SPC58xHxx Firmware Over The Air update with hardware flash A/B context swapping

Introduction

The SPC58NHxx is a member of SPC58x family devices that implements a powerful set of peripherals and features to implement an efficient, reliable, fast and secure Firmware Over The Air update (FOTA).

This application note introduces some basic concepts of Firmware Over The Air update and SPC58xHxx specific features useful to implement FOTA.

Then it details hardware flash A/B context swapping, which is a powerful SPC58xHxx feature to implement OTA and introduces the basic steps needed to update the firmware in SPC58NH92 microcontrollers.

1 FOTA update

1.1 What is FOTA

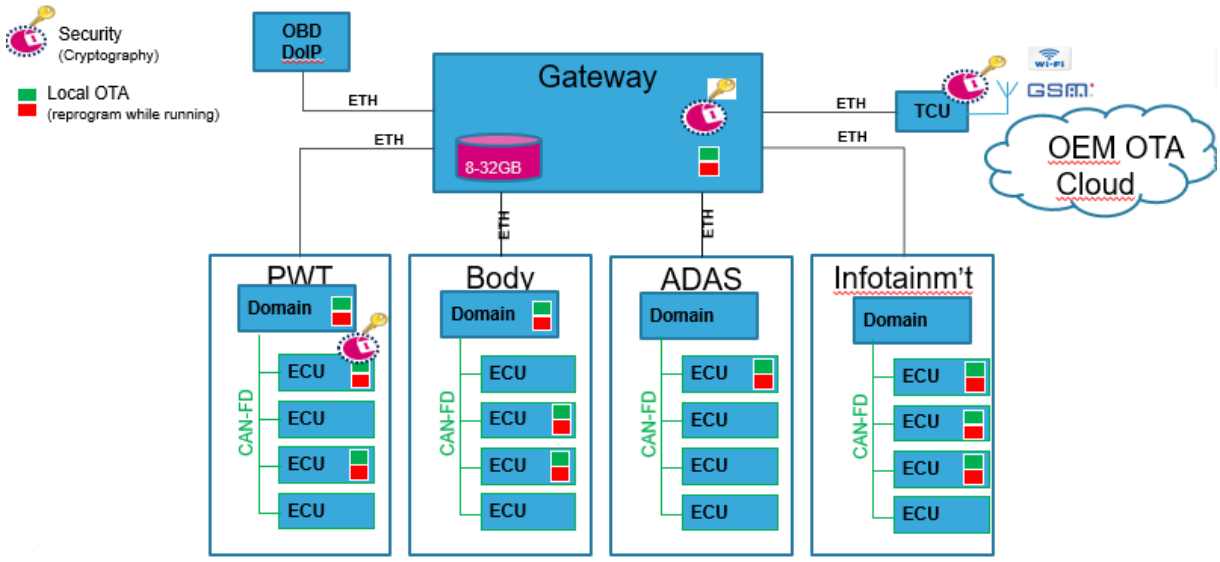
The Firmware Over The Air update, FOTA, is a cost effective, reliable, secure method to update the firmware of the connected cars.

The FOTA can ensure that vehicles are state of art, enabling critical bugs to be patched immediately, adding new features to the vehicle at any time during its lifecycle without requiring a visit to repairing shop saving manufacturer money and customers time.

Directly from the cloud, the updating of the car can be made efficiently, safely and cheaply. The Figure 1 shows a typical FOTA update architecture from the OEM's servers to the ECUs within a vehicle. A secure connection is set up over a cellular network between an individual vehicle and the server. This allows the new updated firmware to be sent securely to the vehicle's Telematics Unit and then on to the OTA Manager (Gateway),

- The Gateway manages the update process for all ECUs within the vehicle. It controls the distribution of firmware updates to ECUs scheduling the firmware update.
- The new firmware will be held in an external storage available in the gateway until the update is required.
- The external flash can also be used to store backup copies of the firmware for other vehicle ECUs which can be called in the case of a major fault in an ECU update, which leaves an ECU without any working firmware.
- Several networks are available to receive the firmware and to update the ECUs in the different domain, PWT, Body, ADAS, Infotainment.

Figure 1. FOTA architecture



1.2 SPC58xHxx features for FOTA and basic concepts

SPC58xHxx (we refer in particularly to SPC58NH92) microcontrollers have specific and powerful features to manage FOTA. Below the list of these features and its usage for OTA:

- 8 Read While Write (RWW) partitions:

  While the embedded Flash memory is performing a “write” (program or erase) to a given partition, it can simultaneously read from any other partition. This is an important feature, as car owners do not tolerate downtime of their vehicles while updates take place. Therefore, updates to vehicle should ideally take place seamlessly and invisibly with background reprogramming. So while one partition is used to run the vehicle application, in background the other partition can be reprogrammed in the meanwhile.

  Background reprogramming can be achieved if the upgrading process is a loop that gets new firmware from external source (like serial link, CAN, UART, FlexRay, ETHERNET) and via DMA/interrupt fills a buffer in RAM and stops periodically to program flash with buffer in RAM. A second core can be dedicated to the firmware update
- Multicore, triple Z4 cores + HSM:

  The main cores can be used to run the vehicle application while a dedicated core could be used to update other flash sectors with the new firmware.
- eMMC and Octal-SPI- Hyperbus:

  Dedicated very fast external memory interfaces like eMMC andOctal-SPI- Hyperbus can be used to store the firmware in the manager ECU (Gateway) so that the firmware can be retrieved when update is required.
- HSM (Hardware Secure Module) full security level:

  A remote update potentially gives to a hacker the possibility to have access to the ECU and take control of the car, but also to have access to it to steal information. HSM and encryption improves protection from hackers attacks
- Ethernet 1G/100M/10 - CAN - FLEXRAY:

  OTA architecture usually uses different network protocols, all are linked to the gateway. The networks Ips and in particular the ETH 1G can ensure a fast firmware update getting the Flash images for the whole set of ECUs subsystem from cloud connected TCU,
- Hardware flash A/B context swapping:

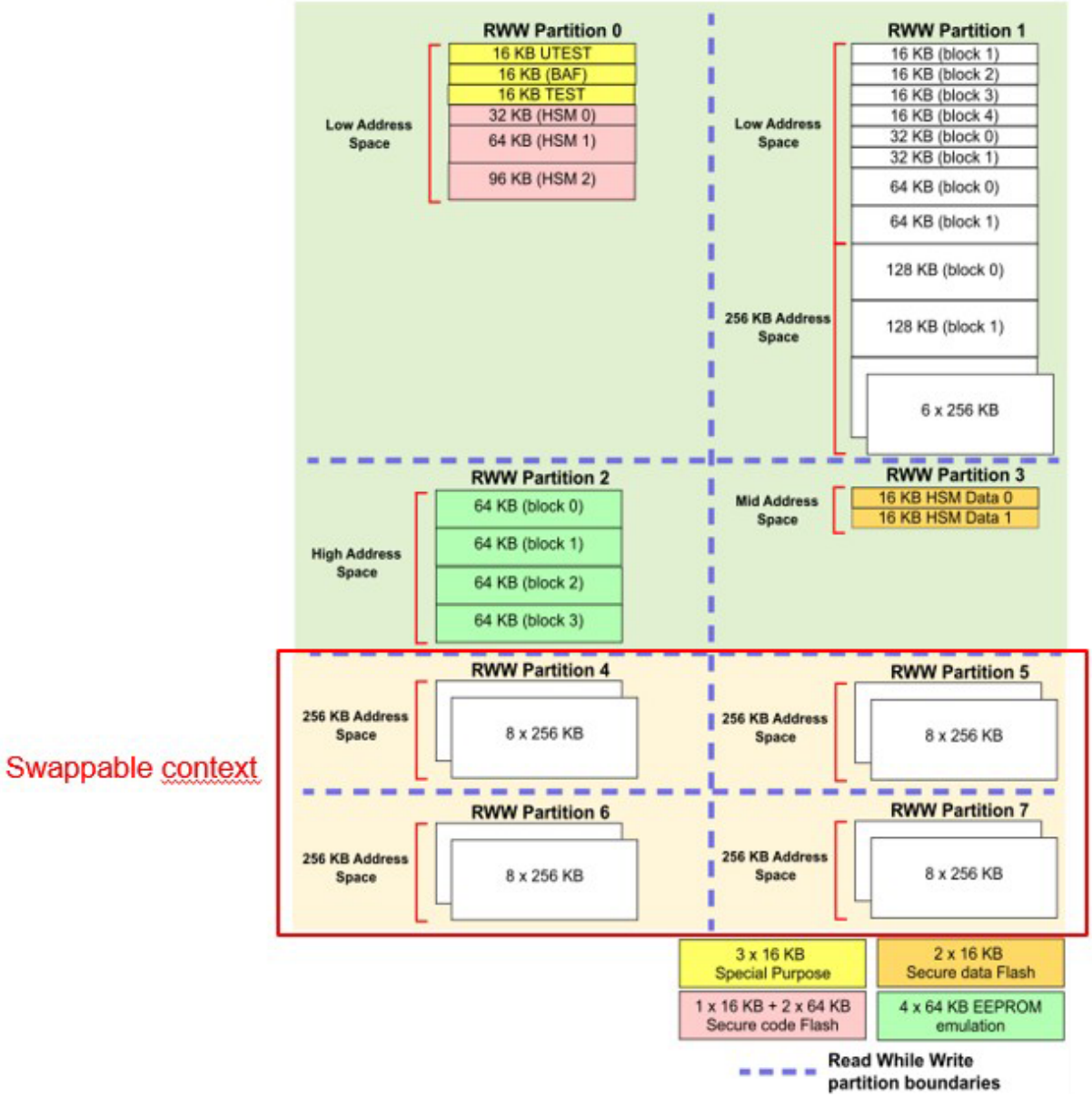
  This is a key feature for FOTA update implementation. It allows to have a flash Sector A that can be swapped with Sector B. This is a powerful feature that allows to have the same firmware image that can be used either for Sector A or Sector B. In other words, if OTA is enabled, the new firmware image is visible always at the same addresses independently where it is physically written in the flash (Sector A or Sector B). In the following pages the Hardware flash A/B context swapping is explained in details.

2 Hardware flash A/B context swapping

2.1 Flash mapping and contexts swapping

The SPC58NH92 Flash module, as shown in the Figure 2, is divided in eight partitions that determine the locations for valid Read While Write (RWW) operations.

Figure 2. SPC58NH92 Flash memory segmentation and Read While Writing partitioning



When the OTA functionality is enabled, the whole flash can be considered divided in three regions/contexts:

- Common region
- Swappable contexts: context A and Context B

Figure 3 shows the common region available at the symbolic address 0, the Context A available at symbolic address 1 and Context B available at symbolic address 2.

Figure 3. **Flash regions**

The A/B context swapping is the feature for which Context A and Context B, even if physically written respectively at address 1 and address 2 of flash, are visible (can be read) at address 1 or address 2 depending on if the swapping is enabled or not.

Note: the swapping of the two contexts is only valid for reading the flash, while for writing the flash, the swap has no effect.

A/B context swapping is summarized in the Table 1:

Table 1. **FOTA's Flash context swapping concept**

	Writing flash		Reading flash	
	Swap = 0	Swap = 1	Swap = 0	Swap = 1
common	0	0	0	0
Context A	1	1	1	2
Context B	2	2	2	1

Writing the flash, independently from swap value, the address of context A is always 1 and address of context B is always 2. Reading the flash , if swap = 0 (swap disabled), Sector A is at address 1 and Sector B is at address 2, if swap = 1 (swap enabled) Context A is at address 2 and Context B is at address 1.

For what discussed above, because of context A/B swap feature, the SW image is visible always at the same addresses independently where it is physically written in the Context A or Context B of the flash. This allows the new firmware image, to be written in the flash, to be independent from sector destination.

Table 2 reports the swappable partitions and their addresses available in SPC58NH92 microcontrollers.

Table 2. **SPC58NH92 OTA contexts mapping**

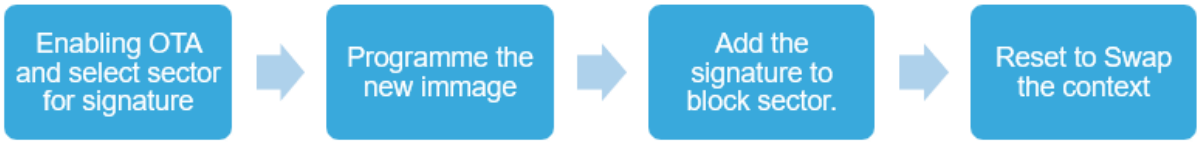
Context	RWW partitions	Physical addresses
A	4-5	0x011C0000-0x013C0000
B	6-7	0x015C0000-0x017C0000

3 OTA steps

In this chapter the basic four steps for FOTA update implementation are described:

1. Enable and configure the flash A/B context swapping for OTA.
2. Program new firmware image.
3. Add the signature to swap the context.
4. Swap the context.

Figure 4. Basic steps for firmware update



The following sections describe each of these steps in detail.

3.1 Enabling OTA and select sector for signature (step 1)

The first step is to enable the OTA and select the OTA flash block for signature(in fact as described later in this document, to swap the context A and B, a signature has to be added in a specific sector of the flash) by means of DCF UTEST2 miscellaneous client.

DCF UTEST2 miscellaneous client and its bitfields to be configured for FOTA are shown in Figure 5:

- bit field 28 : OTA_ENABLE.
  - Set to 1, enable the OTA feature
- bit field 26-27: OTA_SIGNATURE.
  - Select one of the four blocks for signature. The 4 sectors for signatures are @ addresses:
    - 32 KB block 0 0x00FD0000
    - 32 KB block 1 0x00FD8000
    - 64 KB block 0 0x00FE0000
    - 64 KB block 1 0x00FF0000

Figure 5. UTEST2 miscellaneous DCF Client

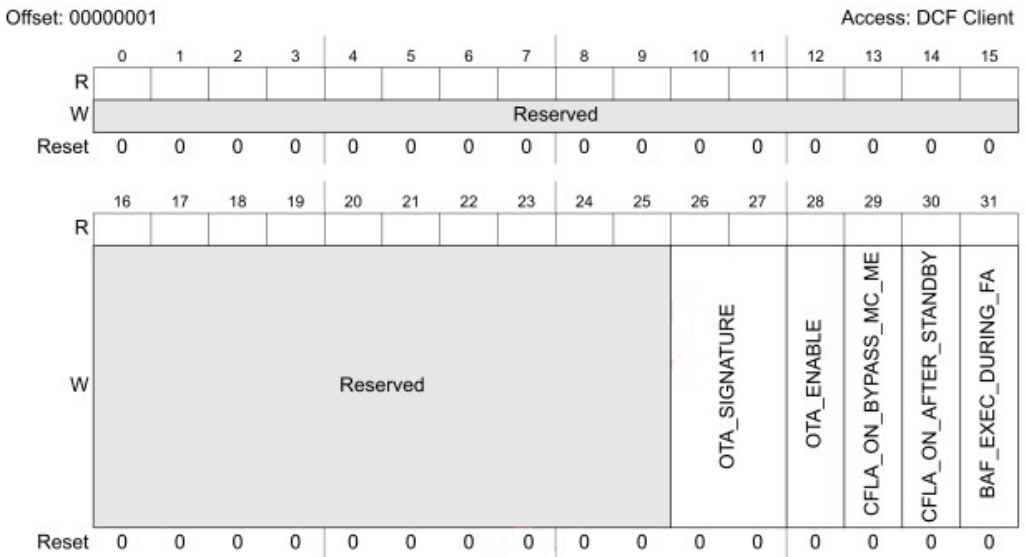


Table 3. UTEST2 miscellaneous DCF client field descriptions

Field	Description
26:27[1] OTA_SIGNATURE	Determines the Flash block where is written the signature indicating the active context. 00 32 KB block 0 (Base address 00FD0000) 01 32 KB block 1 (Base address 0x00FD8000) 10 64 KB block 0 (Base address 0x00FE0000) 11 64 KB block 1 (Base address 0x00FF0000)
28[1] OTA_ENABLE	Determines whether the OTA feature is available.
29 CFLA_ON_BYPASS_MC_ME	1 - Flash memory cannot be swithed-off by MC_ME.CFLAON bits in RUNx modes. 0 - Flash memory can be swithed-off by MC_ME.CFLAON bits in RUNx modes.
28 CFLA_ON_AFTER_STANBY	1 - Flash memory will be turn-on after the STBY exit. 0 - Flash memory can be switched (on/off) by MC_ME.CFLAON bits.
28 BAF_EXEC_DURING_FA	1 - BAF does not stop into an endless loop when LifeCycle = FA 0 - BAF stop into an endless loop when LifeCycle = FA

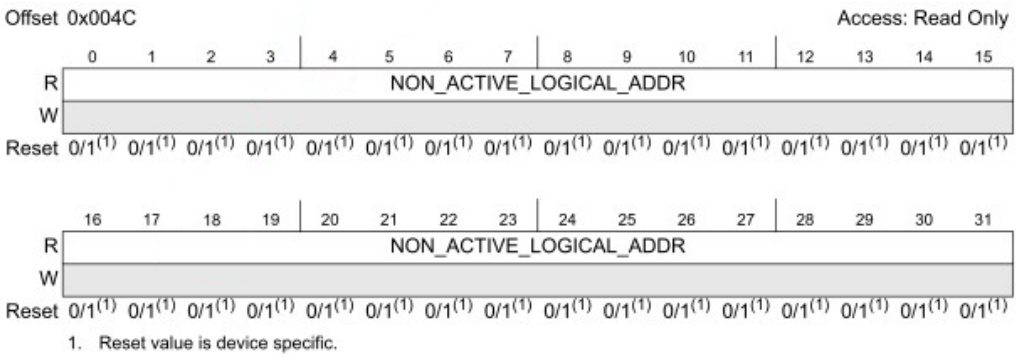
1. *When OTA feature is enabled, it cannot be disabled. Actually, the UTEST2 miscellaneous DCF register is a one time programmable register and so all its bitfields, OTA_ENABLE and OTA_SIGNATURE included, if set, cannot be changed.*

3.2 Programming the new image (step 2)

The second step is to program the new firmware image in the no-active context. While the application is running in the active context, supposing, for example, it is @ addresses 0x011C0000, the new firmware image has to be programmed in the no-active context, that is @ addresses 0x015C0000.

The new firmware image has to be written in the real address of the no active context, in fact for writing the swap has no effect on address. The address of no-active context is readable in the Non Active Context Logical Address register (NACLA) in the SSCM peripheral (see Figure 6).

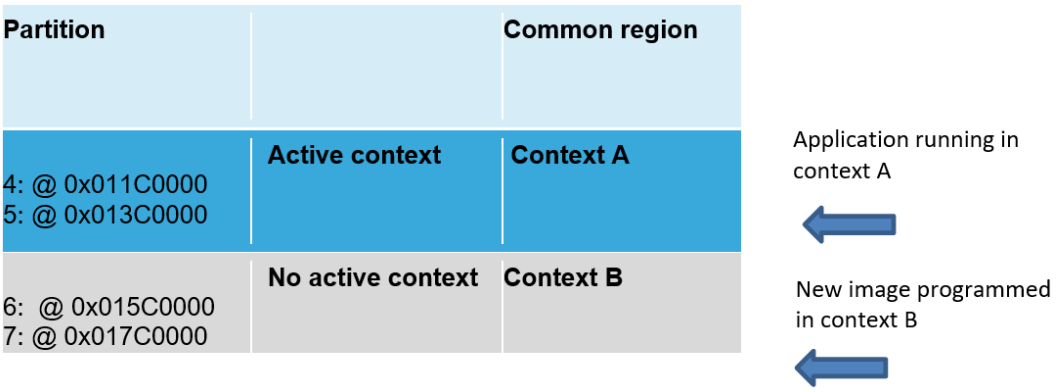
Figure 6. No Active Context Logical Address register (NACLA)



1. Reset value is device specific.

The new firmware image is always compiled considering the flash address in active context that in the example is @ address 0x011C0000. In fact, the firmware is read always at same address.

Figure 7. **Programming non active context B**



3.3 Add a signature to selected block sector (step 3)

The step 3 consists in adding a signature in one of the 4 blocks selected by DCF UTEST2 miscellaneous client. With step 3 the microcontroller is ready for the swapping but only with step 4 the swapping takes place.

The 4 selectable sectors for signatures are @ addresses:

- 32 KB block 0 0x00FD0000
- 32 KB block 1 0x00FD8000
- 64 KB block 0 0x00FE0000
- 64 KB block 1 0x00FF0000

The data programmed in the signature are irrelevant, any value can be written provided that is respected what follow:

- The signature is written in slot of 16 bytes starting from the beginning of the sector.
- The information shall be written in sequential order.
- Last slot of signature must not be programmed.

For each added signature we have a context swapping, from Context A to Context B and viceversa. For example:

- Even slot programmed, context A @ ADDRESS 0x11C0000 is made active, NACLA = 0X015C0000 (so the new firmware to be written shall start from address 0x015C0000).
- Odd slot programmed, context B @ address 0x015C0000 is made active, NACLA = 0X011C0000 (so the firmware to be written starting at address 0x011C0000).

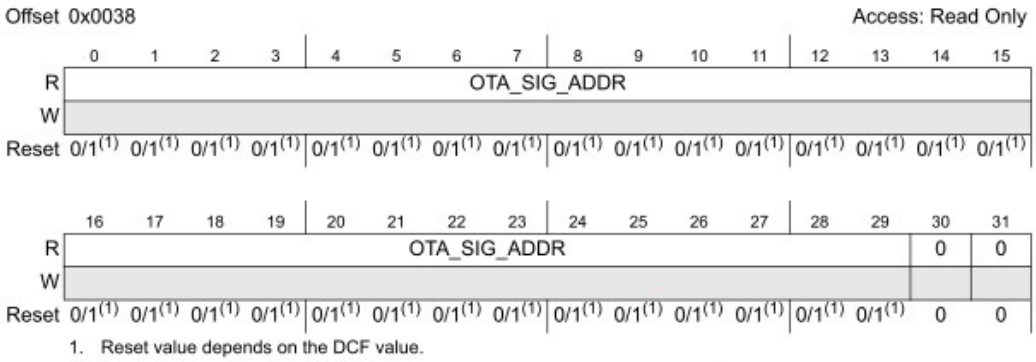
The software can identify the first empty slot address for the signature programming by means of the OTA Signature Address Location register (OSAL).

The contents of this “only read” register has to be manipulated to get the correct address using the following formula:

$$Empty\_Slot\_Of\_SignatureBlock = OSAL[OTA\_SIG\_ADD] \times 4 + 0x00FC0000 \quad (1)$$

Note: OSAL[OTA_ISG_ADDR] is equal to 0x3FFFFFFF when the selected OTA block is full.

Figure 8. OTA Signature Address Location register (OSAL)



1. Reset value depends on the DCF value.

3.4 Swapping the context (step 4)

After enabling the OTA and selecting the sector for signature in step 1, programming the new firmware in the no-active context in step 2, adding a new signature in step 3, the last step 4 is to swap the context.

Step 4 consists of a destructive reset to enable the context swapping. After a destructive reset the new firmware is available in the active context.

A long functional reset shall not change the OTA context, which will remains fixed to the value latched during the last destructive reset. A short functional reset has no relevance from OTA point of view: nothing shall be read from the Flash and the latched out come (that is: which OTA is active) shall be preserved.

Appendix A Other information

A.1 Reference documents

Table 4. **Reference documents**

Doc Name	ID	Title
RM0452	031241	SPC58 H Line - 32 bit Power Architecture automotive MCU Triple z4 cores 200 MHz, 10 MBytes Flash, HSM, ASIL-D
DS12304	031027	SPC58 H Line - 32 bit Power Architecture automotive MCU Triple z4 cores 200 MHz, 10 MBytes Flash, HSM, ASIL-D

Revision history

Table 5. **Document revision history**

Date	Version	Changes
01-Dec-2020	1	Initial release.

Contents

List of tables

List of figures