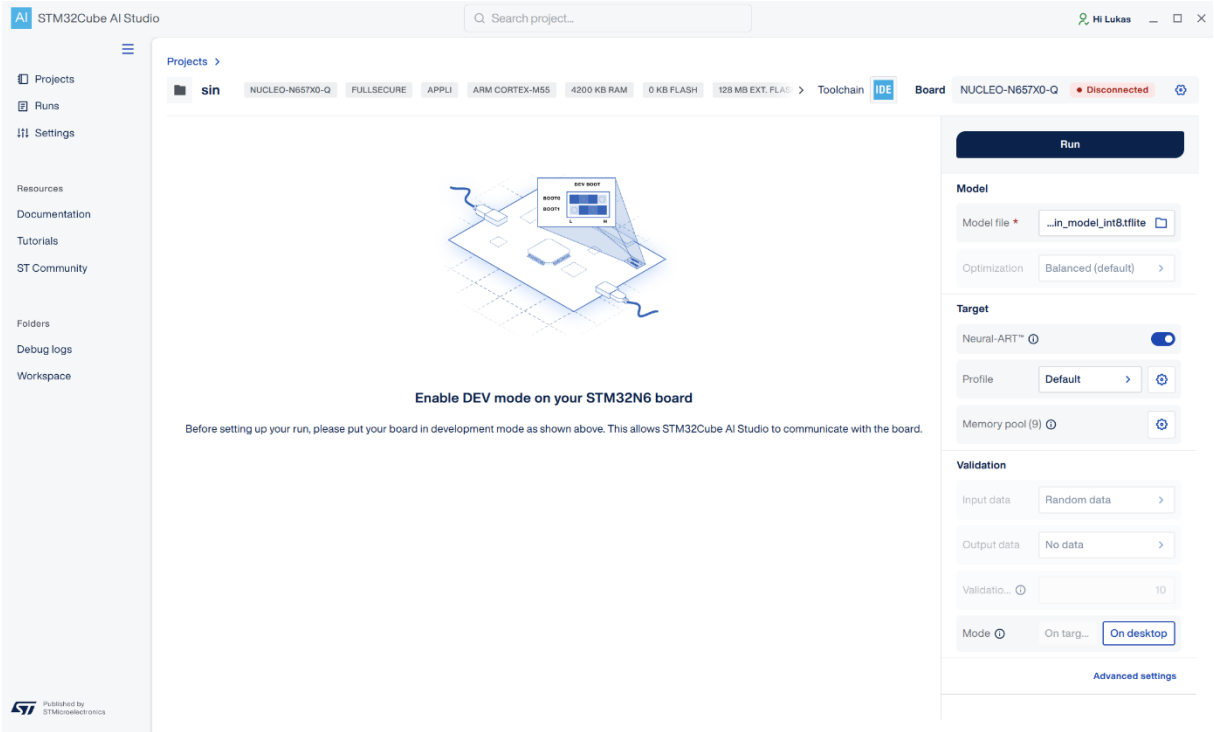
© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

4. Select the toolchain for which the project will be generated, in the 3-*Toolchain* dialog box select **STM32CubeIDE** and click the *Create project* button.

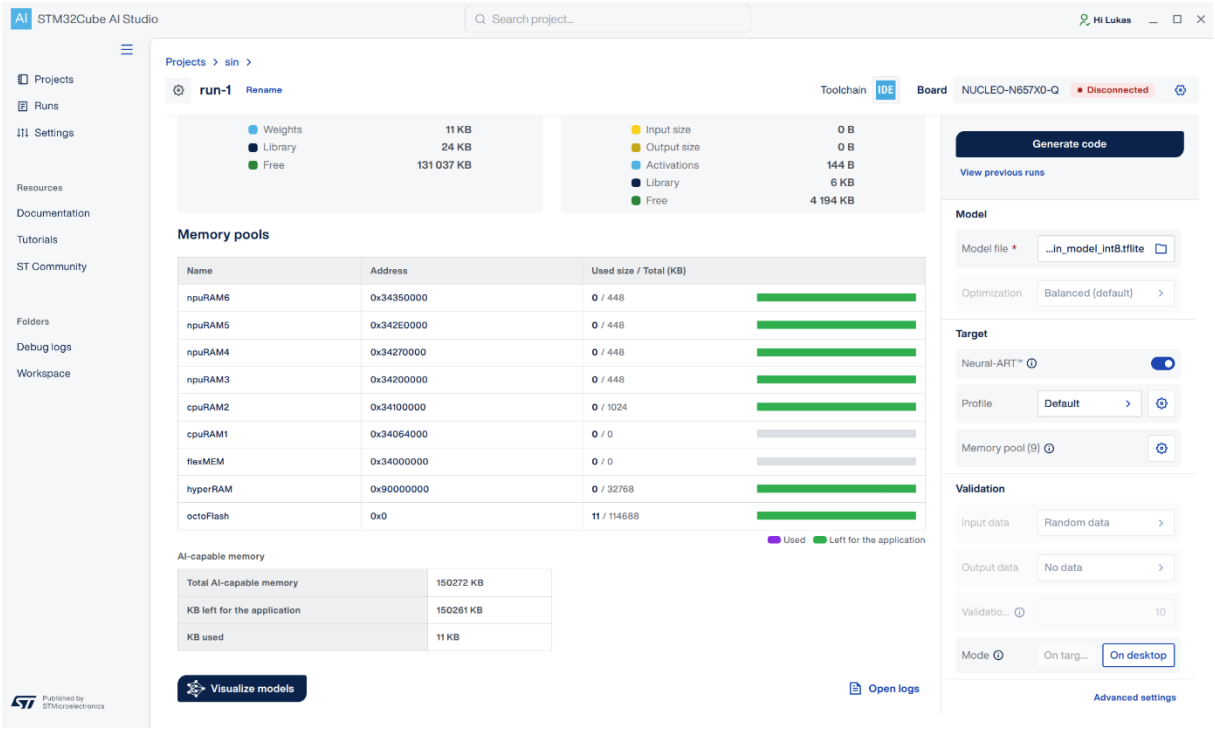


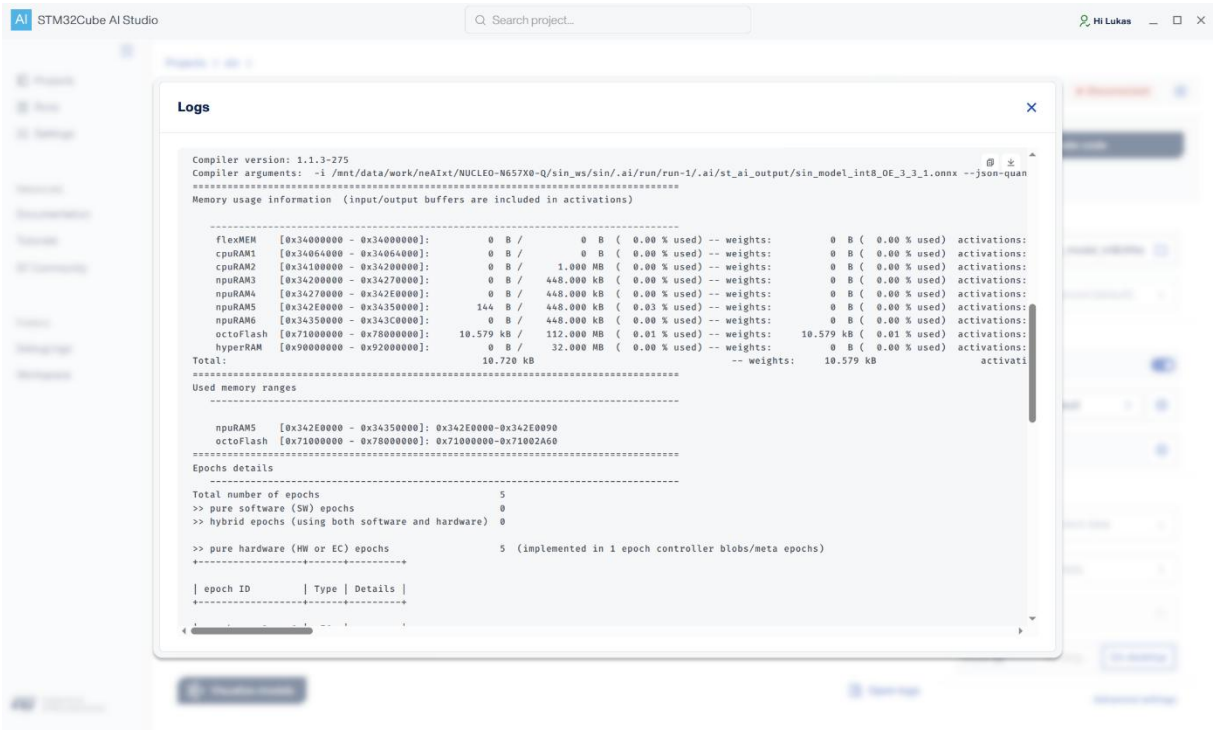
5. Add AI model to project. Select the model **$your_work/sin_model/sin_model_int8.tflite** and click *Run* button to analyze and optimize the model for the selected ST component.

   The tool creates the necessary configuration. When we roll the window all the way down, we can view the log (Open logs) or visualize the model.



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



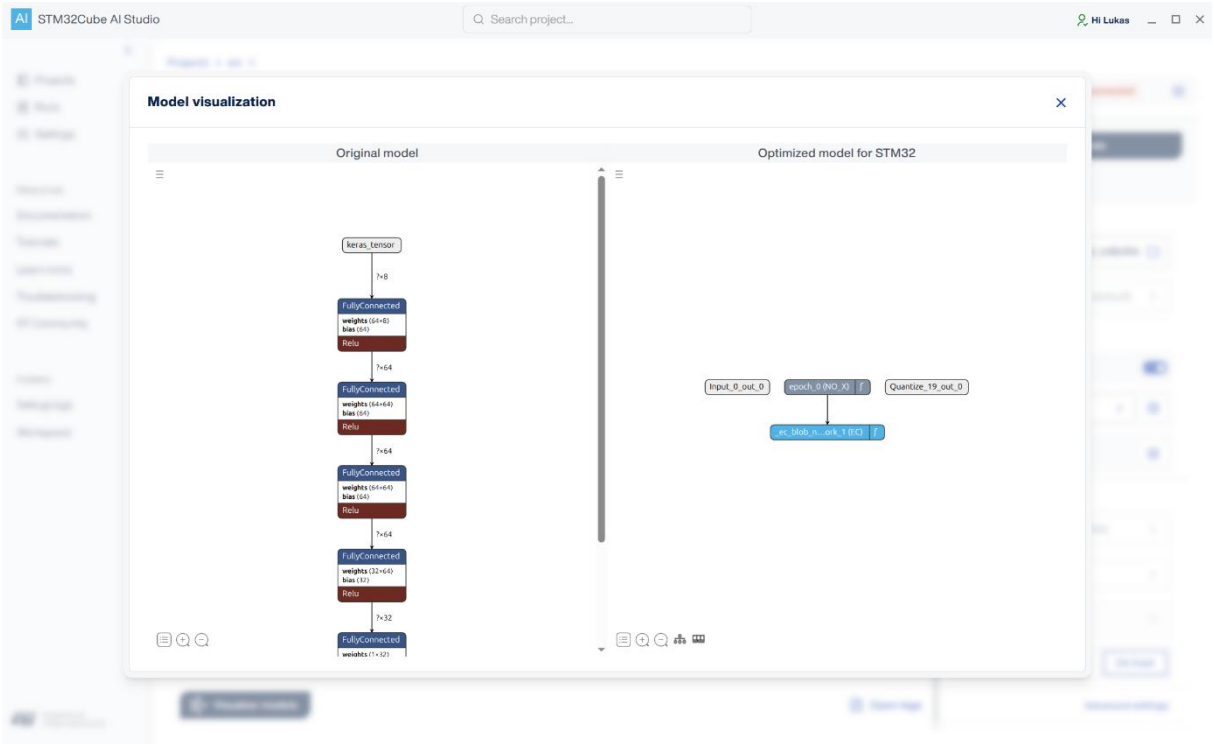


There are key lines in the log file:

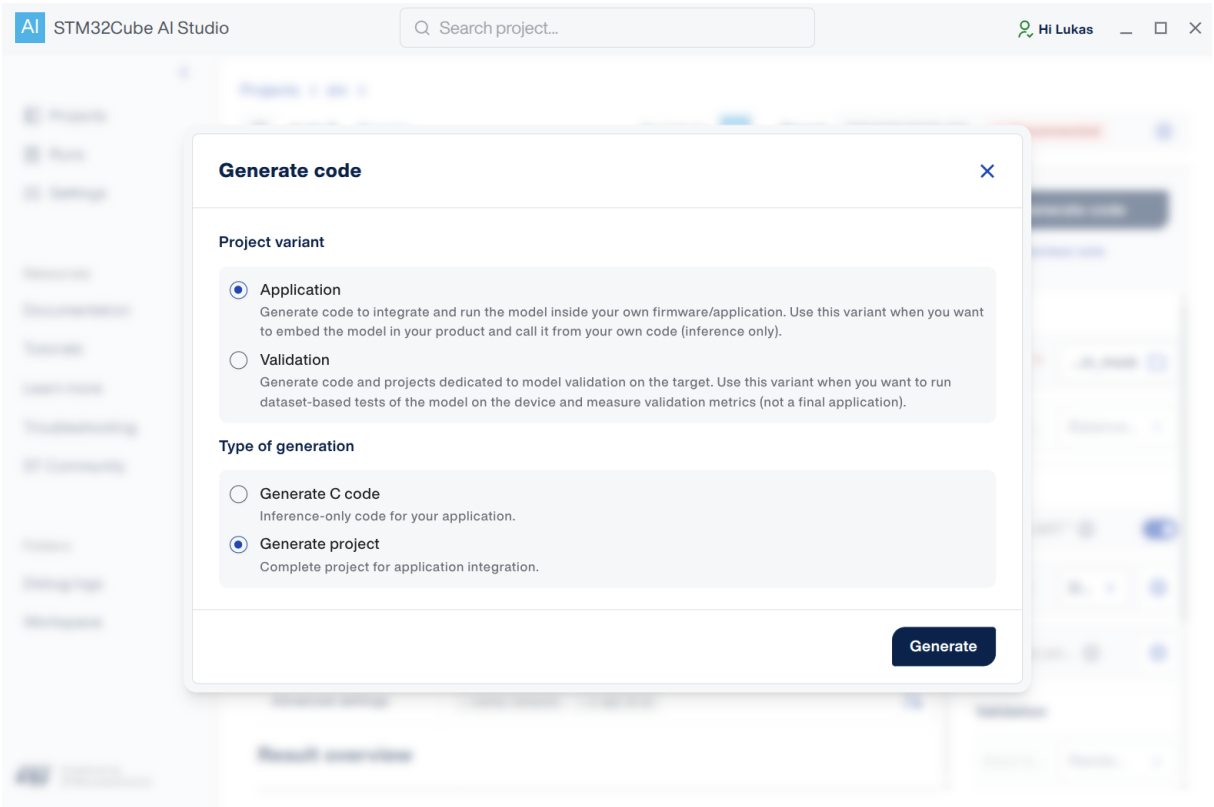
```
Total number of epochs                                5
>> pure software (SW) epochs                          0
>> hybrid epochs (using both software and hardware)   0
>> pure hardware (HW or EC) epochs                    5  (implemented in 1 epoch…
```

which say that 5 layers of the model will be counted as one object in the NPU.

ÚTIA Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.
© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



6. Generate project code, click *Generate code* button and select
   Project variant → **Application**
   Type of generation → **Generate project**



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

This will produce the following structure in the folder *$your_work/sin_ws/sin*

```
.ai
Appli                                   main application
Drivers
FSBL                                    FSBL
Middlewares
Secure_nsclib
STM32CubeIDE                            project for STM32CubeIDE
.mxproject
MX25UM51245G_STM32N6570-NUCLEO.stldr
NUCLEO-N657X0-Q.ioc
project.stai
```

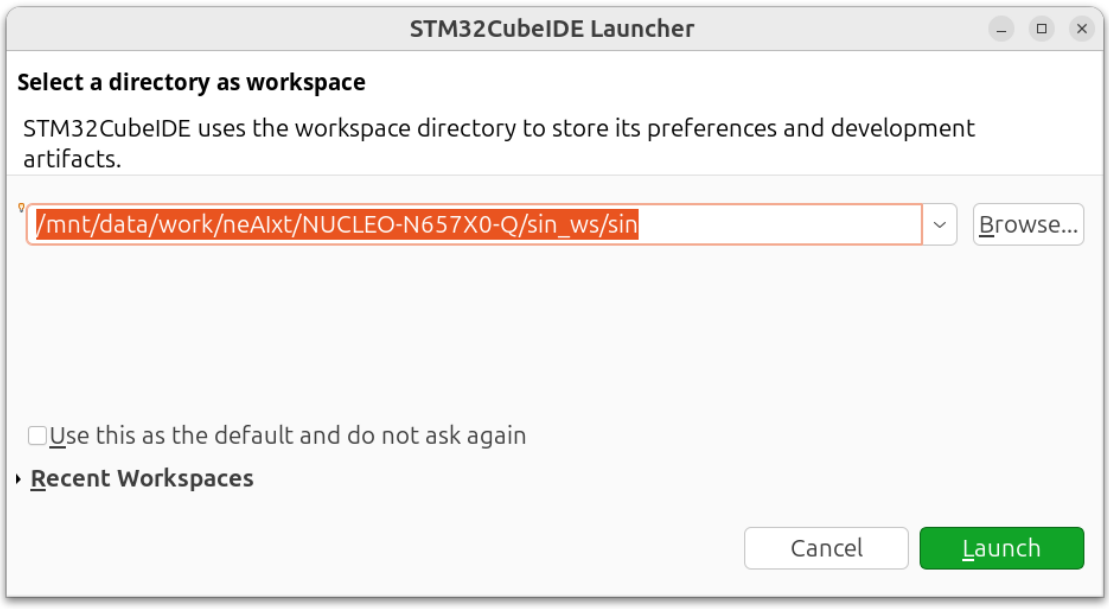
6 Modifying the source code of the STM32CubeIDE project

The project generated for STM32CubeIDE contains all essential components. However, the provided source code does not include input data generation, input quantization, or post-processing of the neural network outputs. To add this functionality, follow the procedure below.

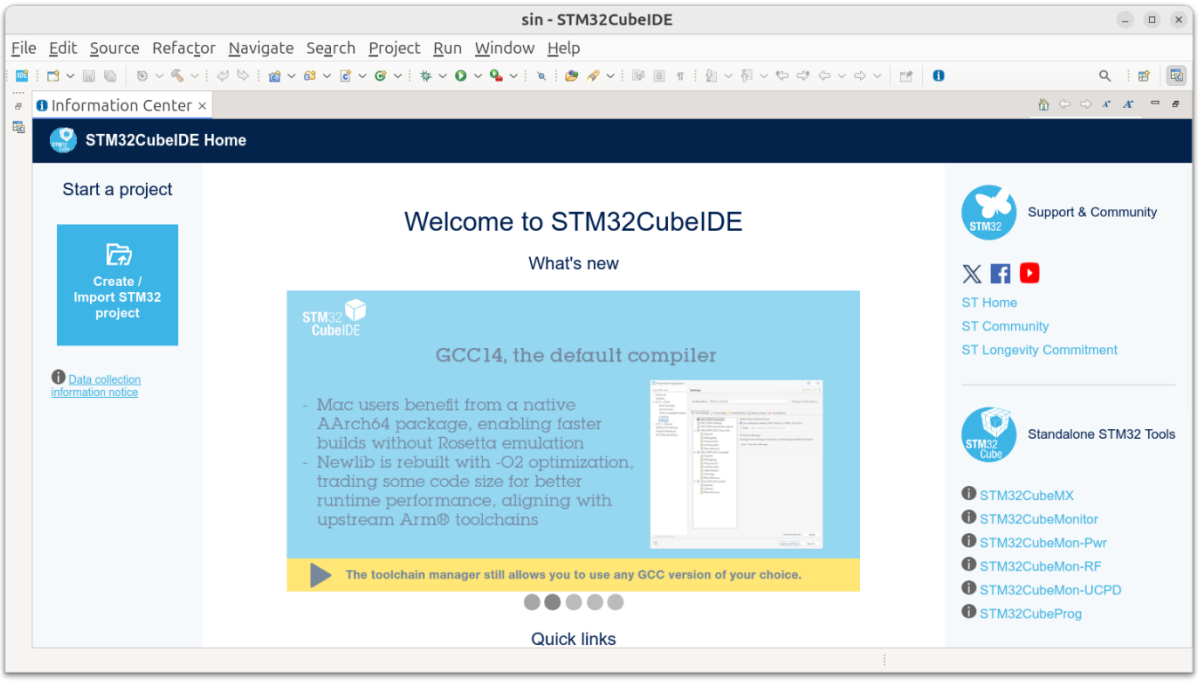
1. First, analyze the file *$your_work/sin_ws/sin/Appli/AI/App/app_x-cube-ai.c*. Here you will find two predefined, but empty, functions:
    - **acquire_and_process_data()** and
    - **post_process()**
2. The attached file *app_x-cube-ai.c* contains all necessary modifications, i.e. preprocessing and postprocessing, quantization and dequantization, and the way of displaying the result.
    - Compare the original *app_x-cube-ai.c* file with the modified one.
    - Overwrite the original *app_x-cube-ai.c* file with the modified one
3. The *app_x-cube-ai.c* file also carries the corresponding header *app_x-cube-ai.h*, which is also modified accordingly. Overwrite the original *app_x-cube-ai.h* file with the modified one.

7 Project Compilation in STM32CubeIDE

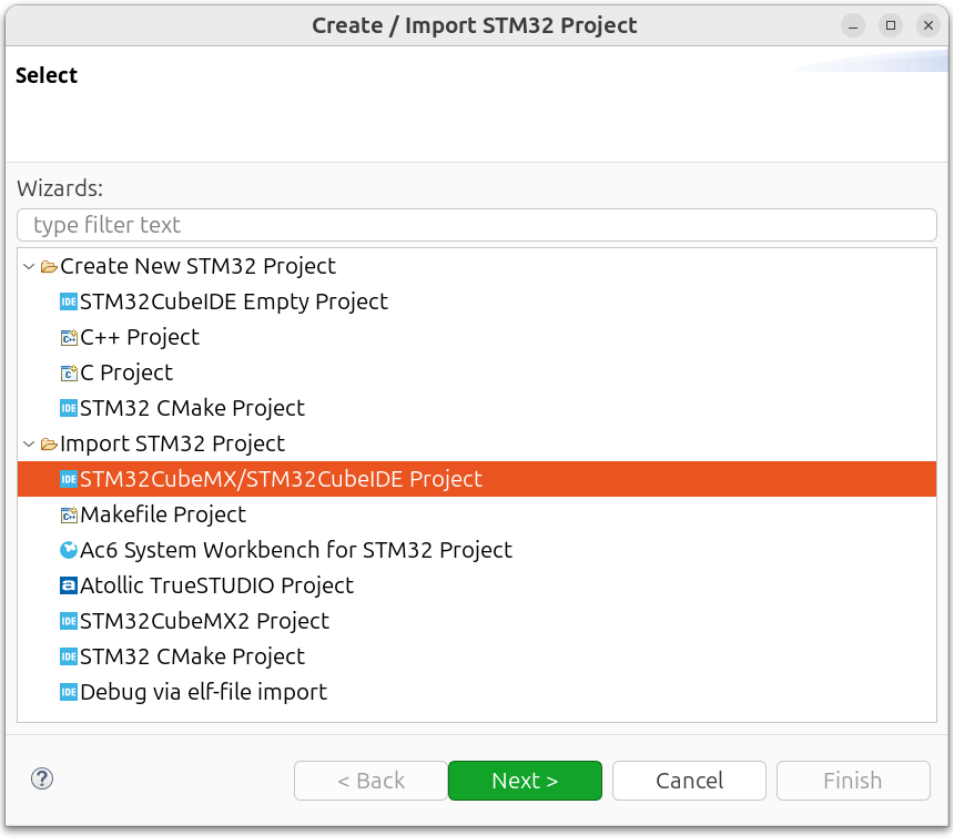
1. Start **STM32CubeIDE**, set the workspace to **$your_work/sin_ws/sin.**



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

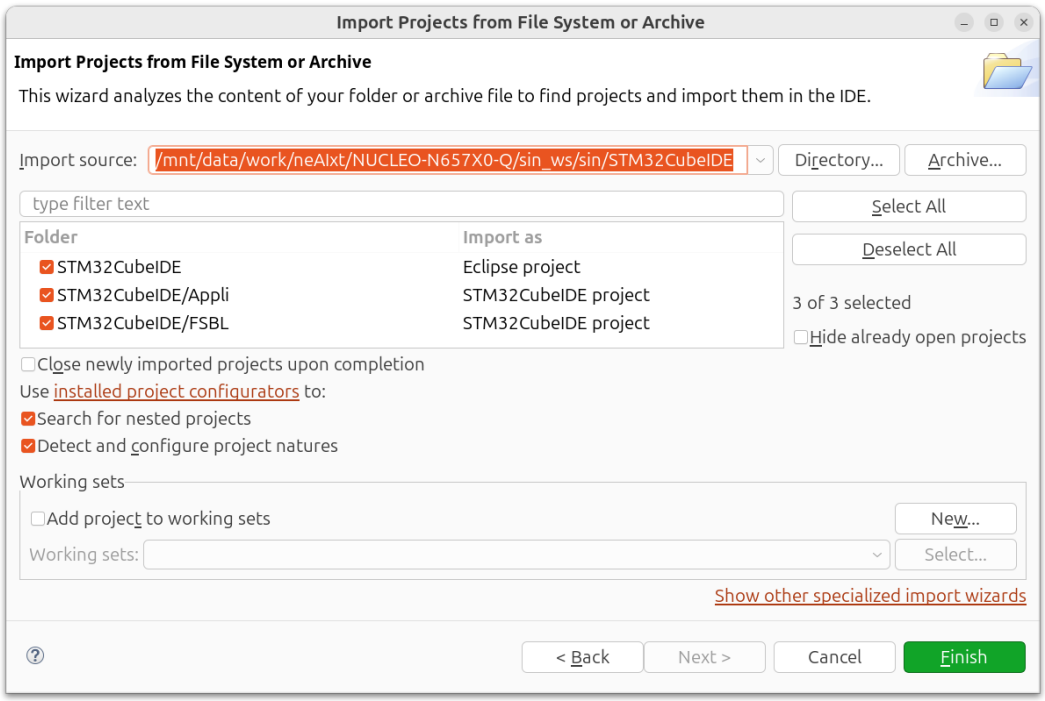


2. Import the prepared project by clicking on *Create / Import STM32 project.*
   Select *Import STM32 Project → STM32CubeMX/STM32CubeIDE Project*



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

Click on the *Directory* button and set the path to **$your_work/sin_ws/sin/STM32CubeIDE**



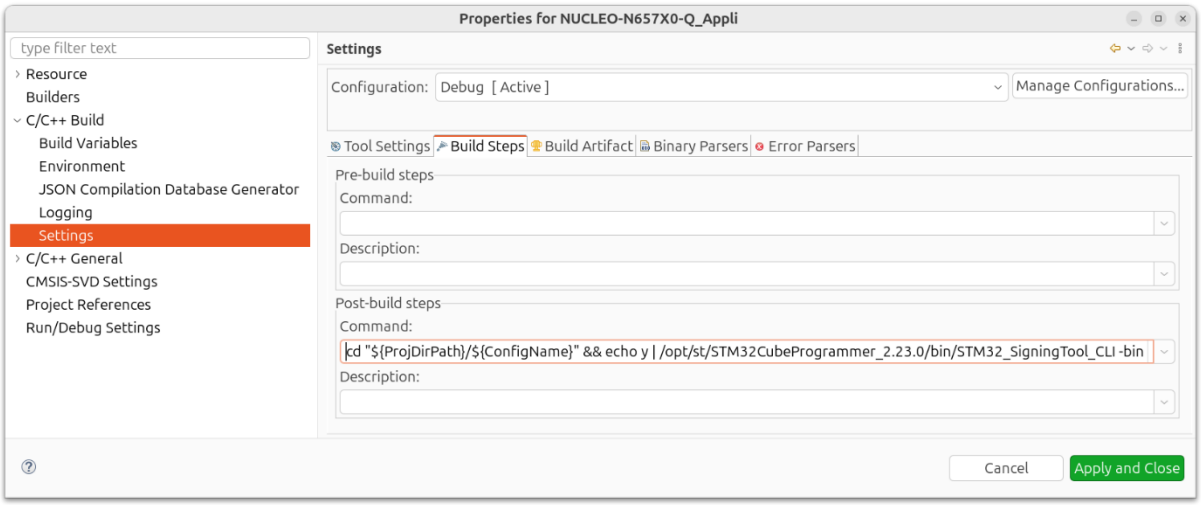
This step imports 2 application projects into the STM32CubeIDE tool:

- NUCLEO-N657X0-Q_Appli
- NUCLEO-N657X0-Q_FSBL

3. For both projects, set up a post build step that will sign the compiled ELFs. The step will be described for *sin_Appli* compiled in debug mode, for the release combination, but also for the *sin_FSBL* project (debug/release), the step is analogous.

   a. For *NUCLEO-N657X0-Q_Appli*, right-click on *NUCLEO-N657X0-Q_Appli* → *Properties* → *C/C++ Build* → *Settings,* tab *Build Steps*. Add the following line to the *Post-build steps* line

```
cd "${ProjDirPath}/${ConfigName}" && echo y |
/opt/st/STM32CubeProgrammer_2.23.0/bin/STM32_SigningTool_CLI -bin
"${ProjName}.bin" -nk -of 0x80000000 -t ssbl -o "${ProjName}-Signed.bin" -hv
2.3 -dump "${ProjName}-Signed.bin" -align
```



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

b. Similarly for *NUCLEO-N657X0-Q_FSBL*, right-click on *NUCLEO-N657X0-Q_FSBL* → *Properties* → *C/C++ Build* → *Settings,* tab *Build Steps*. Add the following line to the *Post-build steps* line

```
cd "${ProjDirPath}/${ConfigName}" && echo y |
/opt/st/STM32CubeProgrammer_2.23.0/bin/STM32_SigningTool_CLI -bin
"${ProjName}.bin" -nk -of 0x80000000 -t fsbl -o "${ProjName}-Signed.bin" -hv
2.3 -dump "${ProjName}-Signed.bin" -align
```

IMORTANT: For Appli and FSBL the type parameter is different, -t ssbl vs. -t fsbl.

Adjust the path to the tool **STM32_SigningTool_CLI** according to your installed tools.

4. Build both projects. We will get binaries:

```
$your_work/sin_ws/sin/STM32CubeIDE/Appli/Release/NUCLEO-N657X0-Q_Appli.elf
$your_work/sin_ws/sin/STM32CubeIDE/Appli/Release/NUCLEO-N657X0-Q_Appli-Signed.bin
$your_work/sin_ws/sin/STM32CubeIDE/FSBL/Release/NUCLEO-N657X0-Q_FSBL.elf
$your_work/sin_ws/sin/STM32CubeIDE/FSBL/Release/NUCLEO-N657X0-Q_FSBL-Signed.bin
```

8 Project Debugging in STM32CubeIDE

The *NUCLEO-N657X0-Q_Appli* AI application cannot be debugged directly. Start the *NUCLEO-N657X0-Q_FSBL* first-stage bootloader under the debugger and let it transfer control to the application. The procedure below configures the *NUCLEO-N657X0-Q*_FSBL debug session to load the application image and symbols.

Before you begin check the following steps:

- Build both projects so that the required ELF and signed binary files are available.
- Keep the development board powered off while changing the BOOT0 and BOOT1 settings.
- Confirm that the external-loader file exists at the path used in your environment.

Important: Before starting the debug session, remove every breakpoint from the NUCLEO-N657X0-Q_Appli project. Existing application breakpoints can cause GDB initialization to fail.

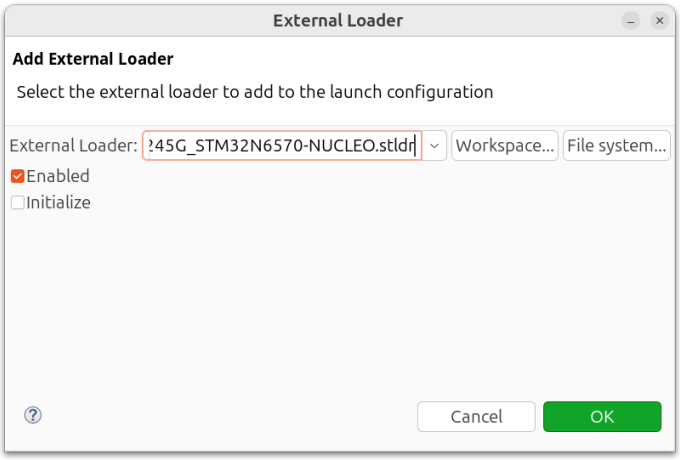
8.1 Create the *NUCLEO-N657X0-Q*_FSBL Debug Configuration

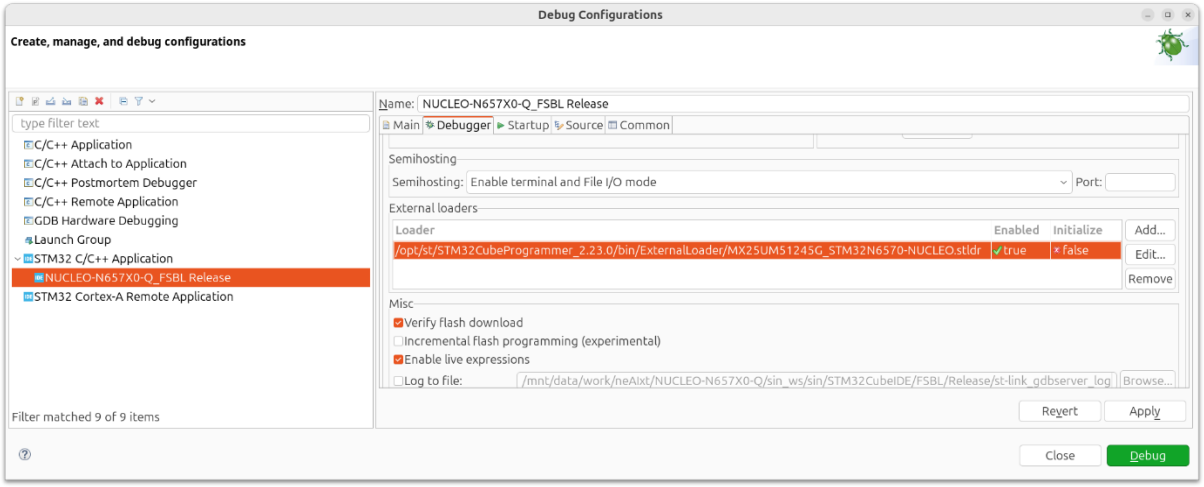
1. In *Project Explorer*, right click on *NUCLEO-N657X0-Q_FSBL* → *Debug As* → *Debug Configurations*.
2. Under *STM32 C/C++ Application*, create a new configuration.
3. In the *Main* tab click *Search Project* button and select **Debug/sin_FSBL.elf**
4. Switch to the *Debugger* tab.
5. Scroll to *External loaders* and click *Add* button.
6. Choose the *external-loader file*, check Enabled, and leave Initialize unchecked.

```
/opt/st/STM32CubeProgrammer_2.23.0/bin/ExternalLoader/MX25UM51245G_STM32N6570-
NUCLEO.stldr
```

Modify the path to the external loader according to your environment. Leave the window open.

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved





8.2 Configure Application Loading and Symbols

1. If you already have the *Debug Configurations* window open from the previous step, switch to the *Startup* tab. Otherwise, open it again and then do the switch.
2. In *Load Image and Symbols*, keep the existing *NUCLEO-N657X0-Q_FSBL* entry and add two more entries.
   a. Add unsigned binary *NUCLEO-N657X0-Q_Appli.elf* as the application symbol.

      Project → **NUCLEO-N657X0-Q_Appli.elf**

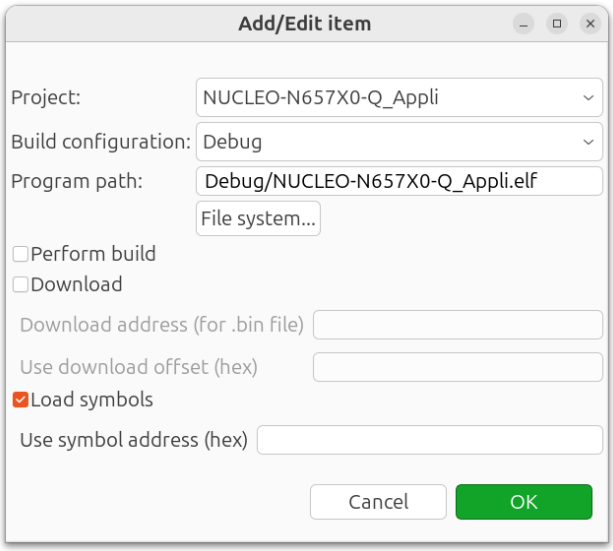
      Build configuration → **Debug**

      Perform build → **unchecked**

      Download → **unchecked**

      Load symbol → **checked**

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

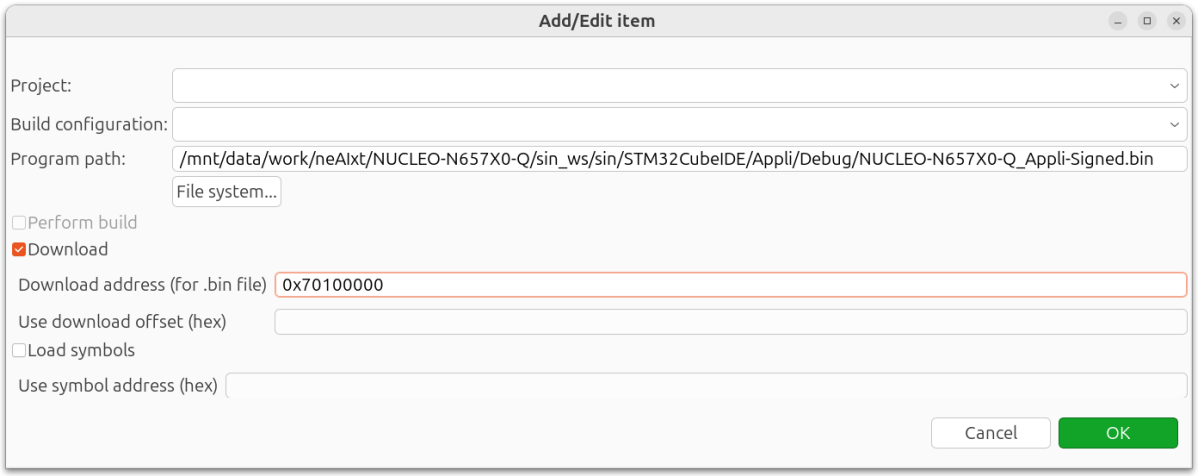


b. Add signed *NUCLEO-N657X0-Q_Appli-Signed.bin* as the application symbol

Program path → **$your_work/neAIxt/sin_ws/sin/STM32CubeIDE/Appli/Debug/NUCLEO-N657X0-Q_Appli-Signed.bin**

Download → **checked**

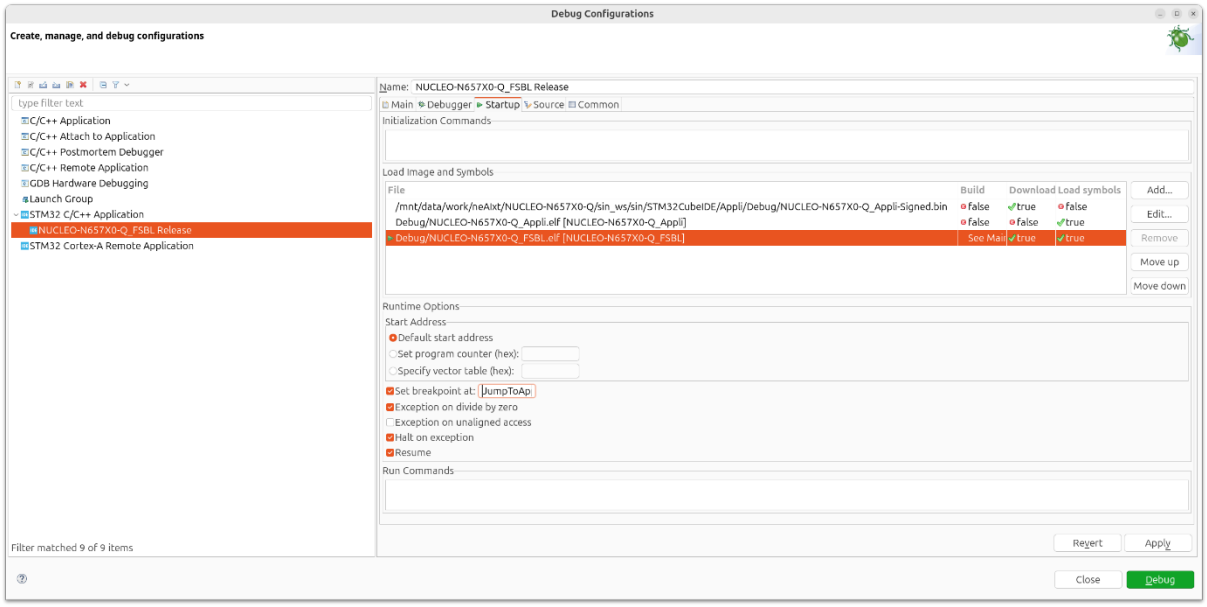
Download address (for .bin file) → **0x70100000**



3. Verify the three Load Image and Symbols entries. The Build, Download, and Load symbols indicators must match the configuration shown below.

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

4. Modify *Set breakpoint*, enter **JumpToApplication**.



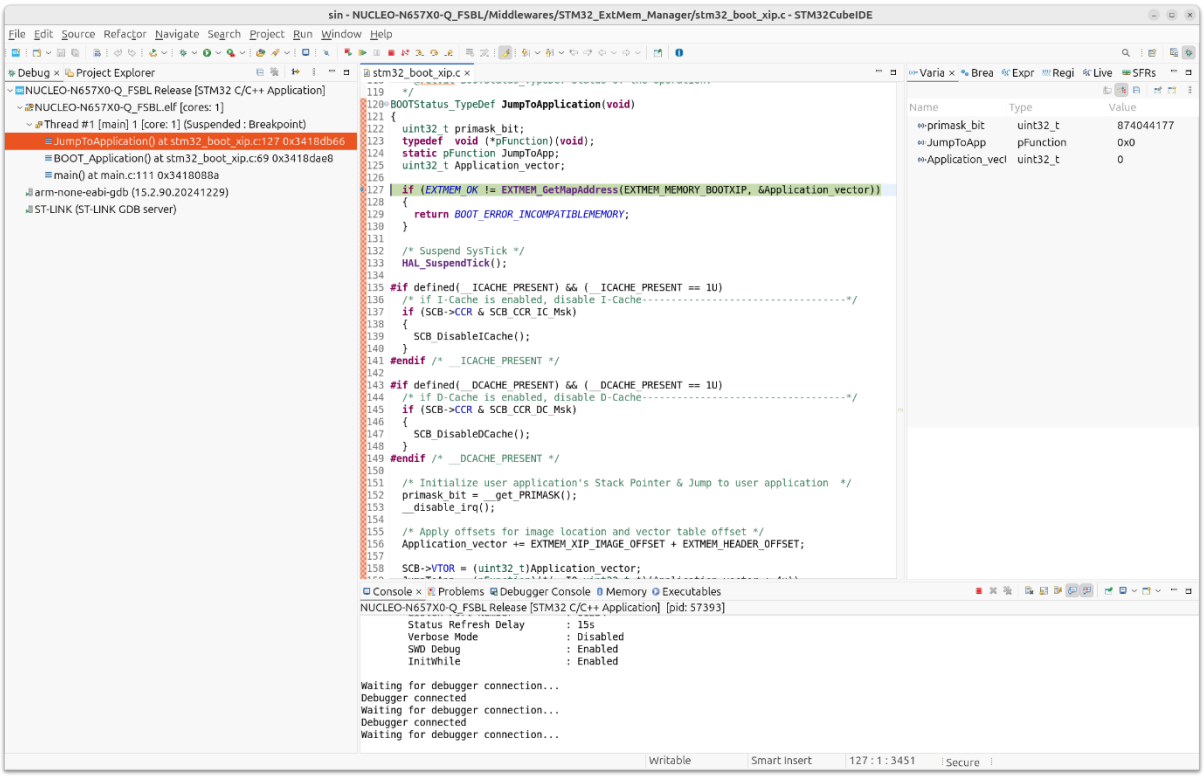
8.3 Start the Debugging Session

1. Set board power supply. NUCLEO-N657X0-Q allows user to choose one of three power supply options, USB-C STLINK, USB-C SNK and external 5V. By default, power supply via USB-C STLINK from PC is selected (USB connector CN10). It is recommended to switch the board to power supply via USB-C SNK labeled as USB (connector CN8) and use USB-C STLINK only for debug/programming/UART. We configure the power supply using the CN9 switch on the board, connecting pins 3-4 5V_USB_SNK with a jumper. It is also recommended to use a separate power supply adapter for power, not a PC.
2. In the *NUCLEO-N657X0-Q_Appli* project, remove all existing breakpoints.

   USEFUL HINT: To view and remove breakpoints in STM32CubeIDE, select menu Window → Show View → Other …, then choose Breakpoints.
3. Set the board boot mode to *development boot* by setting BOOT1 to H, switch JP2 from 1-2 position to 2-3.
4. Power the board on via USB and connect USB-C STLINK with PC.
5. Start the *NUCLEO-N657X0-Q_FSBL* debug session. Execution should stop at the *JumpToApplication()* function.
6. At this point, you can add a breakpoint to the source code of *NUCLEO-N657X0-Q_Appli* and then click the Resume button (F8) to handover from FSBL to the NUCLEO-N657X0-Q_Appli.
7. Debug *NUCLEO-N657X0-Q_Appli*.

   NOTE: To safely exit the debug session, click the Terminate button (Ctrl+F2) and remove all NUCLEO-N657X0-Q_Appli breakpoints.

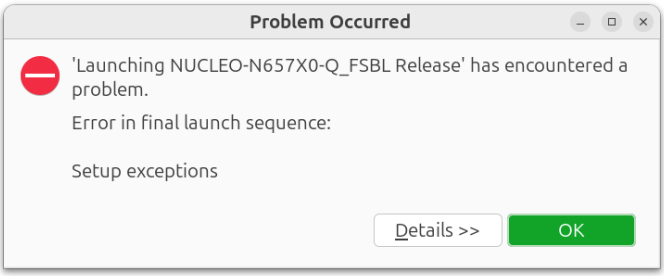
© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



8.4 Troubleshooting

GDB Initialization or Final Launch Sequence Fails

Symptom: STM32CubeIDE reports an error in the final launch sequence, because application breakpoints are still present.



Recovery:

- Remove all breakpoints from *NUCLEO-N657X0-Q_Appli*.
- Power-cycle the board.
- Start the *NUCLEO-N657X0-Q_FSBL* debug configuration again.

STM32CubeProgrammer download fails

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

Symptom: STM32CubeIDE reports that the STM32CubeProgrammer download failed. This commonly indicates that the board was not power-cycled.



Recovery:

- Power-cycle the board, then retry the debug session.

9 Programming NUCLEO-N657X0-Q

To program the board, follow the steps below.

1. Set the board power supply as described in Section 8.3 Step 1.
2. Set the board boot mode to *development boot* (programming mode) by setting BOOT1 to H, switch JP2 from 1-2 position to 2-3.
3. Connect power to USB.
4. Connect the STLINK USB-C programming port to the PC.
5. Configure the serial terminal to 115200, 8 data bits, 1 stop bit, no parity, no control. On Linux, the UART device is visible as */dev/ttyACM0*.
6. In the PC terminal, go to the directory *$your_work/sin_ws/sin/STM32CubeIDE.*
7. Concatenate signed FSBL and applications binaries into a single file, from the PC terminal execute:

   ```
   srec_cat FSBL/Release/ NUCLEO-N657X0-Q_FSBL-Signed.bin -Binary -offset
   0x70000000 Appli/Release/ NUCLEO-N657X0-Q_Appli-Signed.bin -Binary -offset
   0x70100000 -o sin.hex -Intel
   ```

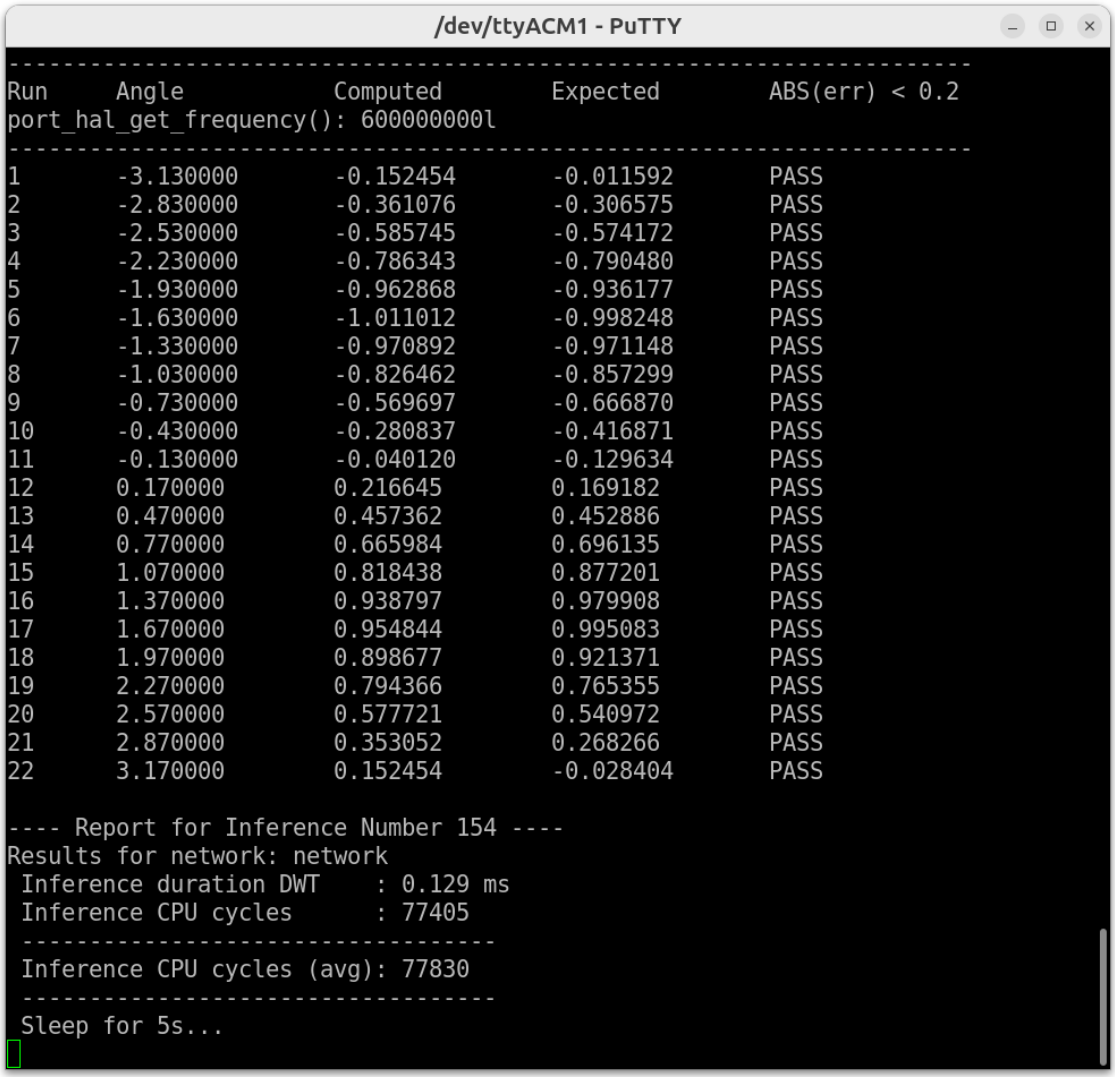
   This will create the file *$your_work/sin_ws/sin/STM32CubeIDE/sin.hex*

   On Windows, the srec_cat tool will probably be in the *c:\Program Files\srecord\bin* folder.
8. Program the board, from the PC terminal execute:

   ```
   /opt/st/STM32CubeProgrammer_2.23.0/bin/STM32_Programmer_CLI -c port=SWD mode=UR
   -el
   /opt/st/STM32CubeProgrammer_2.23.0/bin/ExternalLoader/MX25UM51245G_STM32N6570-
   NUCLEO.stldr -hardRst -w sin.hex
   ```

   Modify path to the programmer *STM32_Programmer_CLI* and external loader *MX25UM51245G_STM32N6570-NUCLEO.stldr* according to your environment.
9. Switch the board back to normal boot mode, move the JP2 marked BOOT1 back to 1-2.
10. Reset the board, press the *RESET* button (black button).
11. Observe the serial terminal.

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



10 XSPI High Speed Mode

The NUCLEO-N657X0-Q board does not support fast clock for the OCTOSPI Flash memory out of the box. It runs at 50 MHz by default, but could run up to 200 MHz. This requires modification of the OTP124 (One-Time Programmable) register which turns on optimizations for low-voltage/high-speed (HSLV) clock. The NUCLEO-N657X0-Q board uses:

- VDDIO3 = 1.8 V → XSPI2 / XSPIM Port 2
- VDDIO2 = 3.3 V → XSPI1 / XSPIM Port 1

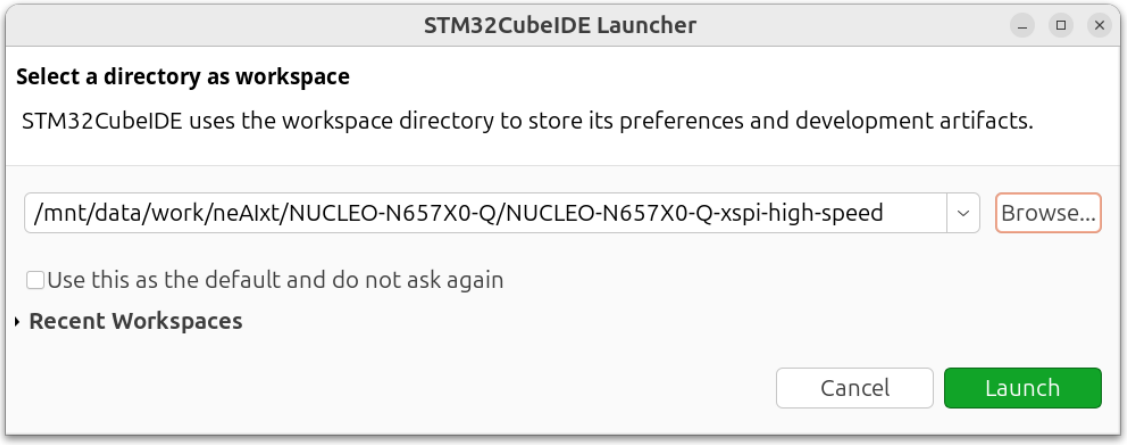
Therefore, only VDDIO3 requires the HSLV configuration and we want to set only bit 15 of the OTP124 fuse.

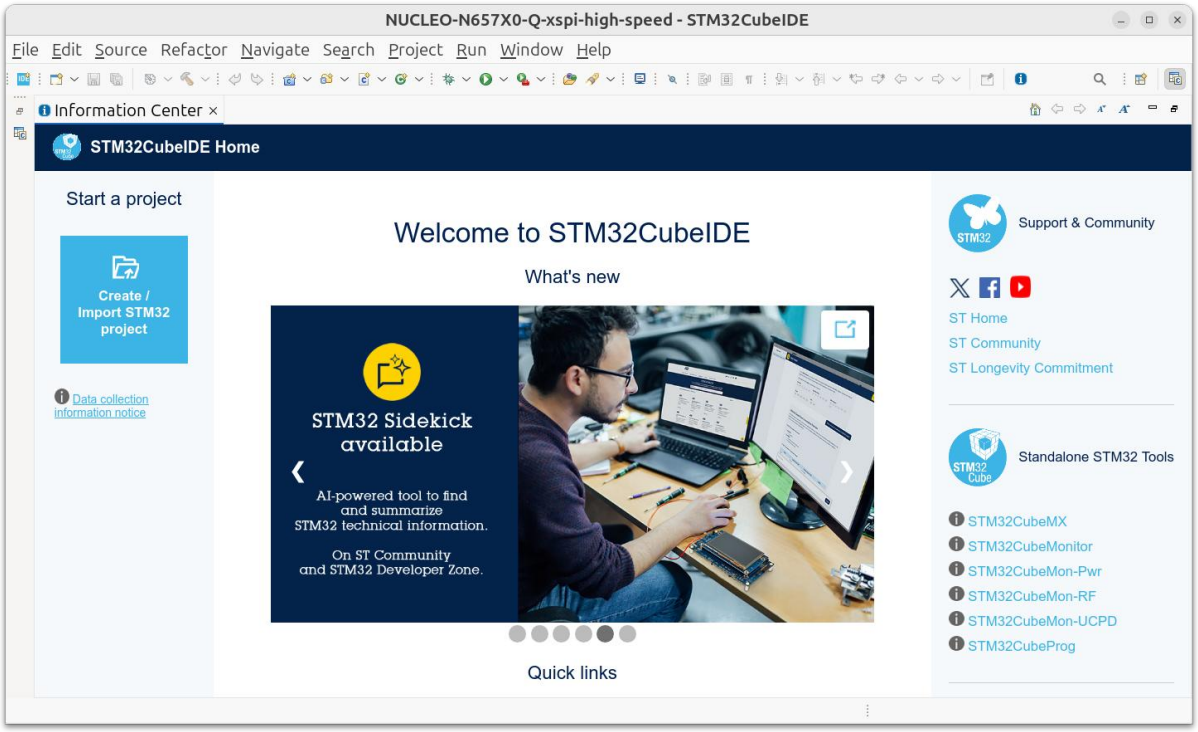
The user does the modification at his own risk, as this operation is irreversible. This application note contains a pre-prepared STM32CubeIDE project that will perform the modification in one run. To speed up XSPI, follow the instructions below. To go through the entire process, continue with section 10.1, if you want to use the precompiled application directly, skip to section 10.2

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

10.1 Setting up HSLV with the Complete Process

1. Start **STM32CubeIDE**, set the workspace to **$your_work/NUCLEO-N657X0-Q-xspi-high-speed.**





2. Import the prepared project by clicking on *Create / Import STM32 project*.

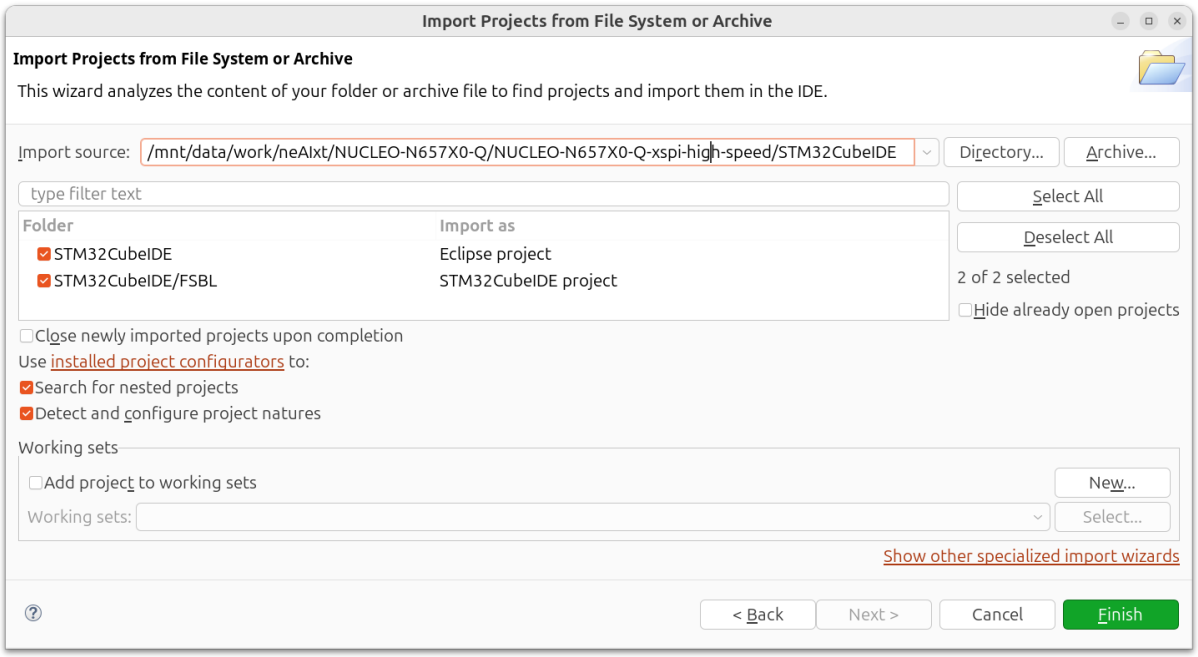
   Select *Import STM32 Project → STM32CubeMX/STM32CubeIDE Project*

   Click on the *Directory* button and set the path to **$your_work//NUCLEO-N657X0-Q-xspi-high-speed/STM32CubeIDE**

   This step imports one application project into the STM32CubeIDE tool:

   - XSPI_high_speed_FSBL

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



3. Build the project (ctrl+b).
4. Set the board power supply as described in Section 8.3 Step 1.
5. Set the board boot mode to *development boot* (programming mode) by setting BOOT1 to H, switch JP2 from 1-2 position to 2-3.
6. Connect power to USB.
7. Connect the STLINK USB-C programming port to the PC.
8. Configure the serial terminal to 115200, 8 data bits, 1 stop bit, no parity, no control. On Linux, the UART device is visible as */dev/ttyACM0*.
9. Start a debug session, right click on *XSPI_high_speed_FSBL* → *Debug As* → *1 STM32 C/C++ Application.*
10. Resume the application, menu Run → *Resume* (F8)
11. Observe the serial terminal. **If you are really sure that you want to make an irreversible write into OTP124, press the 'y' key on the serial terminal**. If the board has not been modified yet, then HSLV will be set.



© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

10.2 Setting up HSLV from the Precompiled Executable Binary

1. Set the board power supply as described in Section 8.3 Step 1.
2. Set the board boot mode to development boot (programming mode) by setting BOOT1 to H, switch JP2 from 1-2 position to 2-3.
3. Connect power to USB.
4. Connect the STLINK USB-C programming port to the PC.
5. Configure the serial terminal to 115200, 8 data bits, 1 stop bit, no parity, no control. On Linux, the UART device is visible as /dev/ttyACM0.
6. Program the board with the precompiled binary file, from the PC terminal go to *$your_work//NUCLEO-N657X0-Q-xspi-high-speed/prebuild* and execute command:

```
/opt/st/STM32CubeProgrammer_2.23.0/bin/STM32_Programmer_CLI -c port=SWD mode=UR
-el
/opt/st/STM32CubeProgrammer_2.23.0/bin/ExternalLoader/MX25UM51245G_STM32N6570-
NUCLEO.stldr -hardRst -w XSPI_high_speed_FSBL-Signed.bin 0x70000000
```

   Modify path to the programmer STM32_Programmer_CLI and external loader MX25UM51245G_STM32N6570-NUCLEO.stldr according to your environment.
7. Switch the board back to normal boot mode, move the JP2 marked BOOT1 back to 1-2.
8. Reset the board, press the *RESET* button (black button).
9. Observe the serial terminal. **If you are really sure that you want to make an irreversible write into OTP124, press the 'y' key on the serial terminal**. If the board has not been modified yet, then HSLV will be set.



10.3 Performance Comparison

Simply turning on HSLV is not enough to speed up the application, in our case sine on the NPU. In the default state, the relevant clock generator is not set up accordingly.

1. Modify the corresponding clock generator configuration, edit file

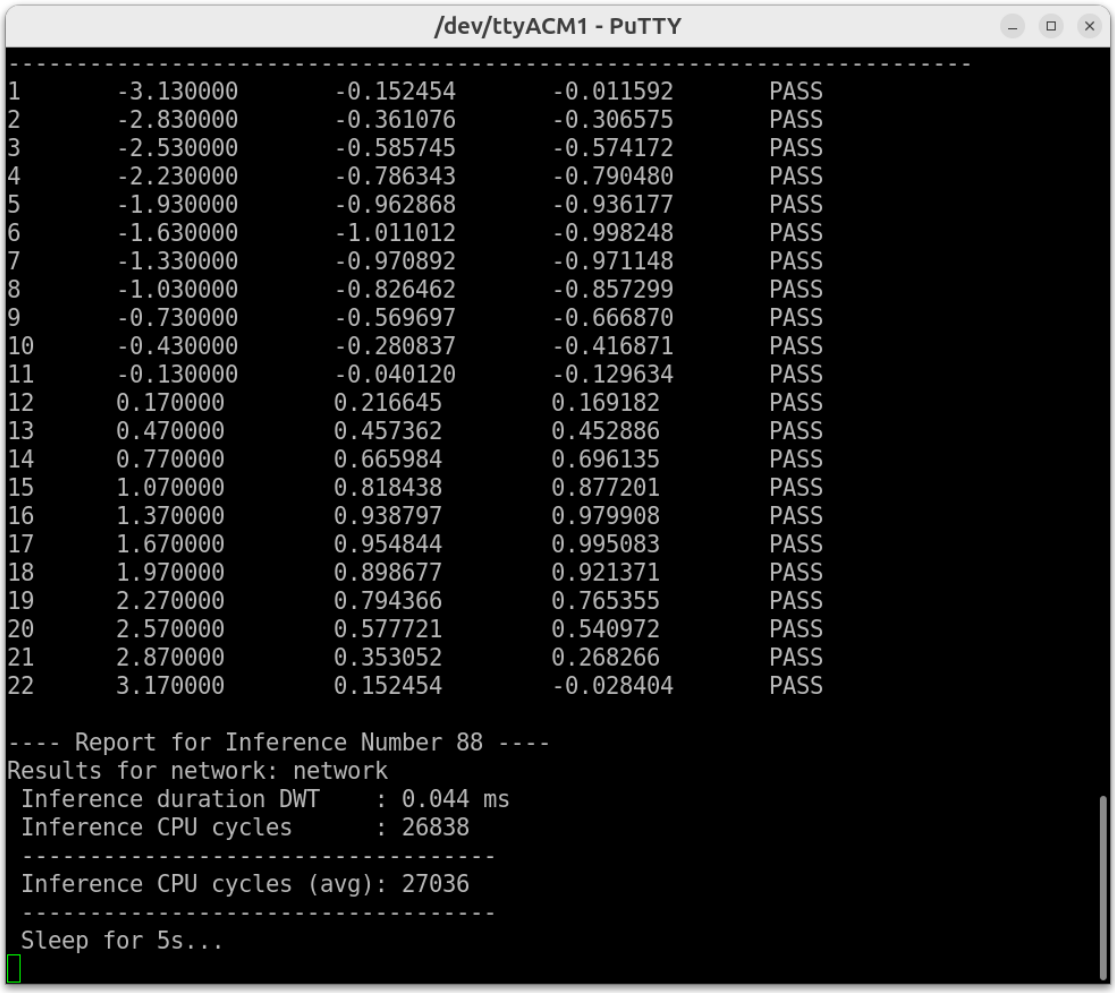
   $your_work/sin_ws/sin/FSBL/Core/Src/stm32n6xx_hal_msp.c

   Line 103 and change clock divider from 48 to 12

```
PeriphClkInitStruct.ICSelection[RCC_IC3].ClockDivider = 12; //48 orig 50MHz
```

2. Re-compile the project and re-program the board.
3. Observe the serial terminal and compare the result with the previous run of the sine application. The difference is in the inference runtime:
   - 129 µs – HSLV disabled, XSPI set to 50 MHz
   - 44 µs – HSLV enabled, XSPI set to 200 MHz

© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



11 Package contents

```
├── doc
│   └── NUCLEO-N657X0-Q-AI-from-the-scratch-v01.pdf
├── NUCLEO-N657X0-Q-xspi-high-speed
├── sin_model
│   ├── sin_model_int8.py
│   └── sin_model_int8.tflite
└── sin_ws
    └── sin
        └── Appli
            └── AI
                └── App
                    ├── app_x-cube-ai.c
                    └── app_x-cube-ai.h
```


© 2026 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved