AN1720
Application Note

Managing the Read-out Protection in Flash Microcontrollers

By DTV - Monitor MCU Applications Lab

Introduction

Once a MCU has been programmed with its final software, it may be protected against piracy by forbidding any further read-out of its contents.

Each Flash MCU (either HDFlash or XFlash type) has this capability, by means of a freely programmable option byte, but each kind of programming tool requires a different method to enable it.

The examples and procedures below are fitted for the ST7FLCD1 MCU, with an HDFlash memory array. These guidelines are applicable to any other MCU (even XFlash) but all MCU-related features, like option bytes, default values, enabled/disabled states, compiler options etc.. must be carefully reviewed and fitted for the new target MCU.

1 Read-out Protection Principle

The protection against further read-out consists of a dedicated configuration bit, named **FMP_R**, to program to the desired value. This configuration bit is located in the Option Byte 1, as described in the *Flash Program Memory* chapter of the ST7FLCD1 datasheet:

STATIC OPTION BYTE 1

	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
								FMP_R
Default	1	1	1	1	1	1	1	1

OPT0= **FMP_R** *Flash memory read-out protection*
The bit0, **FMP_R**, indicates if the user flash memory is protected against read-out piracy:

0 Read-out protection enabled

1 Read-out protection disabled

By default, this bit is set to 1 (protection disabled). Once programmed to 0, the read-out protection is enabled after the next reset of the MCU.

The program and data stored in the Flash program memory are then protected against read-out piracy (including a re-write protection) : the contents of the MCU can no longer be read or verified. In addition, no programming tool can bypass this protection, any attempt to do so will result in an error message.

If this protection is to be removed by reprogramming the Option Byte, the entire program memory is automatically wiped out, making it impossible to read the contents of the MCU by any means.

2 Programming the Option Byte

The Option Byte 1, where the read-out protection bit is located, is a regular Flash memory byte but does not pertain to the regular Flash program memory array of the MCU. As such, it is accessible in read and write mode by means of a separate procedure, which depends on the software and hardw are tool used for programming the MCU and is completely independent of the other program memory array.

The following operations describe how to program and clear this option byte on the following different hardware tools:

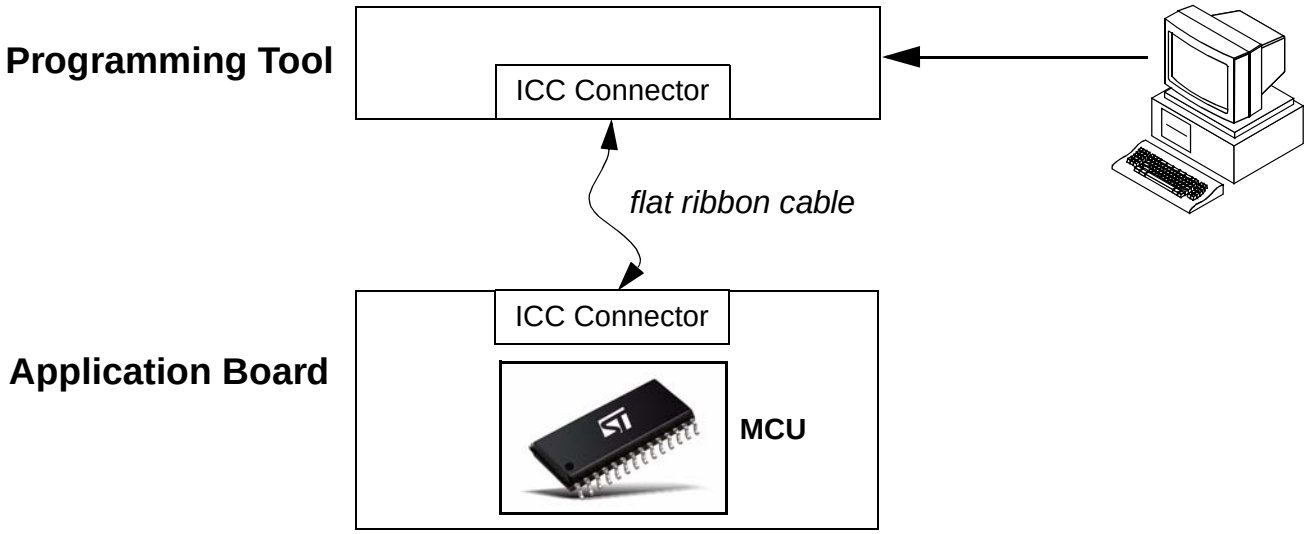
- ST7 STICK tool (maker: ST) under ST7 Visual Programmer (STVP7) software tool
- STMC-ICC tool plugged into the ST7FLCD-EMU3 emulator (maker: ST)
- FLASHER ST7 (maker: Segger, *Germany*)

The first two work under the ST7 Visual Programmer (STVP7) software tool.

The last one works under a proprietary software tool.

Each tool connects itself to the ST7FLCD1 MCU by means of a standard ICC cable (flat ribbon cable with HE-10 connectors on both ends).

There is no Eprom Programming Board (EPB) proper for the MCU, the only way to program it is to connect the ICC cable from the programming tool to its matching ICC connector on the application board where the MCU is mounted:



2.1 Programming the option byte “manually”

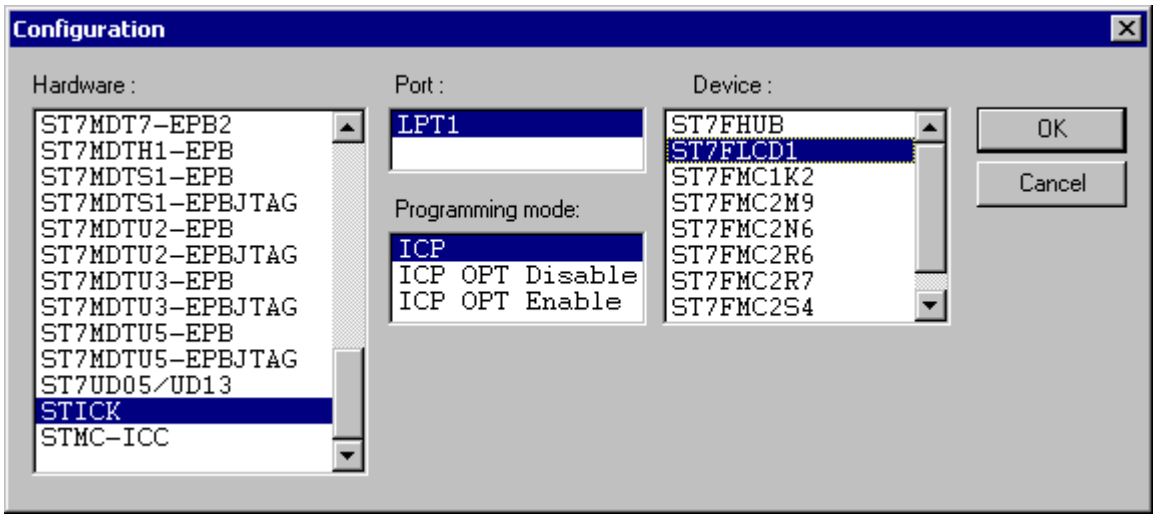
The way to program and erase the option byte directly, as described below, is the simplest and easiest way but is not quite suited for automatic programming, because the option byte configuration must be reloaded each time the programming software is re-run.

2.1.1 Under STVP7 with ST7 STICK

2.1.1.1 Configuring STVP7

Prior to programming an ST7FLCD1 MCU, the STVP7 software must be configured accordingly. STVP7 software version **1.7.0** or above is required to program this MCU. Updates can be freely downloaded from the ST web site *http://www.stmcu.com*, section “downloads”.

The “Configure/Configure ST7 Visual Programmer” menu entry launches a configuration window, which must be set as follows to work with the STICK:



Note: The STICK can only be connected to the computer by means of the parallel port (LPT1).

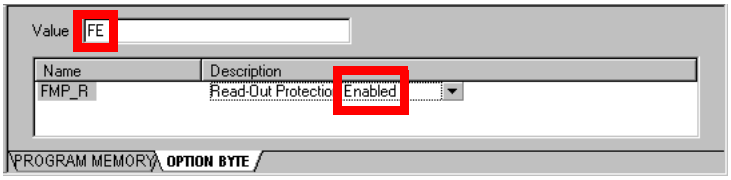
2.1.1.2 Enabling the Read-Out Protection

By default, the STVP7 window opens on the “program memory” area. Clicking on the “option byte” tab on the bottom of the main window launches the following screen:

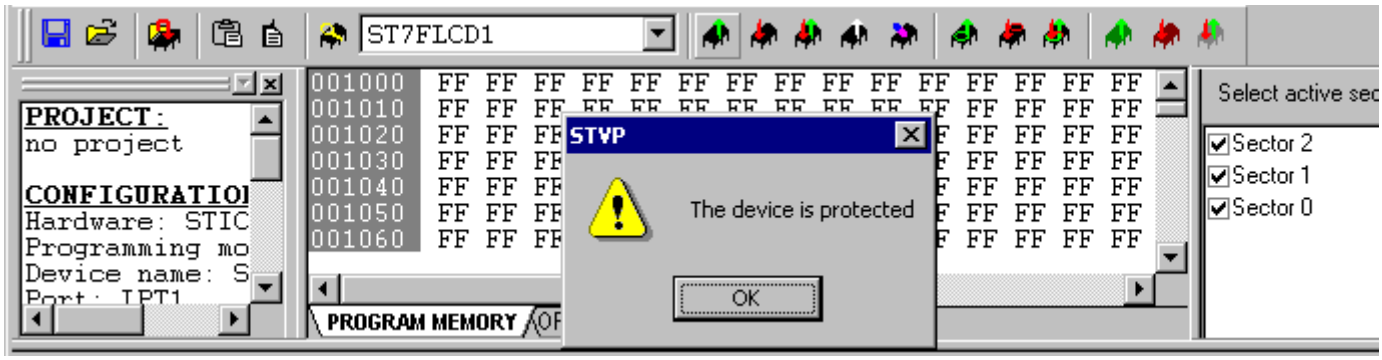
This is the specific window where the read-out protection bit **FMP_R** can be read, programmed and verified like any other Flash memory location, thanks to the 3 icons on the top bar:

By default on a *fresh* MCU, the read-out protection bit comes up as disabled, which means that bit 0 of option byte 1 is set to 1. To enable it, the “**Read-Out Protection Enabled**” entry must be selected by clicking on the right arrow; alternately, the new option byte value can be directly entered in the “Value” field (only bit 0 is meaningful).

Then the option byte can be programmed and verified by means of the 2nd icon “Program” above. The “value” field (the actual value of the Option Byte 1) has also changed to **0xFE**, since bit 0 is now cleared:



From now on, any attempt to read, program, verify or erase the sector(s) of the main Flash program memory array from the “Program Memory” tab will give an error message:

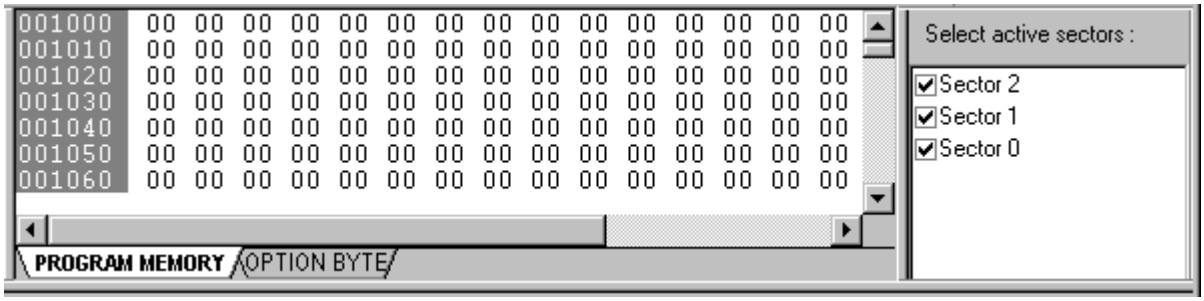


2.1.1.3 Disabling the Read-Out Protection

Once enabled, the read-out protection can only be removed by clearing the entire contents of the Flash memory array: this means that the program memory will be filled with 00s, making it useless to read back. The STVP7 tool is not responsible for this, this is done at the chip level by embedded algorithms which cannot be bypassed (*hardware protection*).

The protection removal is available again from the “Option Byte” tab, by selecting the “**Read-Out Protection Disabled**” entry value, and then programming the option byte again by the same “Program” icon. This operation takes slightly more time because the MCU is not programming one bit alone but its entire Flash memory array.

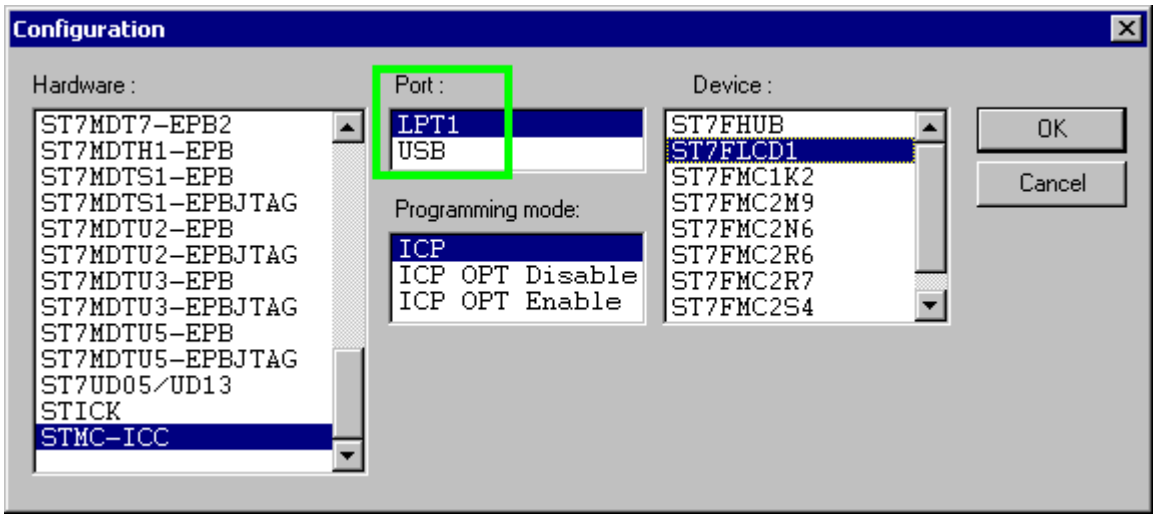
Once completed, the Flash memory array can be read again but the main window shows that it is completely filled up with **00s**: the previously stored software has been wiped out.



To reprogram the MCU, it is therefore needed to erase the sector(s) first, as usual.

2.1.2 Under STVP7 with STMC-ICC Tool

It works exactly in the same way as the ST7 STICK described above, the configuration must simply be changed to match the new hardware:



Since the programming is now done through the ST Microconnect box of the EMU3 emulator, it can be connected to the computer by means of the parallel port (LPT1) but also the USB port.

The rest of the operating mode remains strictly the same. If the USB port has been chosen, the time it takes to program/erase/verify a chip will be significantly shorter.

2.1.3 With Segger FLASHER ST7 Tool

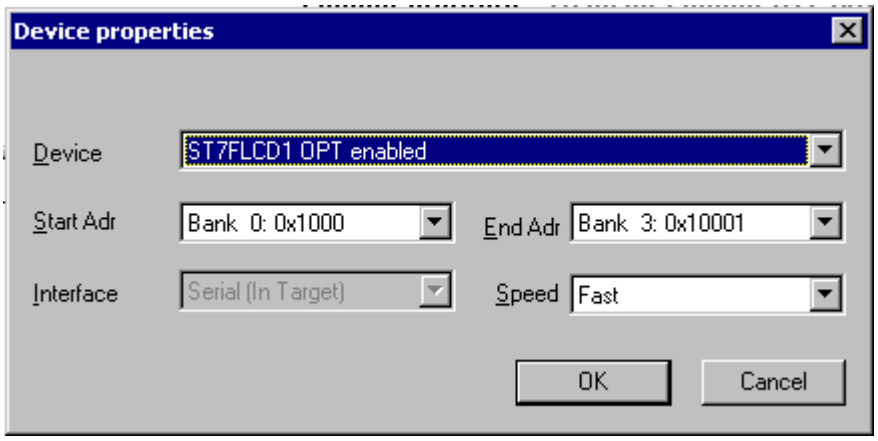
This tool has a different user interface, which needs some time to get used to.

The software itself is completely different, and the hardware tool (the Flasher blue box) is connected to the PC by means of a serial port (COM1 or else).

2.1.3.1 Configuring the FLASHER Tool

Prior to programming an ST7FLCD1 MCU, the FLASHER software must be configured accordingly. Segger software version **1.76c** or above is required to program this MCU. Updates can be freely downloaded from the Segger web site *http://www.segger.com*, section “download”.

The “Options/Device..” menu entry launches a configuration window, which must be set as follows to work with the FLASHER:



*Note: The FLASHER blue box **must** be connected to the PC to configure the software properly, otherwise the “Options/Device..” menu entry is grayed out and cannot be selected.*

The **Device** field is where the MCU type is chosen. The two options “OPT enabled” and “OPT disabled” are of no concern to the ST7FLCD1, and both configurations work.

By means of the **Start Adr** and **End Adr** fields, the user can freely choose which sector(s) will be erased, programmed or verified by the FLASHER:

- The **Start Adr** field defines the beginning of the memory array area to program
- In a similar way, the **End Adr** field defines the end of the memory array area to program

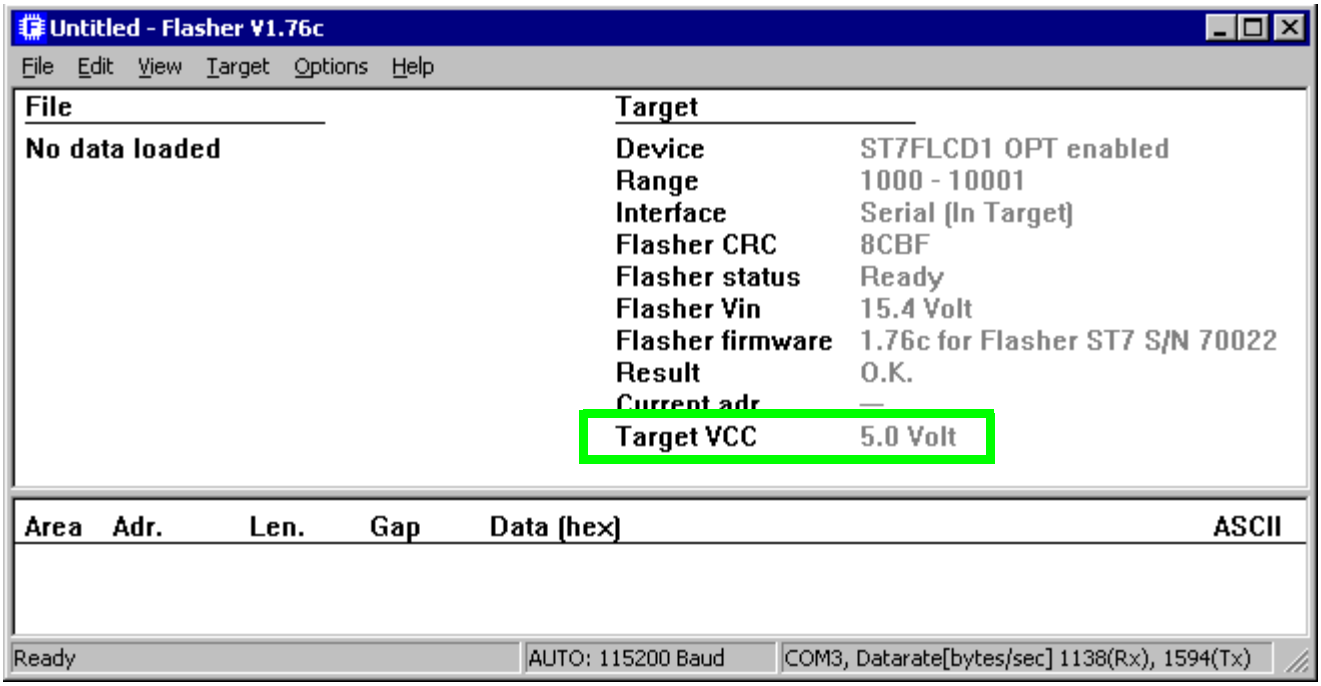
The FLASHER uses the word “bank” instead of sector, and the bank numbers are slightly different from their corresponding sector numbers:

Bank Number →	Sector Number	Start Address	End Address
0	2	0x1000	0xDFFF
1	1	0xE000	0xEFFF
2	0	0xF000	0xFFFF
3	N/A	Option Byte 1 (0x10000)	Option Byte 2 (0x10001)

The first 3 banks (banks 0, 1 and 2) have a matching sector in the usual 60KB memory space of the Flash memory array, between 0x1000 and 0xFFFF.

The last bank (bank 3) has no matching sector: it allows access to the two programmable Option Bytes 1 and 2, and is defined outside the normal 64KB (16-bit) memory space at virtual addresses 0x10000 and 0x10001.

Once everything has been configured properly, and a running ST7FLCD1 MCU has been connected to the FLASHER by means of the ICC cable, the main window displays the following status under the “Target” right hand side:

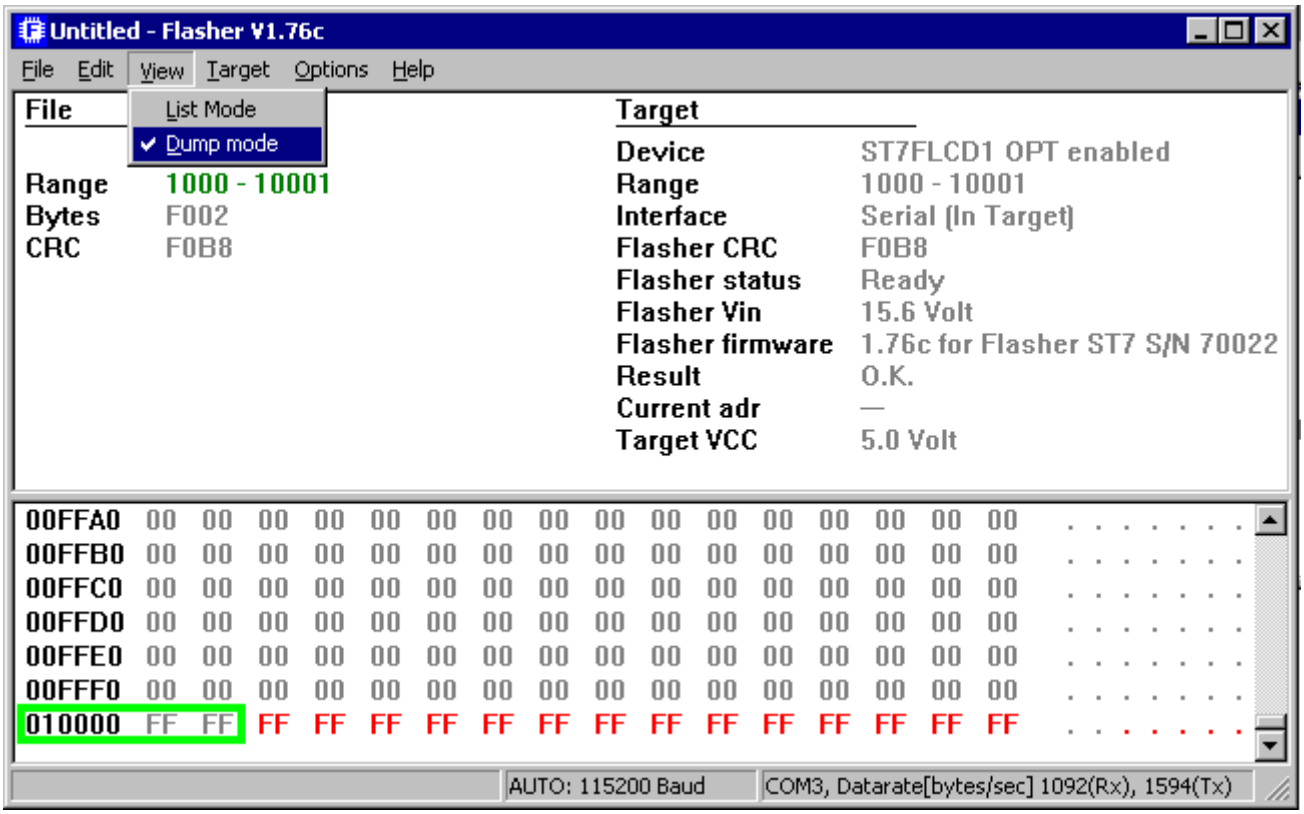


*Note: The FLASHER blue box **must** be connected to a properly powered ST7FLCD1 MCU with its own clock (crystal, oscillator or else), otherwise the window will display: “***Target VCC 0.0 Volt***” in red.*

2.1.3.2 Enabling the Read-Out Protection

The contents of the ST7FLCD1 MCU must be first downloaded to the PC, in order to work on them. Clicking on the “Target/Read memory” menu entry will upload the chip contents to the PC.

The bytes will then be displayed into the 2nd lower window, either in “list” mode (the bytes are grouped by packets of consecutive addresses) or in “dump” mode (similar to STVP7) as shown below:



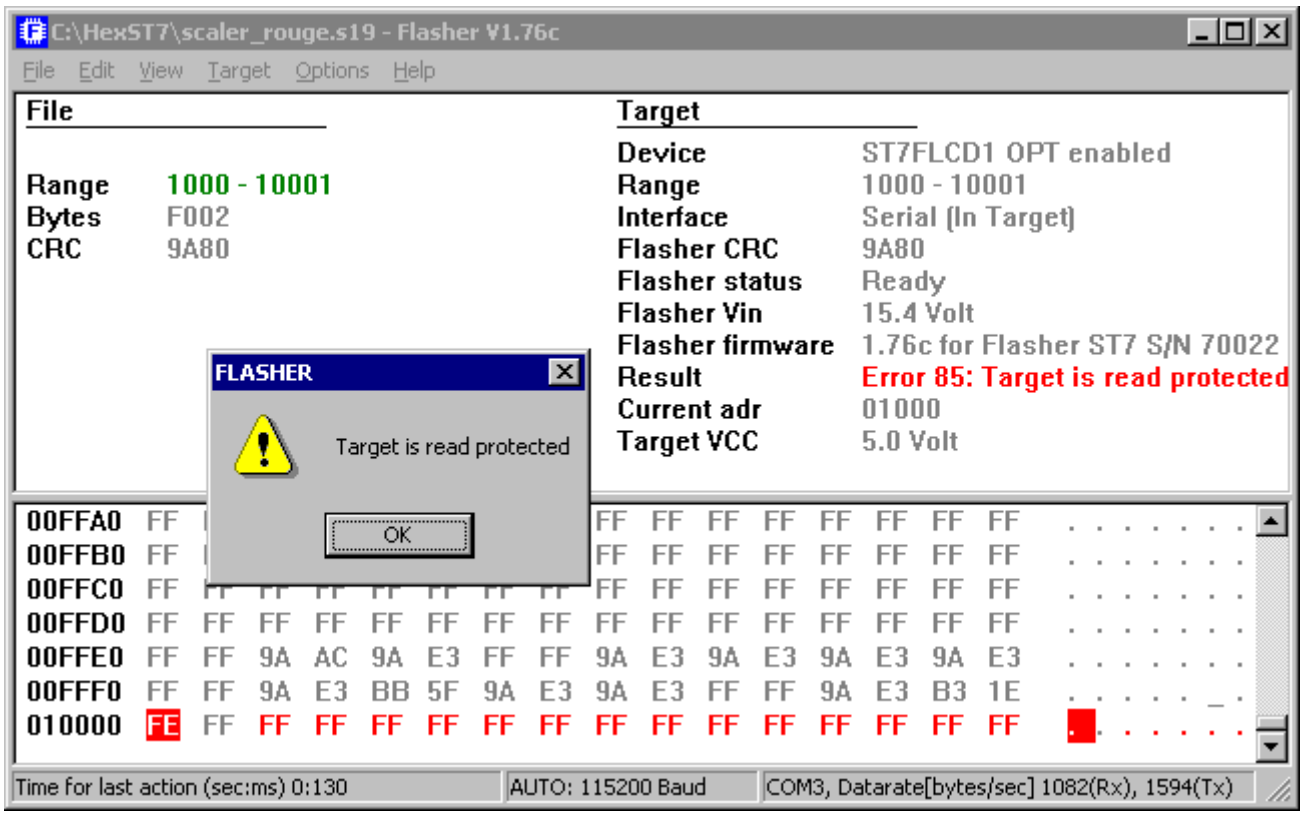
Scrolling down till the very end of the memory will show the two Option Bytes at addresses 0x10000 and 0x10001. The subsequent bytes in red are meaningless.

To enable the read-out protection, the value at 0x10000 must be changed to **0xFE** (bit 0 = 0).

Different contents can be programmed into the MCU, but must be merged first to the existing contents (which have just been downloaded) by means of the “File/Merge” menu. Do **not** load a S19 file, but merge it! Otherwise, the previously loaded contents will be overwritten and the two Option Bytes will likely disappear.

Finally, the chip can be erased (“Target/Clear” menu entry, or F6 function key) and reprogrammed (“Target/Program & Verify”, or F7 function key).

If the user tries to read the chip contents afterwards, the following error message shows up:



2.1.3.3 Disabling the Read-Out Protection

This is achieved by simply clicking on the “Target/Clear Readout protection” menu entry.

Then the FLASHER will work on its own: the entire ST7FLCD1 MCU contents will be wiped out (filled with 00s) and, unlike STVP7, erased immediately afterwards. This leaves the MCU in an erased state, ready for a new programming cycle. On STVP7, clearing the read-out protection bit would leave the MCU programmed with 00s, this would require a “manual” erasure later on (refer to Section 2.1.1.3).

2.2 Enabling the Read-out Protection automatically

The procedure previously described has a major drawback: it must be repeated for every new chip to program. There is an easy way to make it fully automatic, each time a new S19 file is rebuilt (assembly toolchain, C language etc).

The example provided for C language application is fitted for Metrowerks/Hiware C toolchain.

2.2.1 Under STVP7, with the STICK or the STMC-ICC

2.2.1.1 Creating the Morotola file

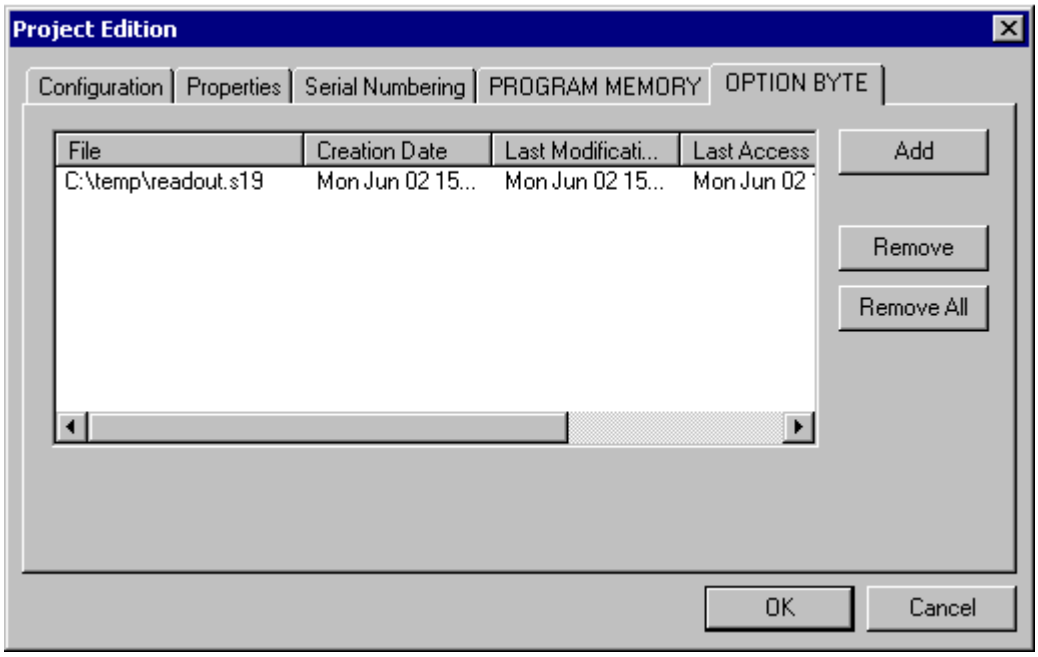
First, the read-out protection bit must be enabled as described in Section 2.1.1.2, and then saved with the “File/Save As..” menu entry. This creates an S19 file that only contains the specific Option Bytes configuration for STVP7:

```
S1048020FE5D
S9030000FC
```

This file must be set aside for later use, let’s call it “readout.s19”.

2.2.1.2 Embedding the option bytes into the project

In a 2nd step, a project must be created, as described in the other application note AN1658 *Automatic Serial Number Generation in MCU* (or refer to the on-line help), and then the “Option Byte” tag must be set to load the previously created “readout.s19” file:



Once the project has been fully configured, the programming can be started by clicking:

- either on “All tabs” in the “Program” menu
- or on the icon shown below:

Then the list of actions enabled in the “Properties” Window will be executed by STVP7, including the programming of the Option Bytes.

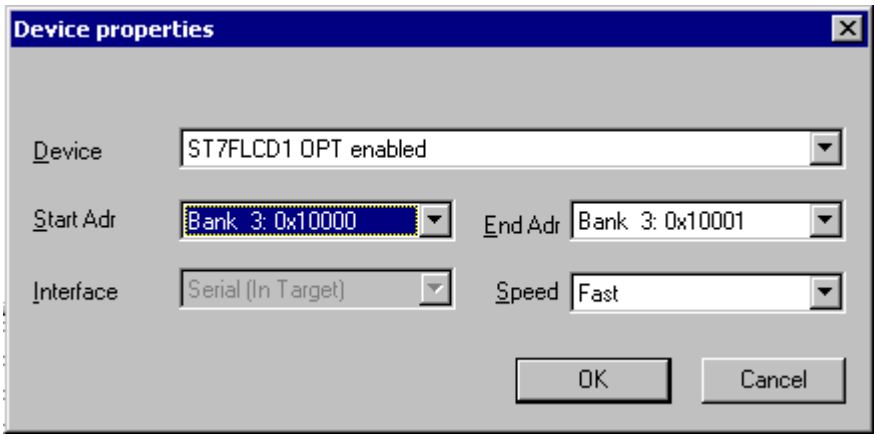
2.2.2 With the FLASHER

The software does not manage projects the way STVP7 does, therefore the option bytes configuration must be embedded inside the Motorola file to program into the MCU.

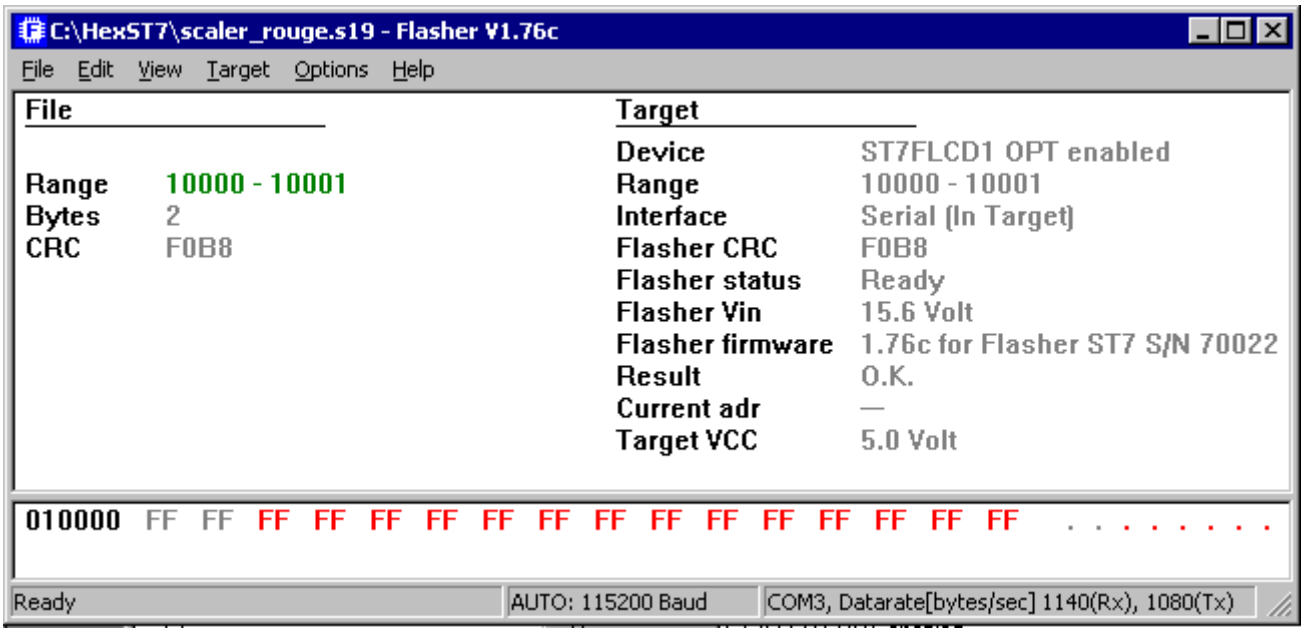
2.2.2.1 Creating the Morotola file

First, in a similar way as described above for STVP7, a Morotola file must be created with the desired option bytes configuration only. Bringing the “Options/Device..” configuration window (refer

to Section 2.1.3.1), the **Start Adr** and **End Adr** fields must be set to enclose the bank 3 alone, where the option bytes are located:



Clicking on the “Target/Read memory” menu entry will upload the chip contents to the PC, but only the two Option Bytes:



The 1st byte 0xFF at address 0x10000 must be changed to **0xFE**, and then the file must be saved with the “File/Save As” menu entry.

This creates a very short S29 file (= S19 file with addresses beyond 0xFFFF), which contains the following bytes alone for the Flasher:

```
S206010000FEFFFB
S9030000FC
```

The read-out protection can now be enabled “manually”, by copy-pasting the contents of the previously created S29 file at the very bottom of the S19 file created by the development toolchain. But each time the software is rebuilt, a new S19 file will be created and those lines will be removed.

The best solution is to force the C toolchain to add those few bytes into the S19 file, transparently and automatically each time the software is rebuilt.

2.2.2.2 Embedding the option bytes inside the final S19 file automatically

Thanks to the Metrowerks support team, two different ways are possible, both work well:

METHOD 1: Use the linker command VECTOR ADDRESS

- Open the PRM file
- At the **end** of the PRM file, add the following line:
  VECTOR ADDRESS 0x10000 0xFEFF
  This adds a fake vector at address 0x10000 made of a 16-bit data field: 0xFE and 0xFF, which are the two 8-bit values to write the option bytes with
- Save the updated PRM file
- Open the MAKEFILE
- After the $(LINK) line of the .abs section, add the following lines:

```
test.abs : $(ENV) $(OBJ_LIST) test.prm
    $(LINK) test.prm
    burner.exe OPENFILE "C:\test.s19" \
      format=motorola \
      busWidth=1 \
      origin=0 \
      len=0x10002 \
      destination=0 \
      SRECORD=Sx \
      SENDBYTE 1 "c:\toto\obj\test.abs"
```

The directories must be changed according to your project.
Or, if this section already exists, change “len=0x10000” to “len=0x10002”.

Note: The above changes can also be done in the burner’s “.bbl” file, if used.

- Save the updated MAKEFILE

Now each time the application is rebuilt, the generated “test.S19” file contains the following record: **S107FFFExxxxFEFFE4** (xxxx being the reset vector).

METHOD 2: Use the compiler to define a variable at an absolute address

- Open one of the source files in the application, add the following line:
  const int ID @0x10000 = 0xFEFF;
  This line defines a 2 byte variables called “ID”, allocates it at address 0x10000 and initializes it with the value 0xFEFF
- Save the updated source file
- Open the PRM file and add the following line in the middle of the file, for example between “NAMES..END” and “SECTION” entries:
  ENTRIES ID END
  This will make sure variable ID is linked to the application.
- Save the updated PRM file
- Go to METHOD 1 above and follow the procedure starting at the 4th line: “Open the MAKEFILE” and follow the procedure till the end

Now each time the application is rebuilt, the generated “test.S19” file contains the following record: **S107FFFExxxxFEFFE4** (xxxx being the reset vector).

Once the S19 file is loaded by the FLASHER software, the final 2 bytes at addresses 0x10000 and 0x10001 will be automatically identified as option bytes and programmed accordingly.