AN12523

Firmware versus Library model applications

Rev. 2 — 10 December 2019 Application note

1 Introduction

This document explains the difference between Firmware Model Applications and Library Model Applications and how to migrate from the firmware model to the library model.

2 Memory map of the FXTH87/87E and NTM88 devices shipped by NXP

NXP programs the FXTH87/87E devices at production with an embedded firmware containing:

- Firmware functions;[1]
- A jump table used by the user application to call the firmware functions in the application code;
- Firmware interrupt vectors;
- Trim section containing trim coefficients and constant bytes like the Unique ID.

In FXTH devices, the firmware version flashed by NXP at production is stored in byte CODE 0. Use the function TPMS_READ_ID to return the value[2].

NTM88 devices are programmed by NXP at production only with trim coefficients. After an application is programmed in the NTM88, the firmware library version is stored in byte CODE F. Use the function TPMS_READ_ID to return the value. NXP does not program this byte at production so CODE F is empty when NTM88 devices are shipped.

Devices shipped by NXP have the following memory map:

1 Refer to the Firmware User Guide for the list and description of the functions.
2 Refer to the Firmware User Guide.

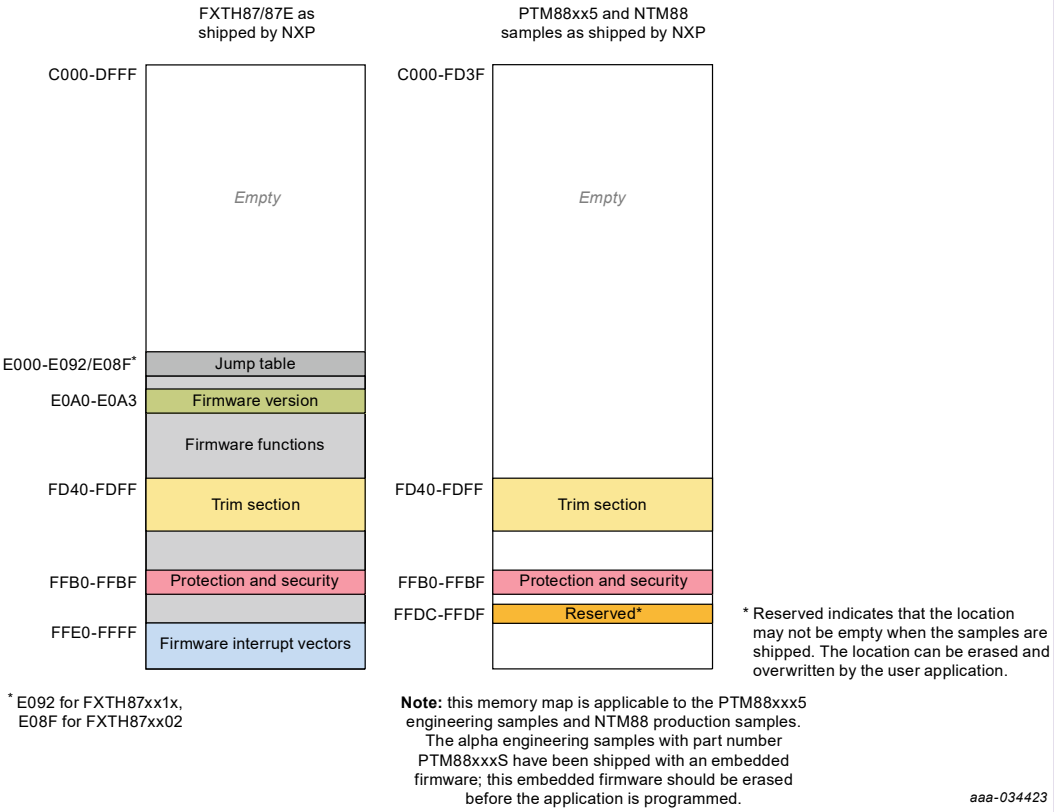


Figure 1. Memory map of the FXTH87/87E and NTM88 devices as shipped by NXP

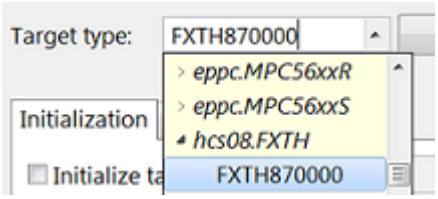
With FXTH87/87E, the user may keep or erase the embedded firmware. Applications using the embedded firmware are named “firmware-based” applications. Applications erasing the embedded firmware are named “library-based” applications.

With NTM88, NXP does not program firmware functions, therefore, "firmware-based" applications cannot be used. NTM88 requires library-based applications.

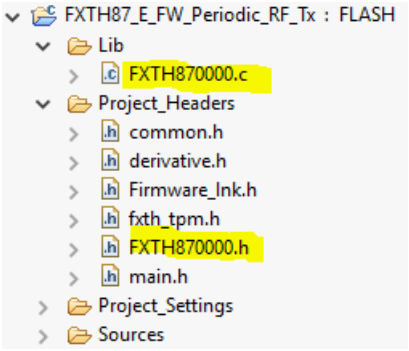
3 Firmware-based applications for FXTH87/87E

3.1 CodeWarrior target

Selecting the target MCU is required when creating a project or configuring a debug session. When installing the latest CodeWarrior versions, FXTH87/87E firmware-based applications use the default CodeWarrior target “FXTH870000” shown here:



In an FXTH87/87E application using the FXTH870000 target, the files *FXTH870000.c* and *FXTH870000.h* contain register descriptions. These files are automatically copied to the *Lib* and *Project_Headers* folders shown here:



The CodeWarrior target "FXTH870000" provides access only to user flash from C000h to DFFFh. The firmware flash from E000h to FFFFh contains protected, embedded firmware which cannot be erased or overwritten by CodeWarrior.

3.2 Memory map

FXTH87/87E firmware-based applications are programmed in the user application flash from C000h to DFFFh. In that situation, the application does not have access to the firmware flash from E000h to FFFFh.

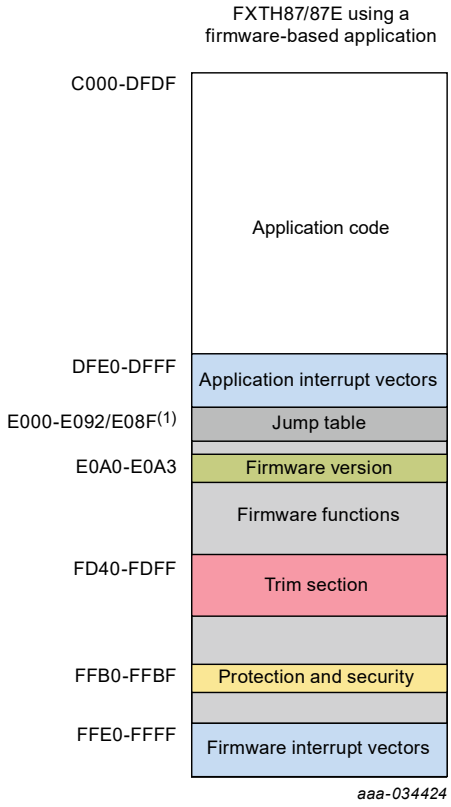


Figure 2. Memory map of the FXTH87/87E using a firmware-based application

3.3 Calling the firmware functions

The application code calls the firmware functions via the jump table. All of the firmware functions reside in the firmware flash. During execution, when a firmware function is called, the program jumps in the firmware flash to execute the firmware function. Upon completion of the function, the program returns to the application flash.

Calling a firmware function from the jump table is demonstrated in the demo projects provided by NXP.

3.4 Interrupt vectors

When an interrupt occurs, the program

1. Enters the firmware interrupt vector programmed by NXP,
2. Updates the TPMS_INTERRUPT_FLAG when applicable[3] and
3. Jumps into the user interrupt vector.[4]

3 Refer to the Firmware User Guide.

4 Under certain conditions for the LF interrupt vector. See *Note on LF interrupt vector* in this section for an explanation.

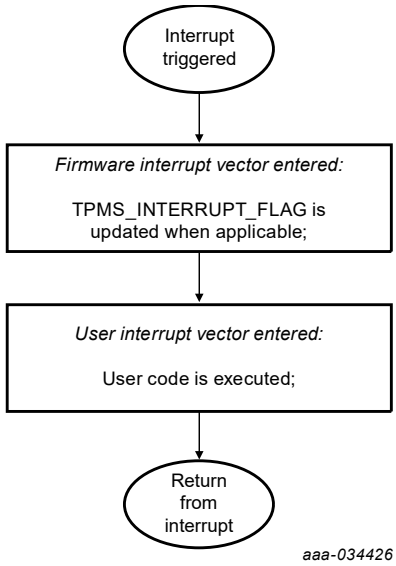


Figure 3. Program flow when an interrupt is triggered

The array of user interrupt vectors must be defined in the application project. Refer to the Reference Manual for information on the addresses for user interrupt vectors.

```
void(* const USER_INTERRUPT_TABLE[]) () @ 0xDFE0 =
{
  USER_15_INTERRUPT,
  USER_14_INTERRUPT,
  USER_13_INTERRUPT,
  USER_12_INTERRUPT,
  USER_11_INTERRUPT,
  USER_10_INTERRUPT,
  USER_9_INTERRUPT,
  USER_8_INTERRUPT,
  USER_7_INTERRUPT,
  USER_6_INTERRUPT,
  USER_5_INTERRUPT,
  USER_4_INTERRUPT,
  USER_3_INTERRUPT,
  USER_2_INTERRUPT,
  USER_1_INTERRUPT,
  main
};
```

Note on LF interrupt vector

In the LF firmware interrupt vector a different sequence is executed: the program jumps in the user interrupt vector only if user application code was being executed when the LF interrupt occurred. If a firmware function was under execution when the interrupt occurred, then the LF user interrupt vector is not accessed.

A different sequence is executed to ensure the correct reception of all data bytes for LF working in data mode. In data mode, when the first *data ready* interrupt[5] is triggered, the user application must call the firmware function TPMS_LF_READ_DATA[6]. TPMS_LF_READ_DATA waits and stores all data bytes received. As the baud rate of

5 Refer to the Reference Manual.
6 Refer to the Firmware User Guide.

the LF data is 3906 bit/s, one new byte is received every 256 µs. At each new data byte received, a *data ready* interrupt is triggered so the interrupt vector is accessed. This situation implies that, if the LF interrupt vector code takes more than 256 µs to execute, LF data bytes are missed. The LF interrupt will remain under execution at the arrival of the next data byte and the function TPMS_LF_READ_DATA will not store all the data bytes.

To prevent this problem, when TPMS_LF_READ_DATA is under execution, the program does not jump in the LF user interrupt vector when the user code takes longer to execute.

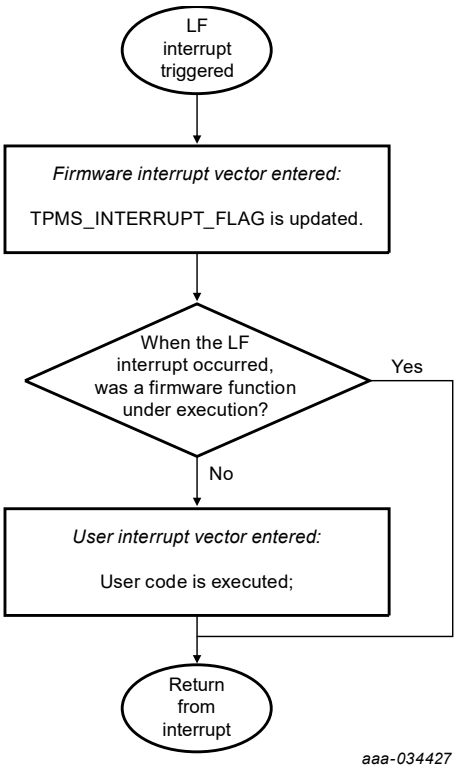


Figure 4. Program flow when an LF interrupt is triggered

Note on the reset vector

Upon reset, the reset vector in the firmware flash area is accessed. The following actions are performed in this interrupt vector:

- LF registers LFCTRLB to LFCTRLE are configured with NXP recommended values;
- The stack pointer is set to address 28Fh (last address of the RAM);
- Jump to user interrupt vector at address DFFEh (it should point to *main*).

3.5 Protection and security registers

Registers configuring MCU protection and security are not directly accessible. The registers are located at addresses FFBDh to FFBFh within the firmware flash[7]. To configure the protection, use the use the firmware function TPMS_FLASH_PROTECTION[8], provided by NXP.

Security configuration is not supported.

7 Refer to the Reference Manual.
8 Refer to the Firmware User Guide.

3.6 TPMS_FLASH functions

Firmware functions TPMS_FLASH_WRITE[9] and TPMS_FLASH_ERASE[10] write and erase flash bytes. However, write and erase operations are permitted only in the user flash section from C000h to DFFFh. These firmware functions do not operate in the firmware flash section from E000h to FFFFh.

The firmware function TPMS_FLASH_CHECK[11] checks the integrity of the embedded firmware. This function calculates the CRC16 checksum for the NXP firmware area from E000h to FFADh. The function compares the checksum with a pre-calculated, stored value and reports whether the two values match or not.

4 Library-based applications for FXTH87/87E and NTM88

4.1 Compatible part numbers

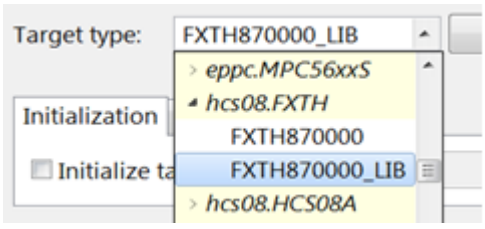
Only three FXTH87/87E part numbers, originally programmed by NXP with an embedded firmware versions 2Ah or 24h, can use library-model applications. They include FXTH87xx02 (2Ah), FXTH87xx11 (24h), and FXTH87xx12 (24h).

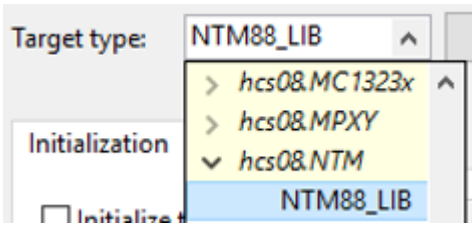
FXTH devices with part number FXTH87xx22 cannot use library-model applications because the trim is flashed in a section not protected by the FXTH870000_LIB target. Refer to Section 4.2.

All NTM88 part numbers can use library-model applications.

4.2 CodeWarrior target

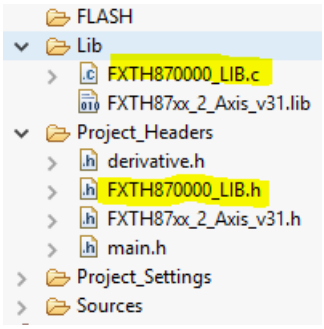
FXTH87/87E library-based application projects use the CodeWarrior target “FXTH870000_LIB,” available after installing the FXTH870000_LIB CodeWarrior patch. NTM88 library-based application projects use the CodeWarrior target “NTM88_LIB,” available after installing the NTM88_LIB CodeWarrior patch. These targets are not available as defaults in CodeWarrior. When creating projects or configuring debug sessions, the target MCU must be selected.

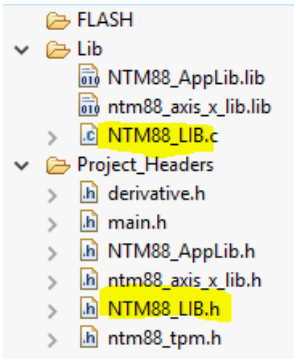




In projects using the FXTH870000_LIB target, the files *FXTH870000_LIB.c* and *FXTH870000_LIB.h* are automatically copied in the *Lib* and *Project_Headers* folders. In projects using the NTM88_LIB target, the files *NTM88_LIB.c* and *NTM88_LIB.h* are automatically copied in the *Lib* and *Project_Headers* folders. These files contain the register descriptions.

9 Refer to the Firmware User Guide.
10 Refer to the Firmware User Guide.
11 Refer to the Firmware User Guide.





These targets give access to the whole flash except for the trim page from FC00h to FDFFh which is protected from write and erase operations. There is no distinction between user and firmware flash, the application has access to the whole flash except the trim section as mentioned.

When a library-based project is programmed, the embedded firmware programmed by NXP at production is erased.

After the embedded firmware has been erased in a device, firmware-based applications cannot be programmed in that device since the embedded firmware is no longer present.

In the FXTH, NXP programs byte CODE 0 with the embedded firmware version. With library-based applications, the embedded firmware version at CODE 0 is erased and overwritten with the firmware library version. Before programming a library-based project, use TPMS_READ_ID to return the version number from byte CODE 0.

In the NTM88, CODE F stores the firmware library version after the library-based application has been programmed. This byte is originally empty when the devices are shipped by NXP.

4.3 Memory map

Library-based applications are programmed in flash from C000h to FBFFh and from FE00h to FFFFh.

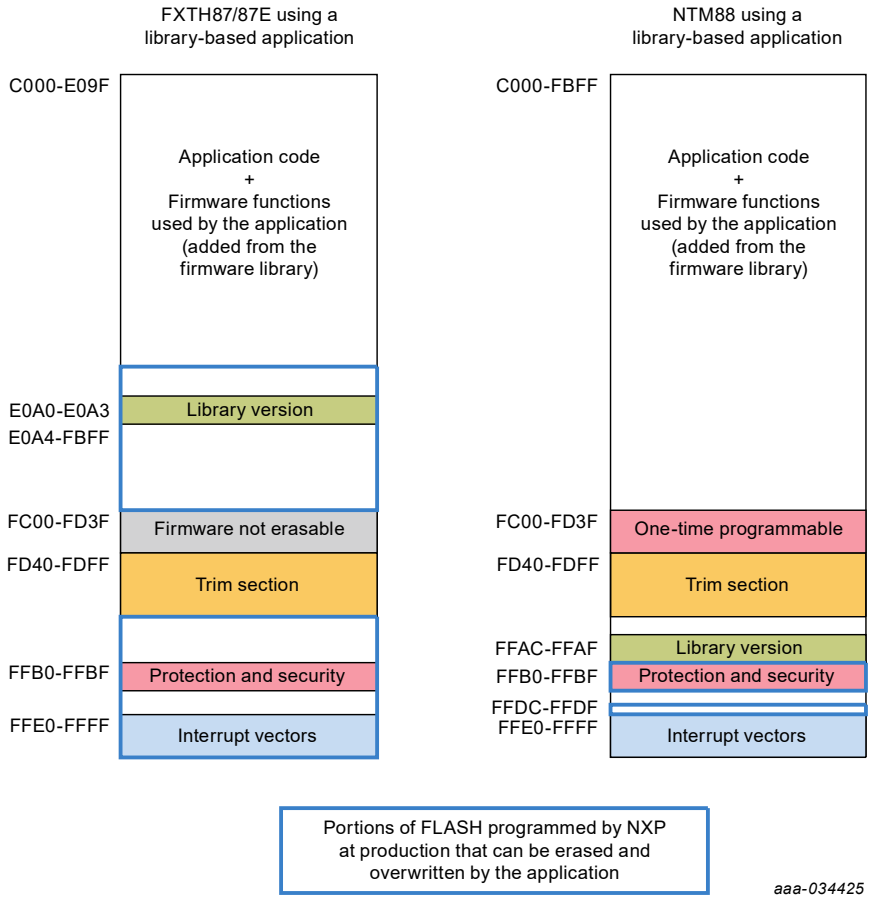


Figure 5. Memory map of the FXTH87/87E and NTM88 using a library-based application

In the FXTH87/87E, a section in the trim page, originally programmed with firmware, is named “Firmware not erasable”. The section contains embedded firmware programmed by NXP at production that cannot be erased. The section of firmware cannot be erased because flash can only be erased 512 bytes by 512 bytes. Erasing the FC00h to FD3F section would also erase the trim coefficients section from FD40 to FDFFh. Erasing the trim coefficients section renders the sensors non-functional.

With NTM88, the section from FC00h to FD3Fh, named “one-time programmable,” is empty when the devices are shipped. Customers may store data in the one-time programmable section, however, it cannot be erased. As previously mentioned, flash can only be erased 512 bytes by 512 bytes. Erasing the one-time programmable section would also erase the trim coefficients from FD40h to FDFFh, rendering the sensors non-functional.

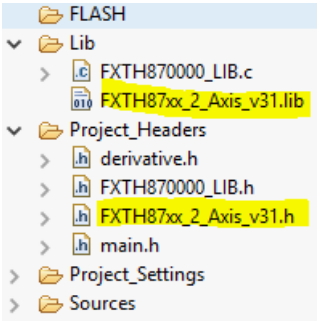
4.4 Calling the firmware functions

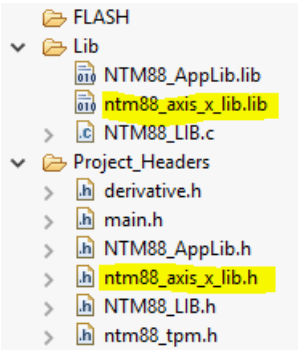
In library-based applications, firmware functions are added from a firmware library provided by NXP. The library is included in the project and the application calls the library functions via the library header file. The firmware library functions used by the application are copied in the application flash.

There is a key difference between the firmware and library models. In the library model, only the firmware functions used by the application are placed in memory. However, in the firmware model, *all* firmware functions are programmed in the embedded firmware flash.

The library model optimizes the flash memory size.

In the CodeWarior project, the library file is added in the *Lib* folder and the library header file is added in the *Project_Headers* folder. The library file and library header file are different for one- and two-axis devices.





4.5 Interrupt vectors

In library-based applications, there is no distinction between firmware and user interrupt vectors. The interrupt vectors are found in a single array, defined by the user in the application.

The interrupt vectors must be placed at the addresses defined in the MCU, starting at FFE0h. From an application point of view, the interrupt vectors fall into three categories:

- Interrupt vectors not containing user application code. This is to ensure correct operation of the firmware functions. In that case, a function defined in the firmware library is used as interrupt vector, so the application should not define the function.
- Interrupt vectors in which the TPMS_INTERRUPT_FLAG must be updated. The interrupt vectors must include code to update the flag in addition to the user code.
  In the firmware library for the FXTH87/87E, handler functions updating the TPMS_INTERRUPT_FLAG are provided. The user must call these handler functions in the interrupt vector and add user code.
  For NTM88, no handler function is provided. The user must add the code to update the TPMS_INTERRUPT_FLAG. The demo projects provide examples.
  User code must address the interrupt acknowledge to clear the register flag.
- Interrupt vectors containing user code only. The user code must address the interrupt acknowledge to clear the register flag.

Table 1 summarizes how each interrupt vector should be defined.

Table 1. Interrupt vector definition

Block generating the interrupt	Vector address	Will the application define the function?	Content of the function
KBI	FFE0h:FFE1h	Yes	87/87E: Call to *vfnTPMSKBIFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
87/87E: *Reserved*	FFE2h:FFE3h	87/87E: No[1]	87/87E: *N/A*
88: FRC		88: Yes	88: user code only
Reserved	FFE4h:FFE5h	87/87E: No[1]	87/87E: *N/A*
		88: Yes	88: Define an empty function (this interrupt vector will never be accessed)
RTI	FFE6h:FFE7h	Yes	87/87E: Call to *vfnTPMSRTIFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
LFR	FFE8h:FFE9h	Yes	87/87E: Call to *vfnTPMSLFRFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
ADC	FFEAh:FFEBh	87/87E: No[2]	87/87E: *N/A*
		88: Yes	88: Update TPMS_INTERRUPT_FLAG + user code
RFM	FFECh:FFEDh	Yes	User code only
SMI	FFEEh:FFEFh	No[3]	*N/A*
TPM1 overflow	FFF0h:FFF1h	Yes	87/87E: Call to *vfnTPMSTPMFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
TPM1 channel 1	FFF2h:FFF3h	Yes	User code only
TPM1 channel 0	FFF4h:FFF5h	Yes	User code only
PWU	FFF6h:FFF7h	Yes	87/87E: Call to *vfnTPMSPWUFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
LVD	FFF8h:FFF9h	Yes	87/87E: Call to *vfnTPMSLVDFslIntHandler()* + user code
			88: Update TPMS_INTERRUPT_FLAG + user code
87/87E: *Reserved*	FFFAh:FFFBh	87/87E: No[1]	87/87E: *N/A*
88: Irq		88: Yes	88: user code only
SWI	FFFCh:FFFDh	Yes	User code only
Reset	FFFEh:FFFFh	87/87E: No[1]	*N/A*
		88: No[4]	

[1] Use *_FXTH87x6LibStartup* defined in the library.
[2] Use *vfnTPMSADCIsr* defined in the library.
[3] Use *vfnTPMSSMIIsr* defined in the library.
[4] Use _Startup defined in the CodeWarrior file start08.c.

In the application project, the array can be defined as shown, with FFE0h as the start address.

For FXTH87/87E:

```
volatile void (* const IsrVector []) (void) @
 INTERRUPT_VECTORS_START_ADDRESS =
  {
    vfnKBIIsrUser,           /* Defined by the user */
    _FXTH87x6LibStartup,     /* Defined in the library */
    _FXTH87x6LibStartup,     /* Defined in the library */
    vfnRTIIsrUser,           /* Defined by the user */
    vfnLFRIsrUser,           /* Defined by the user */
    vfnTPMSADCIsr,           /* Defined by the library */
    vfnRFMIsrUser,           /* Defined by the user */
    vfnTPMSSMIIsr,           /* Defined by the library */
    vfnTPMIsrUser,           /* Defined by the user */
    vfnTPMCH1IsrUser,        /* Defined by the user */
    vfnTPMCH0IsrUser,        /* Defined by the user */
    vfnPWUIsrUser,           /* Defined by the user */
    vfnLVDIsrUser,           /* Defined by the user */
    vfnIRQIsrUser,           /* Defined by the user */
    vfnSWIIsrUser,           /* Defined by the user */
    //_FXTH87X6LibStartup    /*Defined in the library ; See prm
 file: VECTOR 0 _FXTH87x6LibStartup */
  };
```

For NTM88:

```
void (* const IsrVector []) (void) @ (0xFFE0u =
  {
    vfnKBIIsr,               /* IRQ15 KBI Interrupt */
    vfnFRCIsr,               /* IRQ14 FRC Interrupt */
    vfnUnusedIsr,            /* IRQ13 Reserved */
    vfnRTIIsr,               /* IRQ12 RTI Interrupt */
    vfnLFRIsr,               /* IRQ11 LFR Interrupt */
    vfnADCIsr,               /* IRQ10 ADC Interrupt */
    vfnRFMIsr,               /* IRQ9 RFM Interrupt */
    vfnTPMSSMIIsr,           /* IRQ8 SMI Interrupt - defined in
 firmware library */
    vfnTPMIsr,               /* IRQ7 TPM Overflow Interrupt */
    vfnTPMCH1Isr,            /* IRQ6 TPM Channel 1 Interrupt */
    vfnTPMCH0Isr,            /* IRQ5 TPM Channel 0 Interrupt */
    vfnPWUIsr,               /* IRQ4 PWU Interrupt */
    vfnLVDIsr,               /* IRQ3 LVD Interrupt */
    vfnIRQIsr,               /* IRQ2 IRQ Interrupt */
    vfnSWIIsr,               /* IRQ1 SWI Interrupt */
    /* _Startup               * IRQ0 Reset Vector - defined in
 Project.prm */
  };
```

Note on the LF interrupt vector

In firmware-based applications, not jumping in the LF user interrupt vector ensures all data bytes are read by the function TPMS_LF_READ_DATA. See Section 3.4.

However, in library-based applications there is no distinction between firmware and user interrupt. The LF interrupt vector defined by the application is accessed each time an interrupt occurs. To ensure a correct operation of TPMS_LF_READ_DATA, the user must ensure the execution time of the user code in the LF interrupt vector is short enough. If the user code execution time is too long, TPMS_LF_READ_DATA may not read all LF data bytes.

Note on the reset vector

For FXTH87/87E, the reset function "_FXTH87x6LibStartup" is defined in the firmware library. In the application, the user must configure this function to be the reset vector in the *prm* file:

```
VECTOR 0 _fxth87x6LibStartup /* Reset vector defined in
IsrVector */
```

The function "_FXTH87x6LibStartup" performs the following actions:

- LF registers LFCTRLB to LFCTRLE are configured with NXP recommended values;
- The stack pointer is set to address 28Fh (last address of the RAM);
- Jump to main;

For NTM88, the default CodeWarrior startup function is used as the reset vector and is defined in the *prm* file of the project:

```
VECTOR ADDRESS 0xFFFE _Startup
```

The statement performs the following actions:

- Sets the stack pointer to the last address of the stack allocated by the linker;
- Jumps to main;

Note: *if the preprocessor macro __ONLY_INIT_SP is not defined in the project, the _Startup function will also perform global variable initialization, which would overwrite all global variables upon exit from STOP1. This would also apply to the variables located in the RAM section preserved in STOP1. In order to avoid such a situation, make sure that the preprocessor macro __ONLY_INIT_SP is defined in the project. For that, always select the option "Minimal Startup Code" when creating a project from scratch, or manually add this macro in the project's Properties > C/C++ Build > Settings > HCS08 Compiler > Preprocessor.*

4.6 Protection and security registers

Library-based applications have access to the protection and security registers. The protection and security registers are located at addresses FFBDh to FFBFh[12]. Applications can directly configure these registers and firmware functions are no longer required to access these registers. As a result, the TPMS_FLASH_PROTECTION function, available in the embedded firmware programmed by NXP, is not available in the firmware library.

The user must now configure the security register in the application to maintain the device as "unsecured". When the section of the flash containing the security register is erased prior to programming, the security register takes the value FFh. If the security register is set to FFh when a hardware reset is done, it secures the device and prevents any further background debug mode (BDM) access. The user must ensure that the application code configures the security register to unsecure the MCU. For example, the user may configure the security register to value 82h[13].

For example, place the following declaration in main.c to keep the device "unsecure".

12 Refer to the Reference Manual.
13 Refer to the Reference Manual.

```
// This is to keep the device unsecure
volatile const UINT8 FLASH_NVOPT @0xFFBF = 0x82;
```

This declaration ensures that the security register is always programmed to 82h and the device is never secured.

4.7 TPMS_FLASH functions

Firmware functions TPMS_FLASH_WRITE and TPMS_FLASH_ERASE are provided to write and erase flash bytes. The function TPMS_FLASH_CHECK is provided to verify integrity of the flash programmed by NXP at production.

4.7.1 FXTH TPMS_FLASH functions

For FXTH, the functions available in the firmware library have access to the whole flash except the trim page from FC00h to FDFFh. They are not restricted to the section C000h to DFFFh anymore like in the firmware model.

The firmware library function TPMS_FLASH_CHECK is provided to check the trim section integrity. This function calculates the CRC16 checksum for the NXP trim area. The function compares the checksum with a pre-calculated stored value and reports whether the two values match.

4.7.2 NTM88 TPMS_FLASH functions

For NTM88, the function TPMS_FLASH_WRITE has access to the whole flash except the trim section from FD40h to FDFFh. It can be used to write bytes in the one-time programmable section from FC00h to FD3Fh.

The function TPMS_FLASH_ERASE can erase any memory page except the trim page from FC00h to FDFFh. The one-time programmable section from FC00h to FD3Fh cannot be erased because flash memory is erased page by page (512 bytes by 512 bytes) only. Erasing the one-time programmable section would imply erasing the trim coefficients, which must not be done.

The TPMS_FLASH_CHECK function is available to check the trim section integrity with compatible NTM88 devices[14] This function calculates the CRC16 checksum for the NXP trim area. The function compares the checksum with a pre- calculated stored value and reports whether the two values match.

5 Migrating from firmware to library model: summary of differences

Optimization of the flash memory is the advantage of the library model compared to the firmware model. With the firmware model, all firmware functions are placed in memory in the firmware flash. With the library model, only functions used by the application are placed in memory. The library model saves memory space.

14 Part numbers that cannot use TPMS_FLASH_CHECK are:

- PTM88H05 with CODEH = 15h or 95h
- PTM88H06 with CODEH = 16h or 96h
- PTM88H13 with CODEH = 1D or 9D
- PTM88H14 with CODEH = 1E or 9E

All other PTM88 and NTM88 can use the function.

.

Table 2 is a summary of the differences between the two models. Users must consider these differences when migrating the FXTH application from the firmware to the library model.

Table 2. Firmware and library model differences

Migration consideration points	Firmware model	Library model
Part numbers compatible with the model	All FXTH87 and FXTH87E part numbers.	FXTH87xx02 programmed by NXP with firmware 2Ah, and FXTH87xx1x programmed by NXP with firmware 24h. All NTM88 part numbers.
Target selected in the CodeWarrior project	FXTH87/87E: FXTH870000 (available by default when installing the latest CodeWarrior versions)	FXTH87/87E: FXTH870000_LIB (available after installing the FXTH870000_LIB patch) NTM88: NTM88_LIB (available after installing the NTM88_LIB patch)
Flash memory accessible by the application	User flash from C000h to DFFFh. Firmware flash from E000h to FFFFh is not accessible.	From C000h to FBFFh, and from FE00h to FFFFh. Only the trim page is not accessible.
Firmware functions placed in memory	All firmware functions are placed in memory in the firmware flash section.	Only the firmware functions used by the application are placed in memory.
Access to the firmware functions	Application calls the firmware functions via the jump table programmed by NXP.	Application calls the firmware functions via a firmware library added in the project.
Content of CODE 0 / CODE F byte returned by TPMS_READ_ID	Version of the embedded firmware programmed by NXP at production.	Version of the library added in the project.
Interrupt vectors	Upon interrupt, the program enters the firmware interrupt vector where TPMS_INTERRUPT_FLAG is updated when applicable, and then jumps in the user interrupt vector when applicable.	There is only one array of interrupt vectors which must be defined in the application. Code updating TPMS_INTERRUPT_FLAG must be added in the appropriate interrupt functions.
TPMS_FLASH_ERASE and TPMS_FLASH_WRITE	The functions can access flash from C000h to DFFFh only.	The functions can access the whole flash except the trim.
TPMS_FLASH_CHECK	The function checks the integrity of the embedded firmware programmed between E000h and FFFFh.	The function checks the integrity of the trim section.
TPMS_FLASH_PROTECTION	Is provided in the embedded firmware as the application does not have direct access to the protection register.	Is not available in the firmware library as the application has direct access to the protection register.
Security register	Configuration of the register is not supported.	The application has direct access to the security register so must configure the register to maintain the device unsecure after programming.

6 Glossary

Table 3. Glossary

Term	Definition
Library Model Applications	Applications which erase the embedded firmware programmed by NXP at production. In library model applications, there is no distinction between user flash and firmware flash. The firmware functions, used by the application, are added in the application via a firmware library.
Firmware Model Applications	Applications that maintain separate user flash, containing the user application, and firmware flash, containing the embedded firmware programmed by NXP at production. In firmware model applications, the firmware functions are called from the user application code via a jump table.

7 Revision history

Table 4. Revision history

Revision	Date	Description
2	20191210	• Section 2, revised content and image in Figure 1. • Section 4.2, split the last paragraph into two separate paragraphs, one for FXTH and the second for NTM88. • Section 4.3, revised the image and caption for Figure 5 and corrected register range values in the section narrative. • Section 4.5, added new note at the end of the section. • Section 4.7.1, revised "The firmware function..." to "The firmware library function..." in the second paragraph. • Section 4.7.2, revised the register value ranges in the section, revised the third paragraph and included new footnote. • Section 5, Table 2, revised the library model description for TPMS_FLASH_CHECK.
1	20190722	Initial release

8 Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Suitability for use in automotive applications — This NXP Semiconductors product has been qualified for use in automotive applications. Unless otherwise agreed in writing, the product is not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — While NXP Semiconductors has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP Semiconductors accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

CodeWarrior — is a trademark of NXP B.V.

NXP — is a trademark of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10 December 2019
Document identifier: AN12523