# AN4657
# Application note

## STM32 in-application programming (IAP) using the USART

## Introduction

An important requirement for most systems based on flash memories is the ability to update firmware when it is installed in the end product. This ability is referred to as in-application programming (IAP).

Most of the new STM32 products are supported by a middleware open source library called Open Bootloader (OpenBL), described in AN3155 "*USART protocol used in the STM32 bootloader", available on www.st.com*.

The purpose of this document is to provide general guidelines for creating an IAP application on STM32 microcontrollers able to run user-specific firmware to perform an IAP of the embedded flash memory, without need for the OpenBL. This feature uses the available communication interfaces supported by the product. The example described in this application note is based on the USART, using the YMODEM protocol.

The X-CUBE-IAP-USART firmware package*, available on www.st.com,* is delivered with this document, and contains the source code of IAP examples for STM32 microcontrollers.

# Contents

# List of tables

# List of figures

# 1 IAP overview

## 1.1 Principle

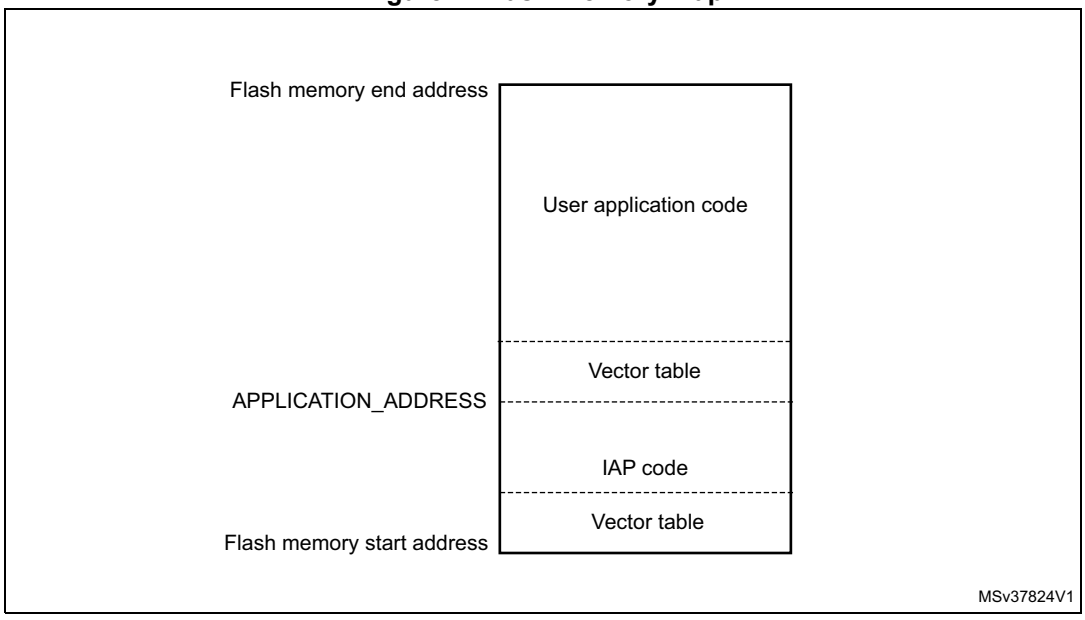
When a reset occurs, the program counter is set to execute the IAP driver. It must be a compact code that checks a specific condition, for example that a combination of keys is pressed. When this condition is met, the IAP driver code either executes a branch that updates the user application, or (usually by default) executes it.

The user application must be separated from the IAP driver. The most practical solution is to place the IAP driver code at the beginning of program memory, and the user code at the beginning of the next free flash memory block, sector, or page. This makes possible to configured independent memory protection on both areas. For the example given in this document the IAP is performed through the USART, rather than a more advanced communication interface, resulting in a minimal memory footprint.

The user application is likely to have independent stack and interrupt vectors (both are recommended but neither is mandatory) as illustrated in *Figure 1*. When the IAP driver directly launches the user application:

1. the IAP driver sets the main stack pointer to the application address
2. the next instruction executes the jump (unconditional branching) to the application
3. the application sets its own interrupt vector table as active.

**Figure 1. Flash memory map**



## 1.2 IAP driver example description

The IAP driver contains the following set of source files:

- *main.c*: where the USART initialization and RCC configuration are set. A main menu is then executed from the *menu.c* file.
- *menu.c*: contains the main menu routine. It gives the options of downloading a new binary file, uploading internal flash memory, executing the binary file already loaded, and managing the write protection of the pages where the user loads the binary file.
- *flash_if.c*: contains write, erase, and configure write protection of the internal flash memory functions.
- *common.c*: contains functions related to read/write from/to USART peripheral
- *ymodem.c*: used to send and receive the data to and from the terminal emulation application using the YMODEM protocol[a]. In the event of a failure when receiving the data, the “Failed to receive the file” error message is displayed. If the data is successfully received, it is programmed into the internal flash memory from the appropriate address. A comparison between internal RAM contents and internal flash memory contents is performed to check the data integrity. If there is any data discrepancy, the “Verification failed” error message is displayed. Other error messages are also displayed when the image file size is greater than the allowed memory space and when the user aborts the task.
- STM32Cube hardware abstraction layer files.

## 1.3 IAP driver flowchart

The user chooses to launch the application or to execute the IAP code for reprogramming purposes by pressing a push-button connected to a pin:

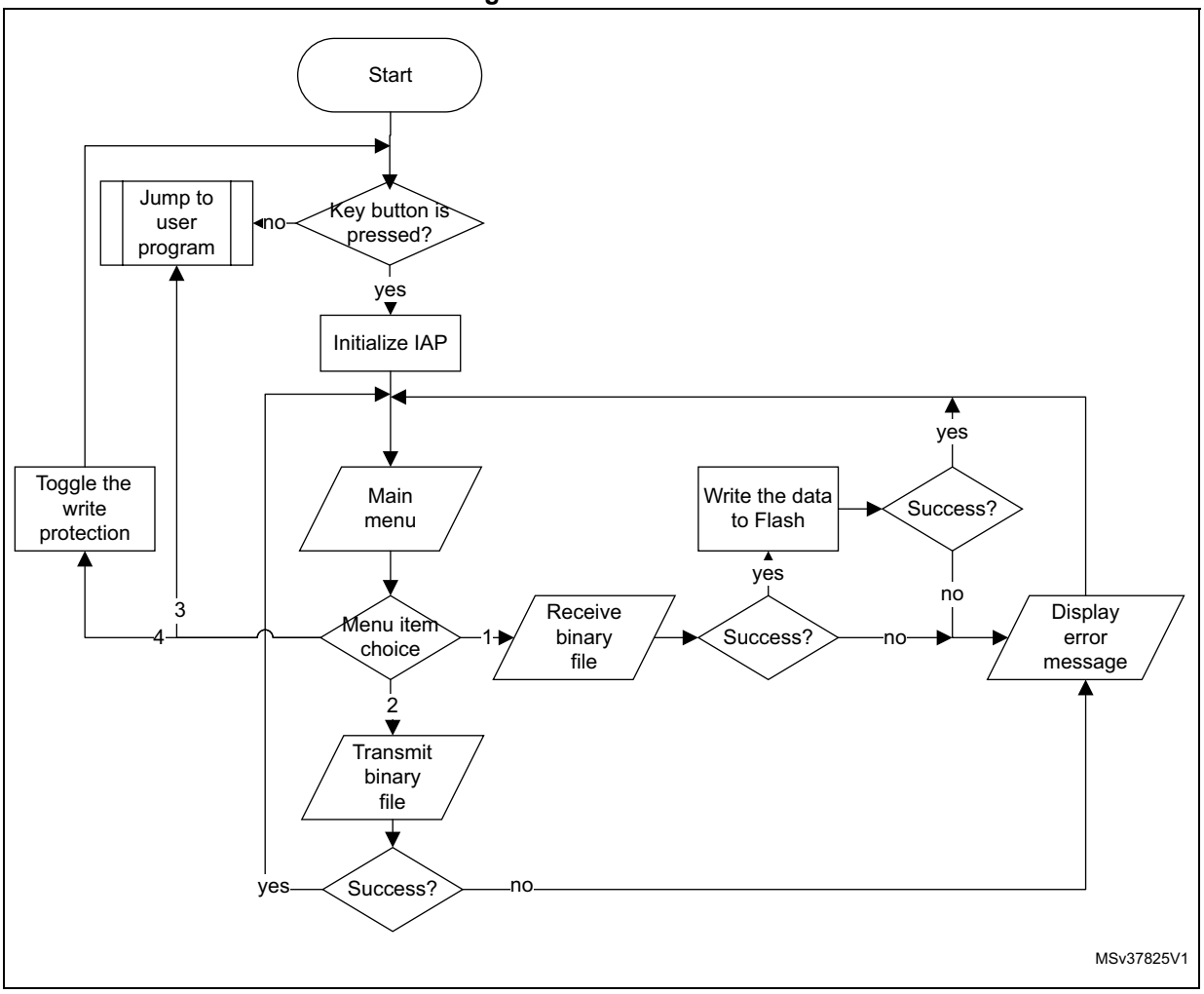
- not pressing the push-button at reset switches to the user application if it is present in the memory
- pressing the push-button at reset displays the IAP main menu (see *Figure 4*).

Refer to *readme.txt*, provided with the example code, for more details on your board implementation.

The IAP application flowchart is shown in *Figure 2*.

a. The YMODEM protocol sends data by 1024-byte blocks. An error check is performed in the transmitted data blocks to compare the transmitted and received data. Blocks unsuccessfully received are acknowledged with an NAK (negative acknowledgment). For more details about the YMODEM protocol, refer to the existing documentation.

**Figure 2. IAP flowchart**



# 2 Running the IAP driver

The IAP driver must be programmed via the JTAG/SWD interface, starting from the flash memory base address. This is performed either through a chosen development toolchain, or the factory-embedded bootloader located in the system memory area.

The IAP driver uses the USART to:

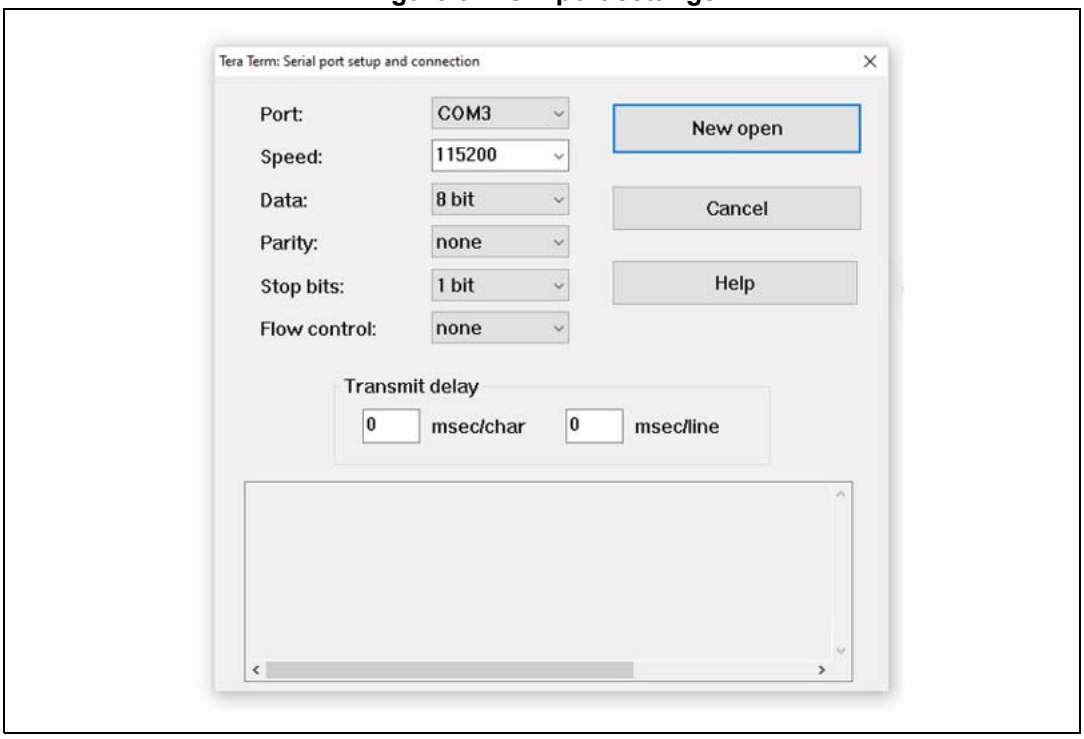
- download a binary file from the terminal emulator to the STM32 internal flash memory
- upload the internal flash memory content (starting from the defined user application address) of the STM32 MCU (based on Arm®(a) cores) into a binary file.

## 2.1 Tera Term configuration

To use the IAP, the user must have a PC running terminal emulator supporting the YMODEM protocol.

This document takes as an example the Tera Term open-source emulator version 4.106. The Tera Term COM port must be configured as shown in *Figure 3*. To do this, select **Setup** followed by **Serial port** from the application menu bar.

**Figure 3. COM port settings**




a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

*Note: The baud rate value of 115200 bps is used in the firmware examples.*

*Take care when selecting the system clock frequency. To ensure successful communication via the USART, the system clock frequency in the end application must support a baud rate of at least 115200 bps.*

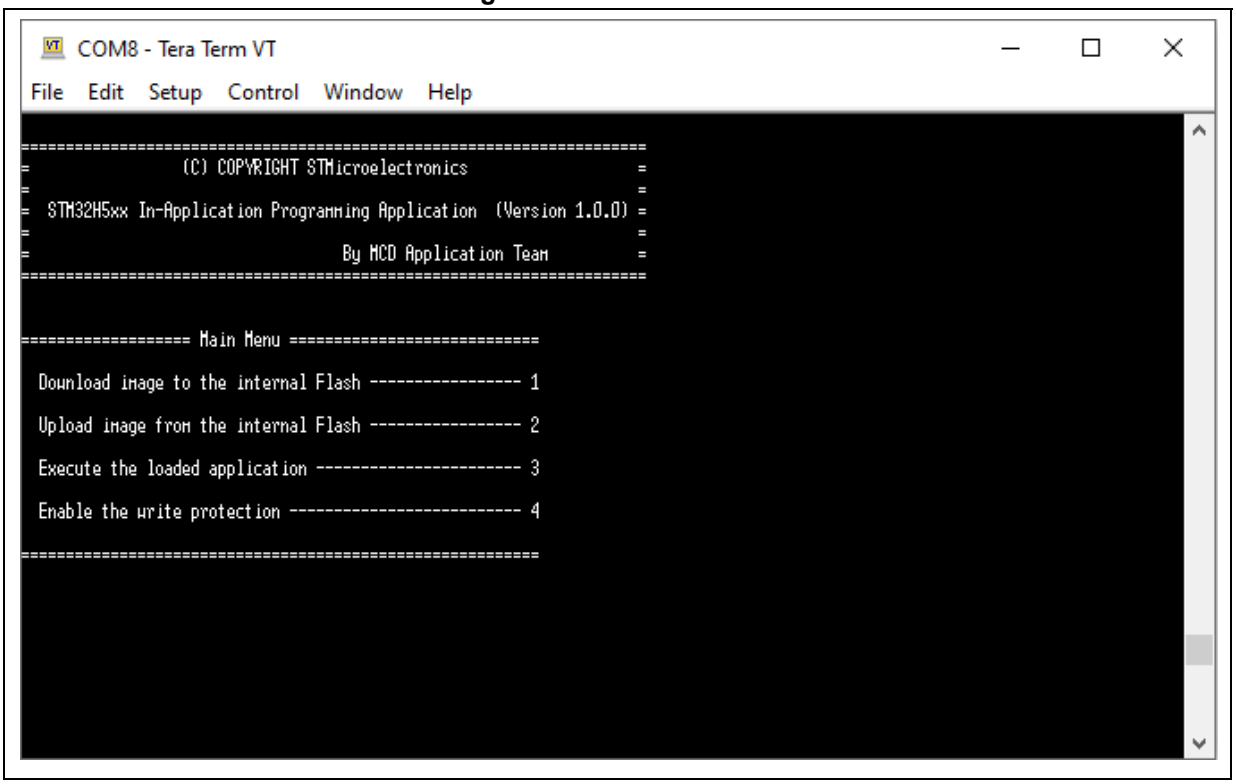
*The COM port number depends upon the user PC configuration. The examples are configured to use the ST link virtual COM port executing the IAP driver.*

In the example given in this application note, pressing the push-button at reset launches the IAP driver. In this way, the user runs the IAP driver to reprogram the STM32 microcontroller internal flash memory. Refer to the *readme.txt* file provided within the firmware package, and to your board documentation for the implementation details.

# 3 IAP driver menu

Running the IAP displays the following menu in the Tera Term window.

**Figure 4. IAP driver menu**




## 3.1 Downloading an image into the internal flash memory

Follow the sequence below to download a binary file via Tera Term into the STM32 internal flash memory:

1. Press **1** on the keyboard to select the **Download image to the internal Flash** menu option.
2. Select **File/Transfer/YMODEM/Send** from the Tera Term menu.
3. In Tera Term **YMODEM Send** dialog window, select the binary file of the application.
4. Click the **Open** button.
5. The IAP driver then loads the binary file into the internal flash memory, starting from the defined base address, and displays the binary file name and its size in the Tera Term window.

## 3.2 Uploading an image from the internal flash memory

Follow the sequence below to upload a copy of the internal flash memory starting from the user application address:

1. Press **“2”** on the keyboard to select **Upload image from the internal Flash** menu option.
2. Select **File/Transfer/YMODEM/Receive** from the Tera Term menu.
3. A file, containing a copy of the internal flash memory content, is saved in the last used folder.

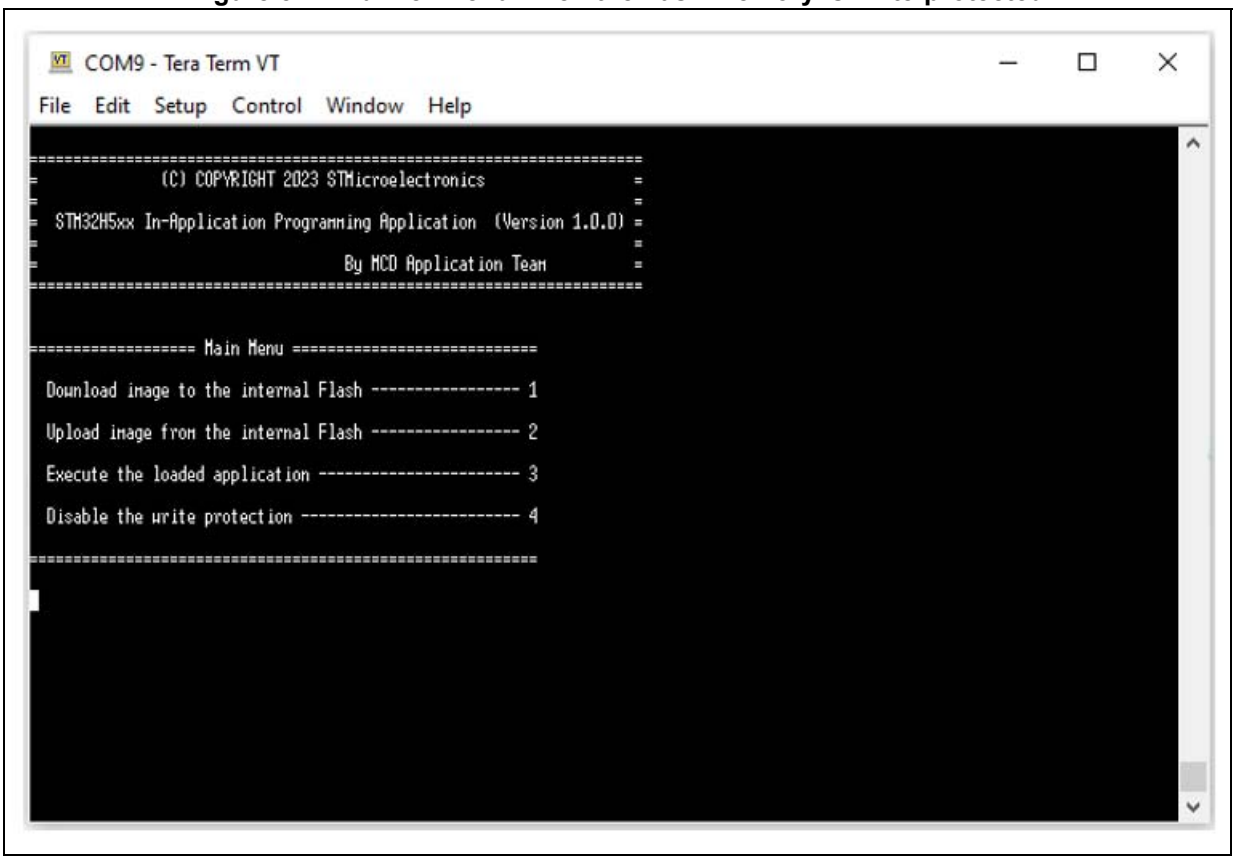
## 3.3 Executing the new program

Once the new program has been loaded, press “**3”** on the keyboard to select the **Execute the loaded application program** menu option and jump to the application code.

## 3.4 Configuring the write protection

When the IAP starts, it checks the flash memory pages where the user program is to be loaded, to see if any are write-protected. If it is the case, the menu shown in *Figure 5* is displayed.

**Figure 5. IAP driver menu when the flash memory is write-protected**



Before downloading the new program, the write protection must be disabled. To do so, press **4 Disable the write protection** on the keyboard. The write protection is then disabled and a system reset is generated to reload the new option byte values. After resuming from reset, the menu shown in *Figure 4* is displayed if the key push-button is pressed.

When a new user application is installed, the write protection may be enabled once again to prevent accidental corruption of the code. To toggle the write protection, select menu option "**4"** again.

*Note: In this example, the read protection is not supported, so the user must verify that the flash memory is not read-protected. Removing the read protection requires a mass erase of the nonvolatile memory content.*

# 4 IAP implementation summary

The IAP firmware package comes with:

- source files and preconfigured projects for the IAP program (*under IAP_Main sub-directory)*
- source files and preconfigured projects that build the application to be loaded into flash memory using the IAP (under *IAP_Binary_Template sub-directory*).

The *readme.txt* files provided within code package describes step by step how to execute this IAP application and contains microcontroller specific implementation details.

# 5 User program conditions

The user application to be loaded into the flash memory using IAP must be built with these configuration settings:

1. Using your toolchain linker settings, set the program load address as configured in the IAP project code.
2. Relocate the vector table to APPLICATION_ADDRESS, for example by modifying the value of the constant VECT_TAB_OFFSET defined in the project source files.

An example application program to be loaded with the IAP is provided with pre-configured projects. Refer to the project *readme.txt* file for details.

# 6 Revision history

**Table 1. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 27-Apr-2015 | 1 | Initial release. |
| 01-Apr-2020 | 2 | Updated *Introduction* and *Section 2.1: Tera Term configuration*. |
| 28-Feb-2023 | 3 | Updated *Section 2.1: Tera Term configuration*.<br>Updated *Figure 3: COM port settings*, *Figure 4: IAP driver menu*, and *Figure 5: IAP driver menu when the flash memory is write-protected*.<br>Minor text edits across the whole document. |