Flash_Programming_1 for KIT_AURIX_TC297_TFT

Flash programming

AURIX™ TC2xx Microcontroller Training
V1.0.0

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 Program Memory Unit (PMU) controls the Flash memory

- The AURIX™ TC29x device features one PMU with the following configuration:
  - 4 Program **Flash Banks** (PFx)
  - 2 Data **Flash Banks** (DFx)

- In AURIX™ TC29x, each PFx is divided into 4 **Physical Sub-Sectors** and each Physical Sub-Sector is divided into **Logical Sectors** (refer to the User Manual for information about the Physical and Logical Sector dimensions)

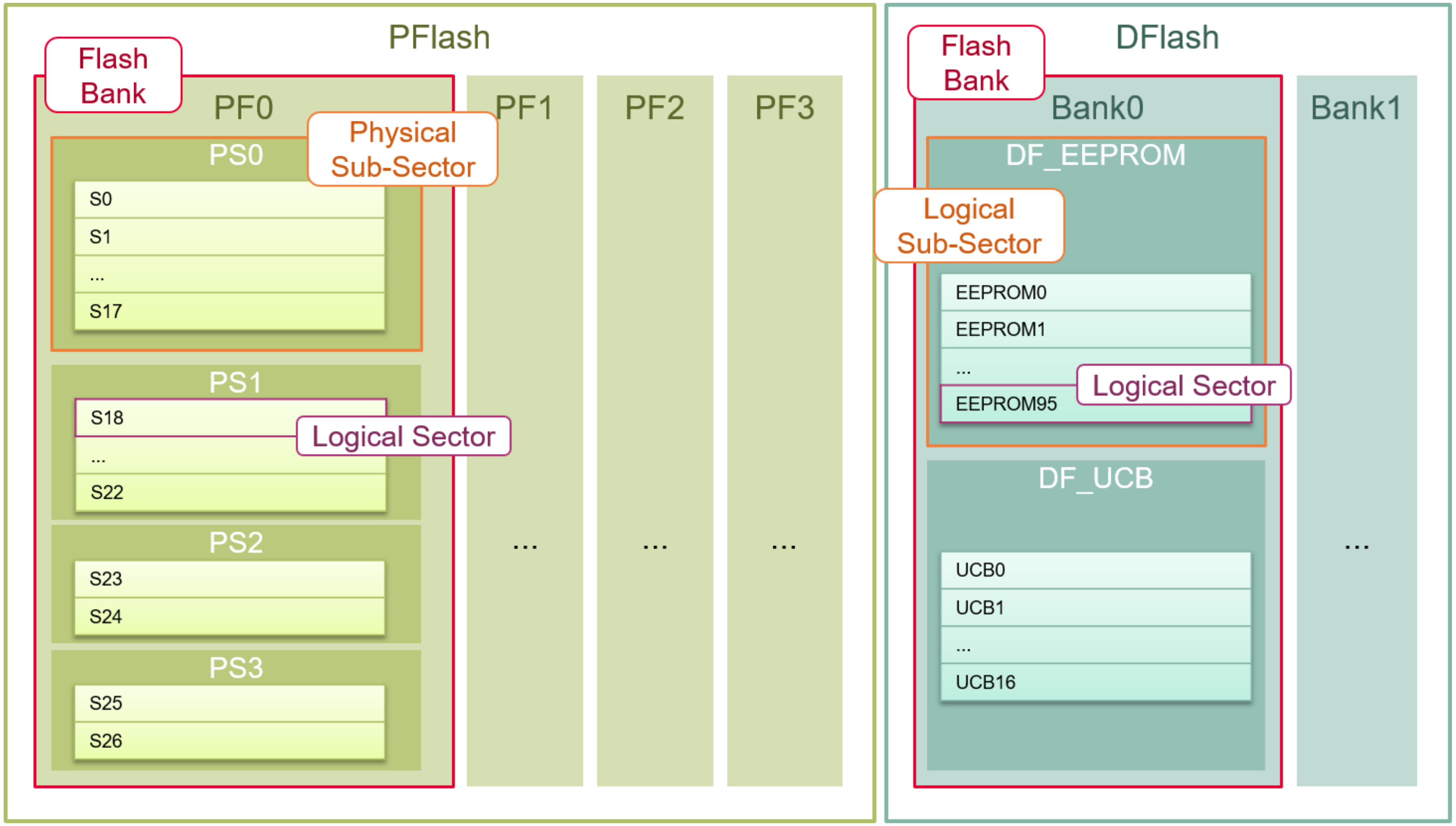
- In AURIX™ TC29x, the Data Flash Bank 0 is divided into 2 **Logical Sub-Sectors**, one divided into 96 **Logical Sectors** and the other divided into 16 Logical Sectors, while the Data Flash Bank 1 is divided into 8 Logical Sectors (refer to the User Manual for information about Logical Sector dimensions)

- 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

Introduction

- AURIX™ TC29x memory



Introduction

› The smallest unit on which an erase operation can be performed is a Logical Sector

› All the flash operations are performed with command sequences

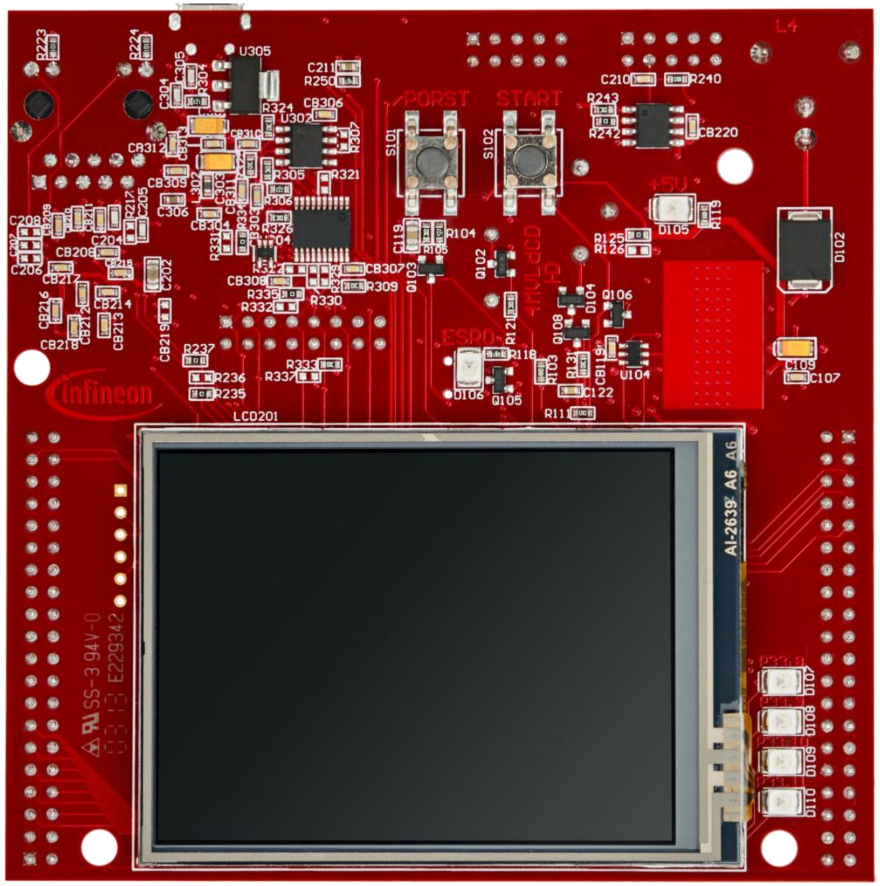
› A minimum sequence of commands for programming the Program Flash memory or the Data Flash memory, is the following:

  - **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_AURIX_TC297_TFT_BC-Step.



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 bit, done through the function ***IfxScuWdt_clearSafetyEndinitInline()***.
Then, the erase command sequence for one or more consecutive Logical Sectors can be executed through the ***IfxFlash_eraseMultipleSectors()*** function, executed from the PSPR by the function pointer ***eraseSectors()***.
Finally, the EndInit bit must be set again through the function ***IfxScuWdt_setSafetyEndinitInline()***.
The function ***IfxFlash_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 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 and it is not needed to copy them into the PSPR.

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.

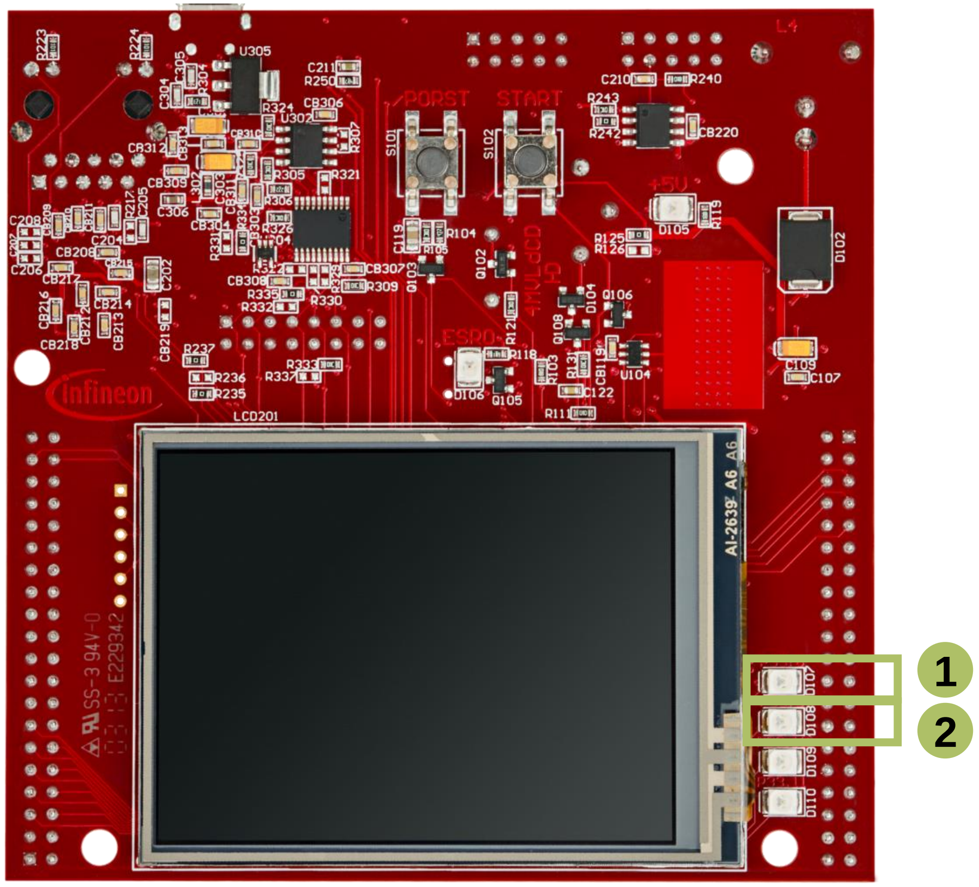
Run and Test

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

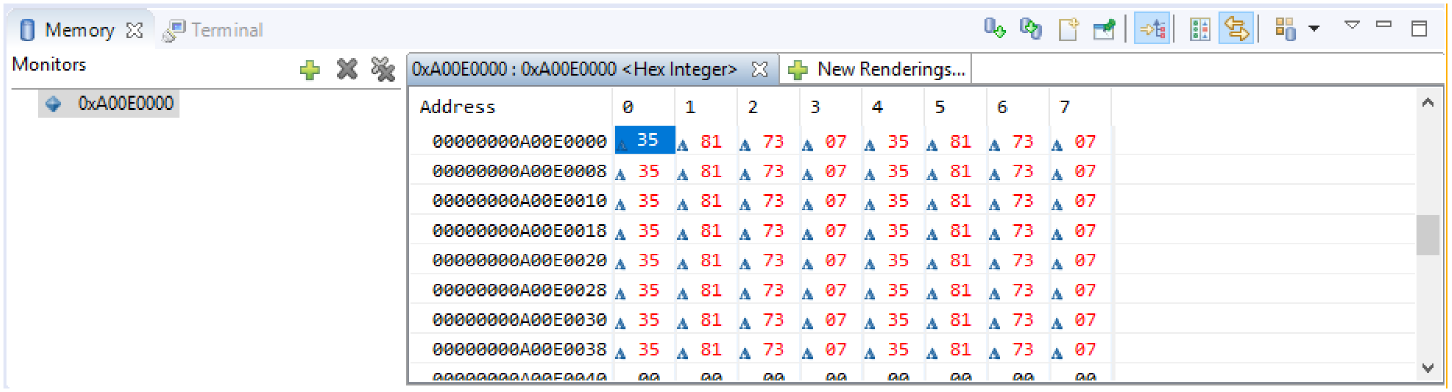
› ***LED2*** is turned on to indicate that the DFLASH memory was correctly written



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 0xEEEEEEEE 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

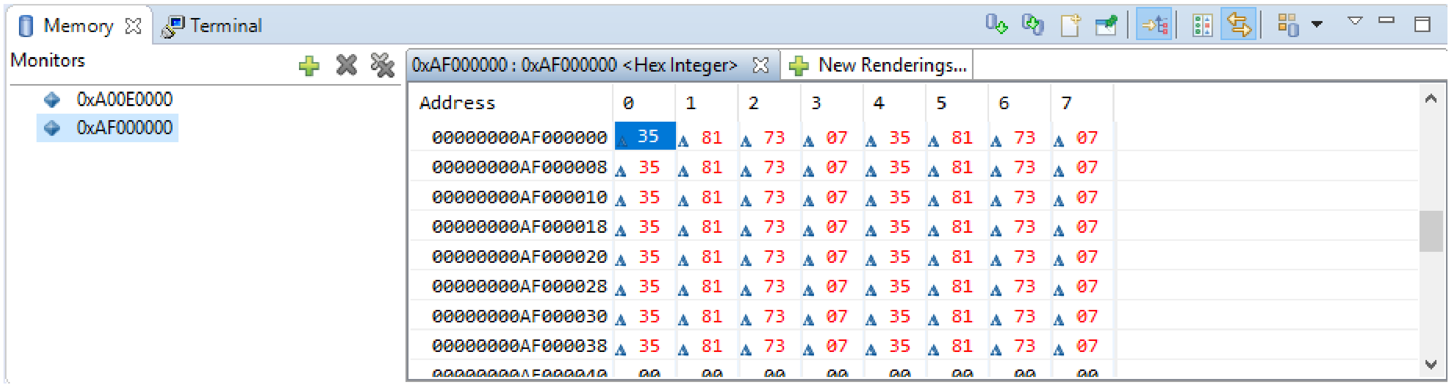


```
Address           0   1   2   3   4   5   6   7
00000000A00E0000  35  81  73  07  35  81  73  07
00000000A00E0008  35  81  73  07  35  81  73  07
00000000A00E0010  35  81  73  07  35  81  73  07
00000000A00E0018  35  81  73  07  35  81  73  07
00000000A00E0020  35  81  73  07  35  81  73  07
00000000A00E0028  35  81  73  07  35  81  73  07
00000000A00E0030  35  81  73  07  35  81  73  07
00000000A00E0038  35  81  73  07  35  81  73  07
00000000A00E0040  00  00  00  00  00  00  00  00
```

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



Address	0	1	2	3	4	5	6	7
00000000AF000000	35	81	73	07	35	81	73	07
00000000AF000008	35	81	73	07	35	81	73	07
00000000AF000010	35	81	73	07	35	81	73	07
00000000AF000018	35	81	73	07	35	81	73	07
00000000AF000020	35	81	73	07	35	81	73	07
00000000AF000028	35	81	73	07	35	81	73	07
00000000AF000030	35	81	73	07	35	81	73	07
00000000AF000038	35	81	73	07	35	81	73	07
00000000AF000040	00	00	00	00	00	00	00	00

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

- 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

Edition 2020-02
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
Flash_Programming_1_KIT_TC297_TFT