

# Flash\_Programming\_1 for KIT\_AURIX\_TC397\_TFT

## Flash programming

AURIX™ TC3xx Microcontroller Training  
V1.0.0



[Please read the Important Notice and Warnings at the end of this document](#)

## Scope of work

---

**This example shows how to flash the Program Flash memory and the Data Flash memory.**

In this example, 64 Bytes of the Program Flash memory (PFLASH) are flashed and verified afterwards. Furthermore, 64 Bytes of the Data Flash memory (DFLASH) are flashed and verified.

Before any write operation, the flash memories are erased. If the flash operation is successful, an LED for each tested memory is turned ON.

# Introduction

---

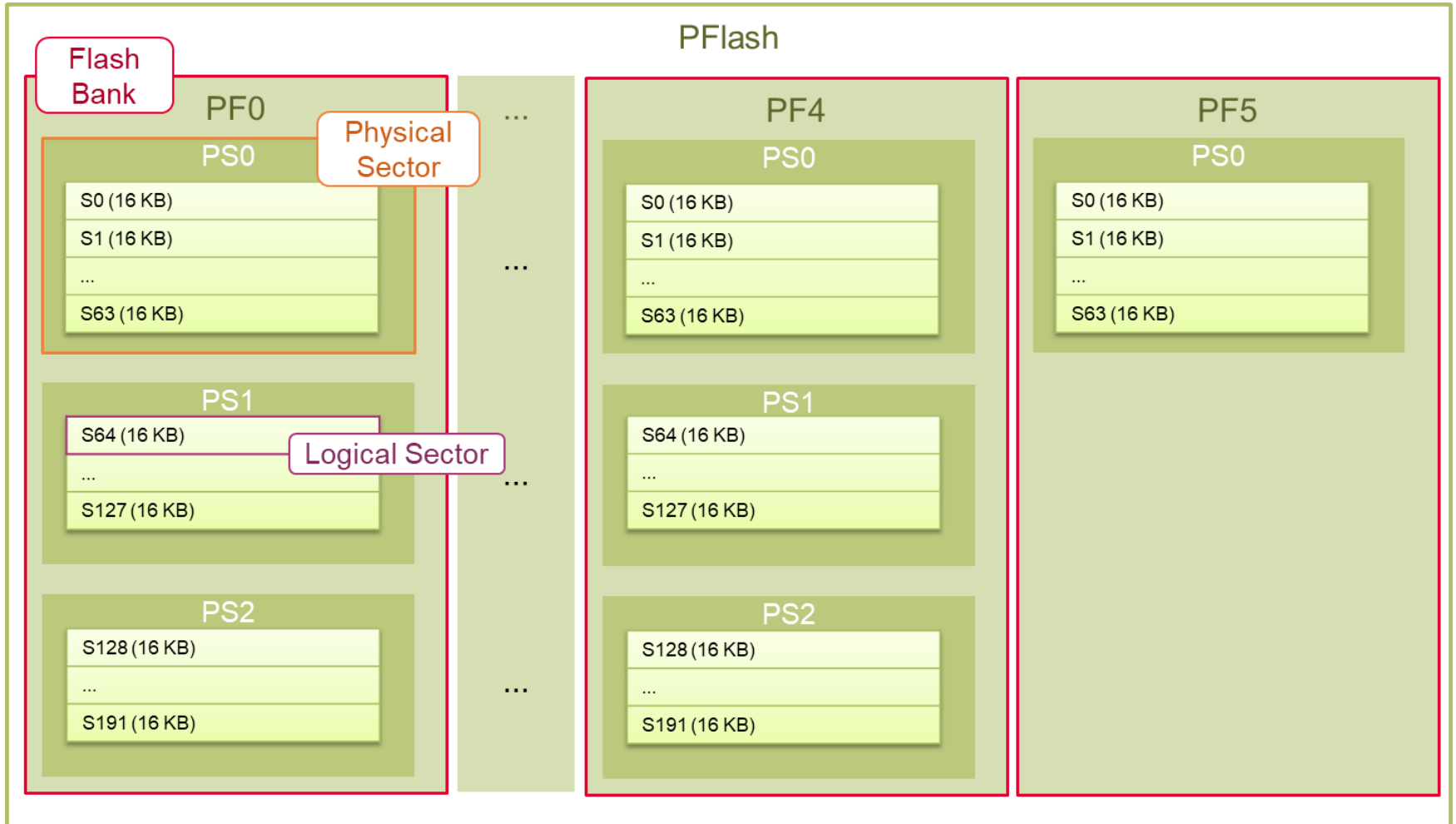
- › The Data Memory Unit (DMU) controls command sequences executed on the Program and Data Flash memories (PFLASH and DFLASH), interfacing with the Flash Standard Interface (FSI) and the Program Flash Interface (PFI)
- › The FSI executes erase, program and verify operations on all flash memories.
- › The PFI provides a unique point-to-point fast connection for each PFLASH bank to a CPU.
- › The AURIX™ TC39x device features:
  - 6 Program **Flash Banks** (PFx)
  - 2 Data **Flash Banks** (DFx)
- › The AURIX™ TC3xx devices feature PFLASH Banks PFx based on the same sector structure. PFx banks may vary in size:
  - 3 Mbyte Program Flash Bank
  - 2 Mbyte Program Flash Bank
  - 1 Mbyte Program Flash Bank

# Introduction

- › AURIX™ TC39x features five Program Flash banks (PF0..PF4) with dimension of 3 Mbyte and one Program Flash bank (PF5) with dimension of 1 Mbyte. Each Program Flash bank is divided into **Physical Sectors** with dimension of 1024 Kbytes and each Physical Sector is divided into 64 **Logical Sectors** with dimension of 16 Kbytes
- › AURIX™ TC39x features two Data Flash banks, DFLASH0 and DFLASH1. Both include multiple EEPROM sectors commonly used for EEPROM emulation. Only DFLASH0 includes User Configuration Blocks (UCBs) for data protection and a Configuration Sector (CFS), which is not directly accessible by the user.
- › The DFLASH EEPROM can be either configured in single ended mode (default) or in complement sensing. Depending on the selected mode, the size of each sector is set to 4 Kbytes and respectively 2 Kbytes.
- › The minimum amount of data that can be programmed in a flash memory is a **page**
  - Program Flash pages are made of 32 Bytes
  - Data Flash pages are made of 8 Bytes
- › A page can be programmed only after an erase operation
- › The smallest unit on which an erase operation can be performed is a Logical Sector

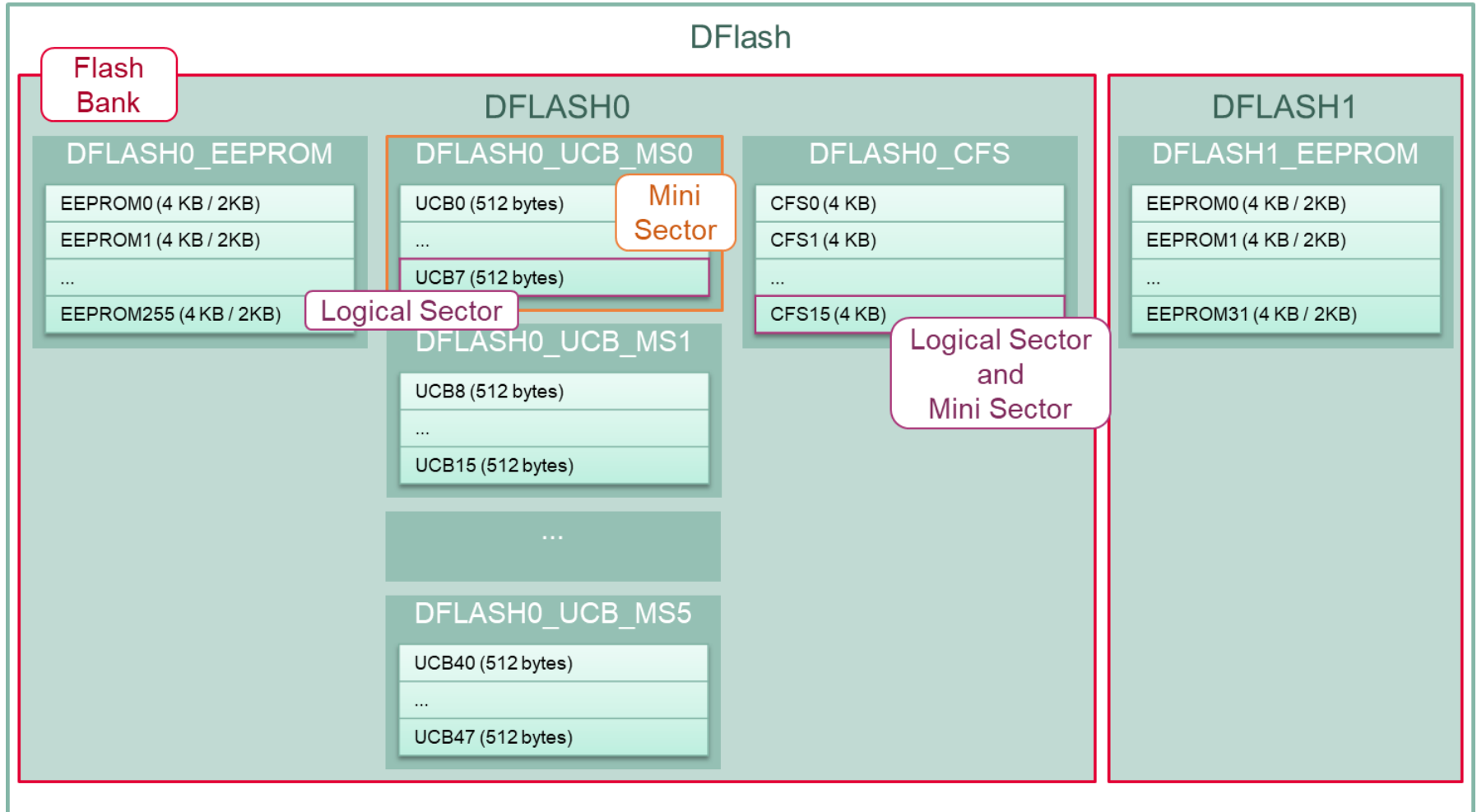
# Introduction

> AURIX™ TC39x Program Flash memory



# Introduction

## > AURIX™ TC39x Data Flash memory



# Introduction

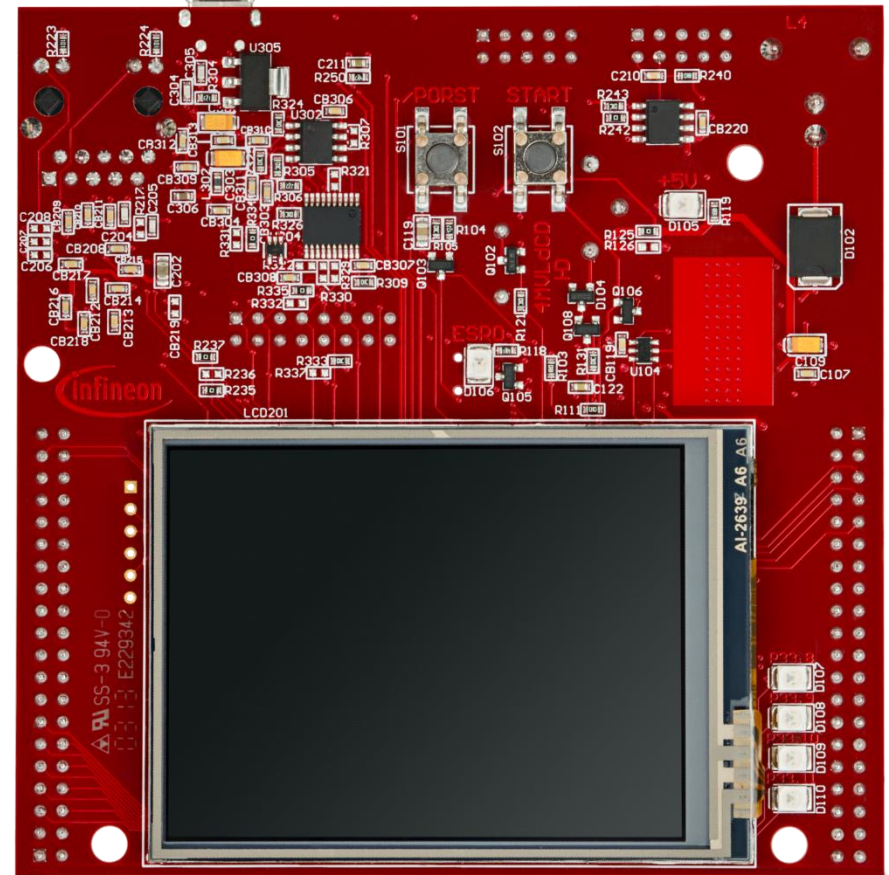
---

- › All the flash operations are performed with command sequences
- › The DMU has a Command Sequence Interpreter (CSI) to process command sequences
- › A minimum sequence of commands for programming the Program Flash memory or the Data Flash memory, is the following:
  - **Erase the Logical Sectors** to be programmed afterwards
  - Wait until the flash memory is ready (not busy)
  - **Enter page mode**
  - Wait until the flash memory is ready (not busy)
  - **Load data to be written in a page**
  - **Write the page**
  - Wait until the flash memory is ready (not busy)

**Note:** Code that performs PFLASH programming or erasing should not be executed from the same PFLASH.

# Hardware setup

This code example has been developed for the board  
KIT\_A2G\_TC397\_5V\_TFT.





# Implementation

---

## Flashing the Program Flash memory

To perform PFLASH programming, it is recommended to run the code from a memory location different from the PFLASH that is going to be programmed. For this reason, before starting the flash operations, the erase and program routines are copied in the Program Scratch-Pad SRAM (PSPR) of the CPU0 by the function ***copyFunctionsToPSPR()***. This uses the ***memcpy()*** function from the standard c library ***string.h*** and assigns a function pointer to the new memory location.

Then, the actual flash programming operations start by erasing the involved Logical Sectors.

## Erase of Logical Sectors

To perform an erase operation, writes have to be enabled on the PFLASH by clearing the EndInit Protection bit, done through the function ***lfxScuWdt\_clearSafetyEndinitInline()***. Then, the erase command sequence for one or more consecutive Logical Sectors can be executed through the ***lfxFlash\_eraseMultipleSectors()*** function, executed from the PSPR by the function pointer ***eraseSectors()***.

Finally, the EndInit Protection bit must be set again through the function ***lfxScuWdt\_setSafetyEndinitInline()***.

The function ***lfxFlash\_waitUnbusy()***, called by the function pointer ***waitUnbusy()***, stalls until the sector is erased and the PFLASH is ready again.

# Implementation

## Write operations

After erasing the needed Logical Sectors, the write operations can start.

The function ***IfxFlash\_enterPageMode()*** called from the PSPR by the function pointer ***enterPageMode()*** is used to enter page mode.

The function ***waitUnbusy()*** is used to stall until the PFLASH is ready and then, the data that has to be written in a page is loaded calling repetitively the function ***IfxFlash\_loadPage2X32()***, executed from the PSPR with the function pointer ***load2X32bits()***.

The loaded page is then written by calling the function pointer ***writePage()***, which executes the ***IfxFlash\_writePage()*** function from the PSPR (before and after the write operation, the EndInit Protection bit is cleared and respectively set).

Finally, ***waitUnbusy()*** is called to wait until the page has been written and the PFLASH is ready again then, the write process can be repeated until all the required data has been successfully written in the PFLASH.

After the flashing operations, the data is read from the PFLASH exploiting the macro ***MEM(address)***, and, if it is correct, an LED is turned on.

All the functions used for executing the command sequences for the flashing operations can be found in the iLLD header ***IfxFlash.h***, while the function pointers are declared and assigned in the ***Flash\_Programming.c*** file.

# Implementation

---

## Flashing the Data Flash memory

The procedure for flashing the Data Flash memory is the same used for flashing the Program Flash memory, but in this case the functions for executing the command sequences for erasing, waiting, loading and writing can be called directly from the PFLASH, thus copying them into the PSPR is not needed.

After the flashing operations, the data is read from the DFLASH exploiting the macro ***MEM(address)***, and, if it is correct, an LED is turned on.

## Configure and control the LEDs

Two LEDs are configured using methods from the iLLD header ***IfxPort.h***.

The port pins to which the LEDs are connected are **configured as push-pull output** using the function ***IfxPort\_setPinMode()***.

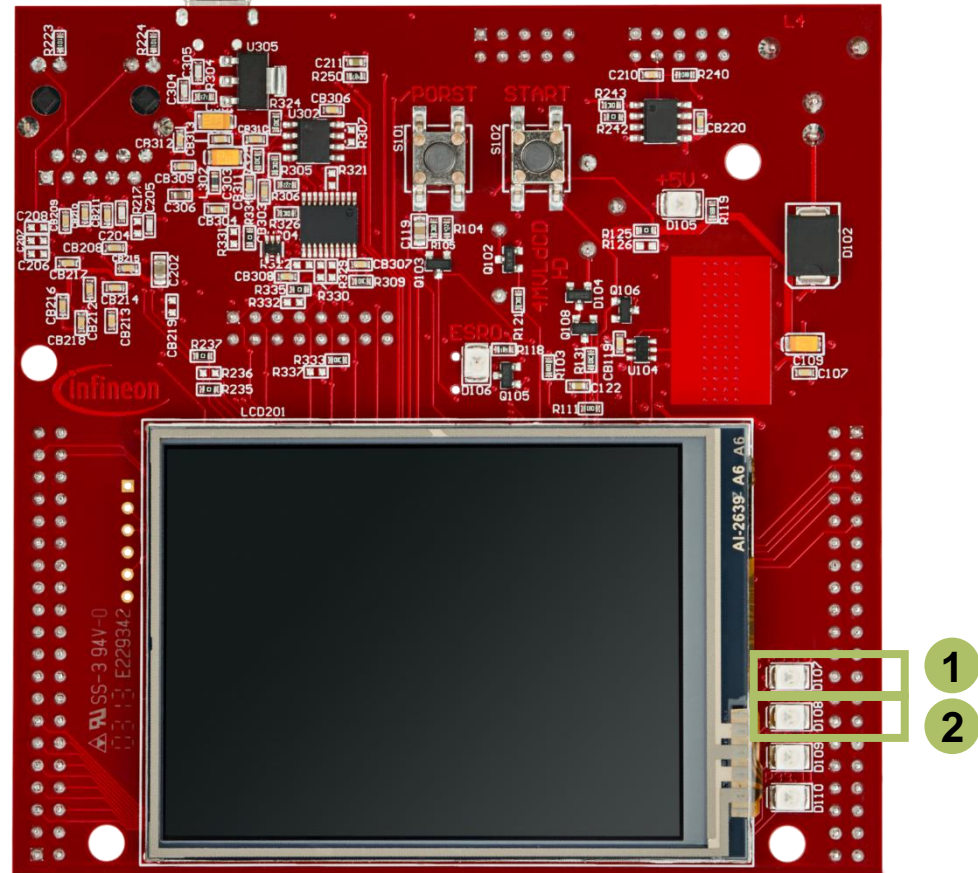
To turn on and off the LEDs, the function ***IfxPort\_setPinState()*** is used.

# Run and Test

After code compilation and flashing the device, observe the behavior of the LEDs.

Check that **LED1** (1) and **LED2** (2) are turned on:

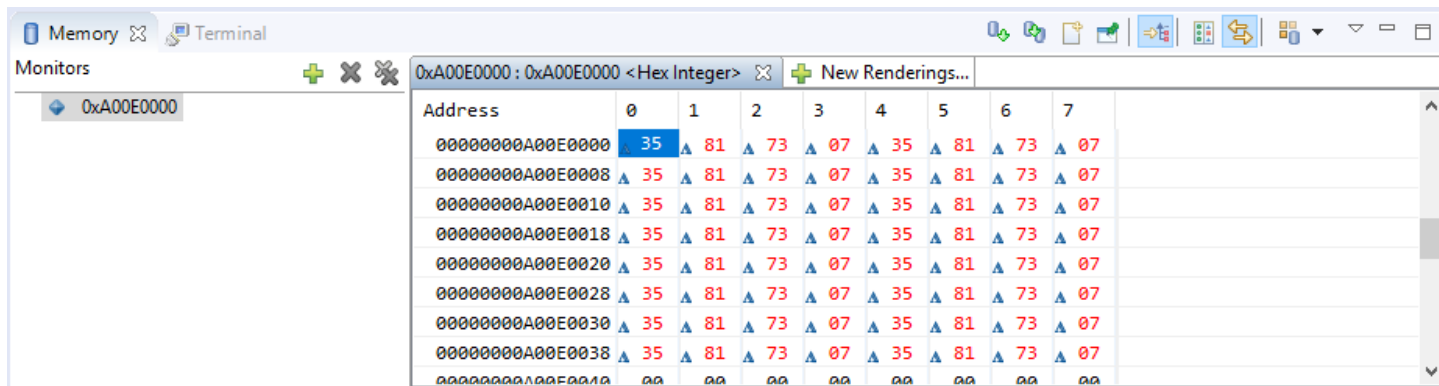
- > **LED1** is turned on to indicate that the PFLASH memory was correctly written
- > **LED2** is turned on to indicate that the DFLASH memory was correctly written



# Run and Test

Check the actual PFLASH memory as an additional test:

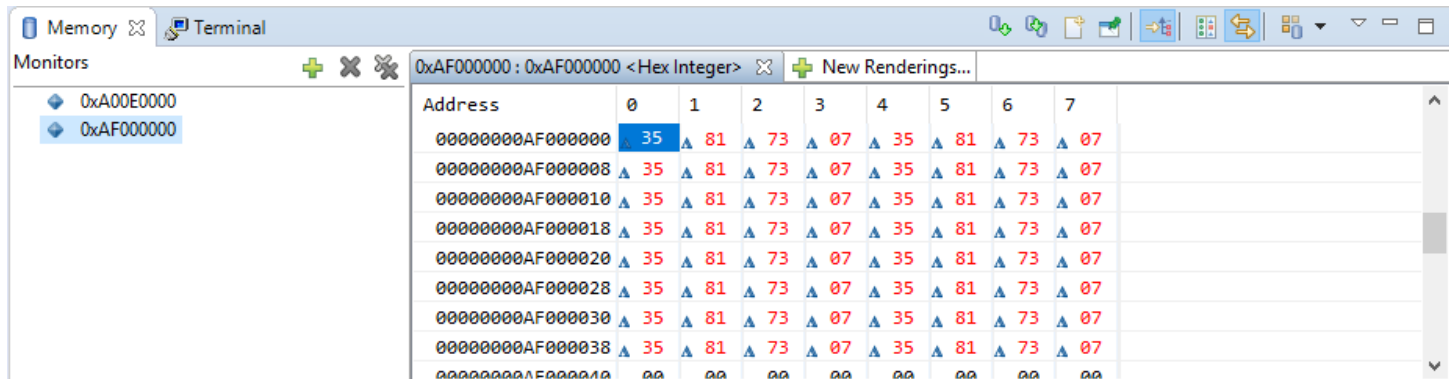
- › Set two breakpoints in the **Flash\_Programming.c** file inside the **writeProgramFlash()** function:
  - After calling the **eraseFlash()** function
  - After calling the **writeFlash()** function
- › Run the debugger
- › In the memory view, add the address 0xA00E0000
- › Resume the debugger and check that when it stops at the first breakpoint, the PFLASH is erased (the memory viewer will show either 0s or 0xEEEEEEEEEE because it cannot read the erased memory)
- › Resume the debugger again and check that when it stops at the second breakpoint, the PFLASH is correctly programmed



# Run and Test

Check the actual DFLASH memory as an additional test:

- › Set two breakpoints in the **Flash\_Programming.c** file inside the **writeDataFlash()** function:
  - After calling the **IfxFlash\_eraseMultipleSectors()** function
  - After the for loop for writing the DFLASH
- › Run the debugger
- › In the memory view, add the address 0xAF000000
- › Resume the debugger and check that when it stops at the first breakpoint, the DFLASH is erased (the memory viewer will show either 0s or 0xEEEEEEEE because it cannot read the erased memory)
- › Resume the debugger again and check that when it stops at the second breakpoint, the DFLASH is correctly programmed



# References



- > AURIX™ Development Studio is available online:
- > <https://www.infineon.com/aurixdevelopmentstudio>
- > Use the „*Import...*“ function to get access to more code examples.



- > More code examples can be found on the GIT repository:
- > [https://github.com/Infineon/AURIX\\_code\\_examples](https://github.com/Infineon/AURIX_code_examples)



- > For additional trainings, visit our webpage:
- > <https://www.infineon.com/aurix-expert-training>



- > For questions and support, use the AURIX™ Forum:
- > <https://www.infineonforums.com/forums/13-Aurix-Forum>

## Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2020-06**

**Published by**

**Infineon Technologies AG  
81726 Munich, Germany**

**© 2020 Infineon Technologies AG.  
All Rights Reserved.**

**Do you have a question about this document?**

**Email: [erratum@infineon.com](mailto:erratum@infineon.com)**

**Document reference**

**Flash\_Programming\_1\_KIT\_TC397\_TFT**

## IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office ([www.infineon.com](http://www.infineon.com)).

## WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.