Freescale Semiconductor
Application Note

Document Number: AN4277
Rev. 1.0, 10/2014

Interfacing to Freescale's FXTH87xx In-Flash Firmware Routines Using C-language Constructors

by: Andres Barrilado
Stephane Lestringuez

Contents

1 Introduction

Freescale's FXTH87xx consist of complex system-on-chip products that contain a pressure sensor, a Radio Frequency (RF) transmitter, a Low Frequency (LF) receiver, an 8-bit S08 microcontroller (MCU), and may contain one or two axis accelerometers, all in one package. As with most S08 solutions, users must program the MCU with their program, but Freescale already ships low-level drivers, as well as compensation routines, within each part. In order to exploit these routines, the developer must be familiar not only with the MCU architecture, but also with the in-flash firmware. The aim of this document is to fast-track the learning curve of the latter part for someone that is already cognizant of the former.

Two separate documents are referenced repeatedly throughout this text. The firmware user guide is available through the web and contains details on the in-flash functions, variables, and their functionality. The device data sheet, or product spec, is also available for

freescale Sensors

download and addresses hardware architecture, register locations, and overall functionality. Both are recommended as reference for development.

2 Creating a New Project

For instructions on how to create a new CodeWarrior C project, refer to Section 4.2 in Freescale application note AN2616, *"Getting Started with HCS08 and CodeWarrior Using C, Revised for CodeWarrior v3.1 - v5.1"*.

When using CodeWarrior 10.x series, please refer to "*MCU_HCS08_COMPILER, HCS08 Build Tools Reference Manual for Microcontrollers V10.x"* and its derivative.

3 Declarations

As with any C-language project, it is a good idea to start by defining the types, global variables, and functions that will be used.

3.1 Special types

Freescale's in-flash firmware does not require any special types to be defined, allowing for a strict Motor Industry Software Reliability Association (MISRA) compilation if necessary. However, certain typedef definitions will make the developer's life easier. Throughout this document, special types are used as an integral part of interfacing to the in-flash firmware.

3.1.1 T_RFDATA

T_RFDATA can be used to configure the Radio Frequency Module (RFM) through the TPMS_RF_CONFIG_DATA routine. Figure 1 shows its definition. For more information on the meaning of each field, refer to the firmware user guide pertinent to your family.

```
typedef struct
{
  UINT8 Prescaler     : 8; /* RFCR0 */
  UINT8 Modulation    : 1;
  UINT8 Frequency     : 1;
  UINT8 Encoding      : 2; /* CODE - Manchester, BiPhase, NRZ or Direct */
  UINT8               : 2;
  UINT8 Polarity      : 1;
  UINT8 EndOfMessage  : 1;
  UINT16 PllA;
  UINT16 PllB;
} T_RFDATA;
```

Figure 1. T_RFDATA typedef

3.2 Shared global variables

Depending on the family of TPMS sensors two global variables must be declared and taken into account for development.

3.2.1 TPMS_INTERRUPT_FLAG

Common between all families and derivatives, this 8-bit variable is used by firmware to identify when a particular interrupt has taken place. Its content, as well as its absolute memory location is defined in the pertinent firmware user guide.

This variable must be declared as shown in Figure 2, being careful to replace TPMS_INTERRUPT_FLAG_LOCATION with the correct value provided in the firmware user guide. Typically, this location is $8F, which is the last location of non-volatile RAM.

```
#define TPMS_INTERRUPT_FLAG_LOCATION ((UINT16)0x008Fu);
extern UINT8 TPMS_INTERRUPT_FLAG @ TPMS_INTERRUPT_FLAG_LOCATION;
```

Figure 2. TPMS_INTERRUPT_FLAG definition

3.2.2 TPMS_CONT_ACCEL_GLOBAL

This variable's content, as well as its absolute memory location, is defined in relevant firmware user guides. This variable must be declared as shown in Figure 3, being careful to replace TPMS_CONT_ACCEL_GLOBAL_LOCATION with the correct value. Typically, this is location $8E, which is part of non-volatile RAM.

```
#define TPMS_CONT_ACCEL_GLOBAL_LOCATION  ((UINT16)0x008Eu);
extern UINT8 TPMS_INTERRUPT_FLAG @ TPMS_CONT_ACCEL_GLOBAL_LOCATION;
```

Figure 3. TPMS_CONT_ACCEL_GLOBAL definition

3.3 Universal uncompensated measurement array

The universal uncompensated measurement array (UUMA) is a four or five word-long array that must be placed in memory to be able to call measurement routines. Its size varies depending on the family; four words are needed for devices with one-axis accelerometers (one for each uncompensated measurement available, i.e., voltage, temperature, pressure, and Z-axis acceleration), while five must be reserved for devices with two axes accelerometers (one more for the X-axis acceleration measurement). As its name indicates, it provides a standard, organized format for uncompensated measurements. A typical declaration is shown in Figure 4. For more information regarding the structure and use of the UUMA, refer to the firmware user guide relevant to your family.

```
/* Declaration for the FXTH87xx02 */
volatile UINT16 au16UUMA[4u];

/* Declaration for the FXTH87xx11 */
volatile UINT16 au16UUMA[5u];
```

Figure 4. UUMA Declaration

3.4 Interfaces to in-flash firmware routines

In-flash firmware routines interface with user firmware through an in-flash look-up table. Each routine is assigned a unique, absolute address, and can be called by declaring a pointer to its location. The example in Figure 5 interfaces to TPMS_READ_PRESSURE, located at address $E00F in all families. Similar pointers can be created for all functions. For a full example, download the accompanying code from the web and refer to the accompanying .h file.

For more information on functions available, and their absolute location, refer to the pertinent user guide.

```
/* UINT8 TPMS_READ_PRESSURE(UINT16 *u16UUMA, UINT8 u8Avg) */
#define  TPMS_READ_PRESSURE          ((UINT8(*)(UINT16*, UINT8))((UINT16)0xE00Fu))
```

Figure 5. Sample Interface to an in-flash firmware routine

3.5 Calling Freescale's in-flash routines

Once in-flash routines have been declared as pointers, calling them should be as simple as calling any other function. Figure 6 illustrates this with the pointer definition performed in Figure 5. For more information on available functions, and their prototypes, refer to the corresponding user guide.

```
  /* Declare local variables */
  UINT8  u8Status = CLEAR;
  UINT16 au16UUMA[4u];

  u8Status = TPMS_READ_PRESSURE(au16UUMA, 1u);
```

Figure 6. Sample call to TPMS_READ_PRESSURE

3.6 Declaring interrupts

In the FXTH87xx family of products, interrupts are first handled by the in-flash firmware, and then, if relevant, handed over to the user. The application developer is in charge of correctly handling these interrupts through a set of pseudo-vectors that behave identically to regular interrupt vectors, but are defined in a different location. User interrupt pseudo-vectors must be declared starting in location $DFE0. Figure 7 shows a sample declaration. Note that the “main” is also declared here. For a comprehensive list of interrupt vectors and their functions, refer to the section titled “Resets, Interrupts and System Configuration” in the pertinent product’s data sheet.

```
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
};
```

Figure 7. Sample user interrupt vector declaration

3.7 Initializing shared global variables

All shared global variables must be initialized to “0” during the first execution cycle of the program. The PDF bit inside the SPMSC2 register is cleared if this is the case, or set otherwise. Refer to Figure 8 for an example of how to do so, and to the product’s data sheet. for more information on the PDF bit behavior.

```
  /* Check if this is the first time we're executing code */
  if(SPMSC2_PDF == CLEAR)
  {
    TPMS_INTERRUPT_FLAG = CLEAR;
    TPMS_CONT_ACCEL_GLOBAL = CLEAR;
  }
```

Figure 8. Sample initialization of shared global variables

4 Conclusion

Using the simple pointers presented here, the developer can quickly interface to the in-flash routines. Sample projects for the FXTH87xx that already implement what has been described in this application note can be downloaded from the web and serve as templates as well.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4277
Rev. 1.0
10/2014