AN2776
Application note

How to migrate from STM32F10xxx FWLib V1.0 to V2.0

Introduction

The purpose of this application note is to explain how to migrate an application based on the STM32F10xxx firmware library (FWLib) V1.0 to STM32F10xxx FWLib V2.0. To that aim, this document does not provide detailed information on the two library versions, but it highlights the differences between them.

In the rest of the document (unless otherwise specified), the term of FWLib will be used to refer to the STM32F10xxx firmware library.

Glossary

Medium-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 32 and 128 Kbytes.

High-density devices are STM32F101xx and STM32F103xx microcontrollers where the Flash memory density ranges between 256 and 512 Kbytes.

Contents

List of tables

1 Why migrate from FWLib V1.0 to V2.0?

FWLib V1.0 is a complete firmware package for Medium-density STM32F10xxx devices. It is a collection of routines, data structures and macros, which covers the features of all peripherals. It includes drivers plus a set of examples for all of the standard device peripherals.

FWLib V2.0 is an updated version of FWLib V1.0 that supports Medium-density devices and the additional peripherals and features embedded in High-density STM32F10xxx devices:

- Embedded memories
  - SRAM: up to 64 KB (instead of 20 KB in Medium-density devices)
  - FLASH: up to 512 KB (instead of 128 KB in Medium-density devices)
- New peripherals
  - SDIO
  - I2S2 and I2S3 (multiplexed with SPI2 and SPI3)
  - DAC
  - FSMC
- New peripheral instantiation
  - DMA2 controller supporting 5 channels
  - ADC3
  - SPI3
  - UART4 and UART5 do not support hardware flow control, Smartcard mode (ISO 7816 compliant) and SPI-like communication
  - TIM8 (advanced-control timer)
  - TIM5 (general-purpose Timer)
  - TIM6 and TIM7 (basic timers that can be used to trigger the DAC)
- Miscellaneous changes
  - Backup user data extended from 20 bytes to 84 bytes
  - Up to 112 I/Os: port F and G added (16 pins each)
  - LSI calibration: LSI clock connected to the TIM5_CH4 input capture for calibration purposes
  - 112 EXTI line inputs instead of 80
  - 17 Interrupt sources added (total of 60 interrupts)
  - external/internal ADC triggers

The main differences between FWLib V1.0 and FWLib V2.0 are listed below:

1. Update of library core files (`stm32f10x_map.h`, `stm32f10x_lib.h`, `stm32f10x_lib.c` and `stm32f10x_conf.h`) to add new peripheral register declarations and header file inclusions
2. Update of `stm32f10x_vector.s` (or `stm32f10x_vector.c`, depending on the used toolchain) and `stm32f10x_it.c`/`stm32f10x_it.h`:
   device vector table updated with new peripheral IRQ channels
3. Drivers added for new peripherals (like `stm32f10x_fsmc.c`/`stm32f10x_fsmc.h`)

4. Update of existing drivers to support the added peripheral instantiation and miscellaneous additional changes. On this occasion, the APIs (application programming interfaces) of some drivers were changed, leading to loss of compatibility with FWLib V1.0. This applies to the drivers of the following peripherals: TIM, SPI, DMA, USART and NVIC.

Note: 1 Loss of compatibility means that an application code based on one of the above listed FWLib V1.0 peripheral drivers needs updating to work correctly with FWLib V2.0.

2 FWLib V2.0 comes with eighty-seven (87) examples that cover a wide range of the peripheral functionalities.

If FWLib V1.0 is used in an application based on Medium-density STM32F10xxx devices and migration to High-density STM32F10xxx devices is contemplated, two cases should be considered:

- The purpose of the migration is to use High-density devices only to benefit from the larger sizes of embedded memories. In this case, you can still use FWLib V1.0. You only need to update the development toolchain-specific files (linker, Flash loader, etc.)
- The purpose of the migration is to use the additional peripherals introduced in High-density devices. In this case, you have to migrate to FWLib V2.0. The development toolchain-specific files also need updating (linker, Flash loader, etc.).

2 FWLib V1.0 to V2.0 migration steps

This section describes how to migrate an application based on FWLib V1.0 to V2.0.

2.1 Migration steps

To update your application code to run on FWLib V2.0, you have to follow the steps listed below:

1. Update the toolchain startup files
   a) Project files: device connections and Flash memory loader. These files are provided with the latest version of your toolchain that supports High-density and Medium-density STM32F10xxx devices. For more information please refer to your toolchain documentation.
   b) Linker configuration and vector table location files. These files are included in the FWLib V2.0 install package under the following directory: *STM32F10xFWLib\FWLib\project*
2. Add FWLib V2.0 source files to the application sources
   a) Replace the `stm32f10x_conf.h` file of your application with the latest version provided in FWLib V2.0.
   b) The `stm32f10x_it.c`/`stm32f10x_it.h` files that contain the peripheral ISRs (interrupt service routine), have been updated to support the new peripherals and the IRQ channel updates of existing peripherals.
   You have to replace the existing `stm32f10x_it.h` file in your application with the new version provided in FWLib V2.0 and, copy your code from the existing `stm32f10x_it.c` file to the new one.
   c) Update the part of your application code that uses the TIM, SPI, DMA, USART and NVIC drivers. Further details are provided in the next section.

2.2 Peripheral driver updates

This section describes how to port an application code based on the FWLib V1.0 TIM, SPI, DMA, USART and NVIC drivers to FWLib V2.0.

Note: In all examples listed below, source code changes between FWLib V1.0 and V2.0 are indicated in italic and bold.

2.2.1 NVIC

Due to a different number of DMA and ADC peripherals in Medium-density and High density devices, NVIC-related code was modified between FWLib V1.0 and FWLib V2.0. These changes concern DMA and ADC interrupts.

DMAs

In Medium-density devices there was only one DMA controller with seven channels (DMA Channel1 to DMA Channel7). In High-density devices, there is an additional DMA controller that supports five channels (DMA2 Channel1 to DMA2 Channel5). As a result, the DMA already present in Medium-density devices was renamed as DMA1:

- DMA1 (Channel1 to Channel7) present in Medium-density and High-density devices.
- DMA2 (Channel1 to Channel5) present only in High-density devices.

This change creates the need to update the DMA IRQ channel names that have been modified in `stm32f10x_nvic.c`/`stm32f10x_nvic.h`, as detailed in *Table 1* below.

Table 1. Update of DMA IRQ channel names

FWLib V1.0	FWLib V2.0
DMAChannel1_IRQChannel	DMA1_Channel1_IRQChannel
DMAChannel2_IRQChannel	DMA1_Channel2_IRQChannel
DMAChannel3_IRQChannel	DMA1_Channel3_IRQChannel
DMAChannel4_IRQChannel	DMA1_Channel4_IRQChannel
DMAChannel5_IRQChannel	DMA1_Channel5_IRQChannel
DMAChannel6_IRQChannel	DMA1_Channel6_IRQChannel
DMAChannel7_IRQChannel	DMA1_Channel7_IRQChannel

The example below shows how to configure and enable the DMA channel6 IRQ channel using FWLib V1.0:

```
/* Configure and enable DMA channel6 IRQ Channel */
NVIC_InitStructure.NVIC_IRQChannel = DMAChannel6_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

In FWLib V2.0, you simply need to change “DMAChannel6_IRQChannel” by”DMA1_Channel6_IRQChannel”

```
/* Configure and enable DMA1 channel6 IRQ Channel */
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel6_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

ADCs

In the same way, Medium-density devices feature two ADCs (ADC1 and ADC2) that share the same IRQ channel. High-density devices have an additional ADC (ADC3) with a separate IRQ channel.

For this reason, the ADC1 and ADC2 IRQ channel names were modified in `stm32f10x_nvic.c/stm32f10x_nvic.h`, as detailed in *Table 2* below.

Table 2. ADC1 and ADC2 IRQ channel name update

FWLib V1.0	FWLib V2.0
ADC_IRQChannel	ADC1_2_IRQChannel

The example below shows how to configure and enable the ADC1 (or ADC2) IRQ channel using FWLib V1.0:

```
/* Configure and enable ADC1 IRQ channel */
NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

In V2.0, you simply need to change “ADC_IRQChannel” by “ADC1_2_IRQChannel”.

```
/* Configure and enable ADC1 IRQ channel */
NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Note: The corresponding DMA and ADC ISR names have also been modified in `stm32f10_it.c/.stm32f10_it.h`.

2.2.2 TIM

In FWLib V1.0 the timer peripherals are covered by two drivers:

- *`stm32f10x_tim.c/stm32f10x_tim.h`*: they cover the functionality of general-purpose timers (TIM2, TIM3 and TIM4). The function format in this driver is as follows:

  ```
  TIM_FunctionName(TIMx, param1, param2,...)
  e.g.: TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)
  ```

- *`stm32f10x_tim1.c/stm32f10x_tim1.h`*: they cover the functionality of the advanced-control timer (TIM1). This driver provides the same set of functions as *`stm32f10x_tim.c/stm32f10x_tim.h`* plus dedicated functions to cover addtional TIM1 features (PWMI mode, complementary PWM with dead-time insertion, etc.). The function format in this driver is as follows:

  ```
  TIM1_FunctionName(param1, param2,...)
  e.g.: TIM1_ITConfig(u16 TIM1_IT, FunctionalState NewState)
  ```

In FWLib V2.0, these two drivers have been merged into a single driver, `stm32f10x_tim.c/stm32f10x_tim.h`, that covers all the timers available in High-density and Medium-density devices: general-purpose timers (TIM2, TIM3, TIM4 and TIM5), advanced-control timers (TIM1 and TIM8) and basic timers (TIM6&7).

This change affects only application code based on the *`stm32f10x_tim1.c/stm32f10x_tim1.h`* driver. The set of functions for TIM1 are the

same but the name is different. In the function name, “TIM1” was changed to “TIM” and a new parameter was added to specify the timer to be configured.

e.g.: in FWLib V1.0 the function used to configure the TIM1 interrupt is:

```
TIM1_ITConfig(TIM_IT_Break, ENABLE)
```

in FWLib V2.0 the function is:

```
TIM_ITConfig(TIM1, TIM_IT_Break, ENABLE)
```

Some typical examples of the TIM1 configuration migration from FWLib V1.0 to V2.0 are given below. For the complete list of changes made in the timer driver, please refer to *Appendix A*.

Time-base configuration

Typical code for TIM1 time-base configuration based on FWLib V1.0:

```
TIM1_TimeBaseStructure.TIM1_Prescaler = 0;
TIM1_TimeBaseStructure.TIM1_CounterMode = TIM1_CounterMode_Up;
TIM1_TimeBaseStructure.TIM1_Period = 4095;
TIM1_TimeBaseStructure.TIM1_ClockDivision = 0;
TIM1_TimeBaseStructure.TIM1_RepetitionCounter = 0;
TIM1_TimeBaseInit(&TIM1_TimeBaseStructure);
```

Using FWLib V2.0, the function name has to be updated and TIM1 should be passed as the first parameter value:

```
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Period = 4095;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
```

TIM1 output compare mode configuration

Typical code for TIM1 output compare mode configuration based on FWLib V1.0:

```
TIM1_OCInitStructure.TIM1_OCMode = TIM1_OCMode_Timing;
TIM1_OCInitStructure.TIM1_OutputState = TIM1_OutputState_Enable;
TIM1_OCInitStructure.TIM1_OutputNState = TIM1_OutputNState_Enable;
TIM1_OCInitStructure.TIM1_Pulse = 2047;
TIM1_OCInitStructure.TIM1_OCPolarity = TIM1_OCPolarity_High;
TIM1_OCInitStructure.TIM1_OCNPolarity = TIM1_OCNPolarity_High;
TIM1_OCInitStructure.TIM1_OCIdleState = TIM1_OCIdleState_Set;
TIM1_OCInitStructure.TIM1_OCNIdleState = TIM1_OCIdleState_Set;
TIM1_OC1Init(&TIM1_OCInitStructure);
TIM1_OCInitStructure.TIM1_Pulse = 1023;
TIM1_OC2Init(&TIM1_OCInitStructure);
TIM1_OCInitStructure.TIM1_Pulse = 511;
TIM1_OC3Init(&TIM1_OCInitStructure);
```

Using FWLib V2.0, the function names have to be updated and TIM1 should be passed as the first parameter value:

```
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = 2047;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Set;
TIM_OC1Init(TIM1, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_Pulse = 1023;
TIM_OC2Init(TIM1, &TIM_OCInitStructure);
```

```
TIM_OCInitStructure.TIM_Pulse = 511;
TIM_OC3Init(TIM1, &TIM_OCInitStructure);
```

TIMx output compare mode configuration

Typical code for TIMx output compare mode configuration based on FWLib V1.0:

```
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_Channel = TIM_Channel_1;
TIM_OCInitStructure.TIM_Pulse = 2047;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInit(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_Channel = TIM_Channel_2;
TIM_OCInitStructure.TIM_Pulse = 1023;
TIM_OCInit(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_Channel = TIM_Channel_3;
TIM_OCInitStructure.TIM_Pulse = 511;
TIM_OCInit(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_Channel = TIM_Channel_4;
TIM_OCInitStructure.TIM_Pulse = 255;
TIM_OCInit(TIM2, &TIM_OCInitStructure);
```

Using FWLib V2.0, the function names have to be updated and the `TIM_Channel` parameter is replaced by a new stucture member: `TIM_OutputState`:

```
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 2047;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OC1Init(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 1023;
TIM_OC2Init(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 511;
TIM_OC3Init(TIM2, &TIM_OCInitStructure);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 255;
TIM_OC4Init(TIM2, &TIM_OCInitStructure);
```

Counter enable and PWM output control

Typical code for TIM1 counter enable and PWM output control configuration based on FWLib V1.0:

```
/* TIM1 counter enable */
TIM1_Cmd(ENABLE);

/* Main Output Enable */
TIM1_CtrlPWMOutputs(ENABLE);
```

Using FWLib V2.0, the function names have to be updated and TIM1 should be passed as the first parameter value:

```
/* TIM1 counter enable */
TIM_Cmd(TIM1, ENABLE);

/* Main Output Enable */
TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

Input capture configuration

Typical code for TIM input capture configuration based on FWLib V1.0:

```
TIM_ICInitStructure.TIM_ICMode = TIM_ICMode_ICAP;
TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
```

```
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;
TIM_ICInit(TIM3, &TIM_ICInitStructure);
TIM_SelectInputTrigger(TIM3, TIM_TS_TI1FP1);
TIM_SelectSlaveMode(TIM3, TIM_SlaveMode_Reset);
/* TIM enable counter */
TIM_Cmd(TIM3, ENABLE);
```

In FWLib V2.0, the *TIM_ICMode* member of the TIM_ICInitTypeDef structure was removed:

```
TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;
TIM_ICInit(TIM3, &TIM_ICInitStructure);
TIM_SelectInputTrigger(TIM3, TIM_TS_TI1FP1);
TIM_SelectSlaveMode(TIM3, TIM_SlaveMode_Reset);
/* TIM enable counter */
TIM_Cmd(TIM3, ENABLE);
```

PWM input capture configuration

Typical code for TIM PWM input capture configuration based on FWLib V1.0:

```
TIM_ICInitStructure.TIM_ICMode = TIM_ICMode_PWMI;
TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;
TIM_ICInit(TIM2, &TIM_ICInitStructure);
TIM_SelectInputTrigger(TIM2, TIM_TS_TI1FP1);
TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Reset);
```

In FWLib V2.0, the *TIM_ICMode* member of the TIM_ICInitTypeDef structure was removed and a new function, *TIM_PWMIConfig*, was added:

```
TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;
TIM_PWMIConfig(TIM2, &TIM_ICInitStructure);
TIM_SelectInputTrigger(TIM2, TIM_TS_TI1FP1);
TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Reset);
```

2.2.3 SPI

In FWLib V1.0, the *stm32f10x_spi.c/stm32f10x_spi.h* driver is used to cover the functionality of the SPI1 and SPI2 peripherals. The function format in this driver is as follows:

```
SPI_FunctionName(SPIx, param1, param2,...)
e.g.: SPI_ITConfig(SPI_TypeDef* SPIx, u8 SPI_IT, FunctionalState NewState)
```

In FWLib V2.0, this driver was updated to support the new SPI3 peripheral and the I^2S protocol.

In high-density devices, the SPI2 and SPI3 interfaces give the flexibility to use either the SPI protocol or the I^2S audio protocol. These two protocols share some configuration bits plus a dedicated bit for each protocol.

The following sections describes the update made on this driver.

Common functions and configuration parameters for SPI and I²S

Some SPI driver functions were updated to support the I²S protocol:

1. "I2S" was added in the function name to indicate that it can be used for SPI and I²S
2. "I2S" was added in the configuration parameter to indicate that it can be used for SPI and I²S
3. The function format is updated as follows:

```
SPI_I2S_FunctionName(SPIx, param1, param2,...)
e.g.: SPI_I2S_ITConfig(SPI_TypeDef* SPIx, u8 SPI_I2S_IT, FunctionalState NewState)
```

Table 3 provides the list of the impacted functions.

Table 3. Common functions for SPI and I²S(1)

FWLib V1.0	FWLib V2.0
`SPI_DeInit(SPI_TypeDef* SPIx)`	`SPI_I2S_DeInit(SPI_TypeDef* SPIx)`
`SPI_ITConfig(SPI_TypeDef* SPIx, u8 SPI_IT, FunctionalState NewState)`	`SPI_I2S_ITConfig(SPI_TypeDef* SPIx, u8 SPI_I2S_IT, FunctionalState NewState)`
`SPI_DMACmd(SPI_TypeDef* SPIx, u16 SPI_DMAReq, FunctionalState NewState)`	`SPI_I2S_DMACmd(SPI_TypeDef* SPIx, u16 SPI_I2S_DMAReq, FunctionalState NewState)`
`SPI_SendData(SPI_TypeDef* SPIx, u16 Data)`	`SPI_I2S_SendData(SPI_TypeDef* SPIx, u16 Data)`
`SPI_ReceiveData(SPI_TypeDef* SPIx)`	`SPI_I2S_ReceiveData(SPI_TypeDef* SPIx)`
`SPI_GetFlagStatus(SPI_TypeDef* SPIx, u16 SPI_FLAG)`	`SPI_I2S_GetFlagStatus(SPI_TypeDef* SPIx, u16 SPI_I2S_FLAG)`
`SPI_ClearFlag(SPI_TypeDef* SPIx, u16 SPI_FLAG)`	`SPI_I2S_ClearFlag(SPI_TypeDef* SPIx, u16 SPI_I2S_FLAG)`
`SPI_GetITStatus(SPI_TypeDef* SPIx, u8 SPI_IT)`	`SPI_I2S_GetITStatus(SPI_TypeDef* SPIx, u8 SPI_I2S_IT)`
`SPI_ClearITPendingBit(SPI_TypeDef* SPIx, u8 SPI_IT)`	`SPI_I2S_ClearITPendingBit(SPI_TypeDef* SPIx, u8 SPI_I2S_IT)`

1. Some parameters are specific for SPI (e.g.: SPI_FLAG_CRCERR) or I²S (e.g.: I2S_FLAG_CHSIDE).

If you are using any function listed in the first column of *Table 3*, update the prototypes and parameter name to the new names given in the second column of the table.

Example: Typical code for SPI interrupt and DMA configuration based on FWLib V1.0:

```
SPI_ITConfig(SPI2, SPI_IT_TXE, ENABLE)
SPI_DMACmd(SPI2, SPI_DMAReq_Rx, ENABLE)
```

Using FWLib V2.0, this code becomes:

```
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_TXE, ENABLE)
SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Rx, ENABLE)
```

Note: In addition to these functions, a new structure is added to this driver to configure only the I²S peripheral.

SPI-specific functions

The functions not listed in *Table 3* where not modified and are specific to the SPI protocol.

New functions for I²S

Three functions were added to cover I²S protocol-specific configuration:

1. `I2S_Init(SPI_TypeDef* SPIx, I2S_InitTypeDef* I2S_InitStruct)`
2. `I2S_StructInit(I2S_InitTypeDef* I2S_InitStruct)`
3. `I2S_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)`

2.2.4 DMA

In FWLib V1.0, the *stm32f10x_dma.c/stm32f10x_dma.h* driver is used to cover the functionality of the DMA controller and its seven channels. The function format in this driver is as follows:

```
DMA_FunctionName(DMA_Channelx, param1, param2,...)
e.g.: DMA_ITConfig(DMA_Channel_TypeDef* DMA_Channelx, u32 DMA_IT, FunctionalState
NewState)
```

In FWLib V2.0, this driver was updated to support the new DMA2 controller and the five associated channels present in High-density devices. The function names are unchanged compared to FWLib V1.0, only the passed parameter has been updated to cover the DMA1 (DMA in Medium-density devices) and DMA2 controllers. This gives the following function format:

```
DMA_FunctionName(DMAy_Channelx, param1, param2,...)
e.g.: DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, u32 DMA_IT, FunctionalState
NewState)
```

As a result, in code based on FWLib V1.0, only the DMA_Channel_TypeDef parameter name needs updating.

Example: Typical code for DMA enable and interrupt configuration based on FWLib V1.0:

```
DMA_Cmd(DMA_Channel5, ENABLE)
DMA_ITConfig(DMA_Channel6, DMA_IT_TC, ENABLE)
```

Using FWLib V2.0, this code is:

```
DMA_Cmd(DMA1_Channel5, ENABLE)
DMA_ITConfig(DMA1_Channel6, DMA_IT_TC, ENABLE)
```

Example: Typical code to check whether DMA Channel6 flag is set, based on FWLib V1.0:

```
/* Test if DMA Channel6 transfer complete flag is set or not */
FlagStatus Status;
Status = DMA_GetFlagStatus(DMA_FLAG_TC6);
```

Using FWLib V2.0, this code is:

```
/* Test if DMA1 Channel6 transfer complete flag is set or not */
FlagStatus Status;
Status = DMA_GetFlagStatus(DMA1_FLAG_TC6);
```

2.2.5 USART

In FWLib V1.0, the *stm32f10x_usart.c/stm32f10x_usart.h* driver is used to cover the functionality of the USART1, USART2 and USART3 peripherals. In this driver, the USART is configured through the *USART_Init* function, which takes the *USART_InitTypeDef* structure as a parameter. It is the structure that contains the configuration information:

```
USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
```

The *USART_InitTypeDef* structure is defined as shown below:

```
typedef struct
{
  u32 USART_BaudRate;
  u16 USART_WordLength;
  u16 USART_StopBits;
  u16 USART_Parity;
  u16 USART_HardwareFlowControl;
  u16 USART_Mode;
  u16 USART_Clock;
  u16 USART_CPOL;
  u16 USART_CPHA;
  u16 USART_LastBit;
```

```
} USART_InitTypeDef;
```

In FWLib V2.0, this driver was updated to support the new UART4 and UART5 peripherals present in High-density devices. UART4 and UART5 do not support synchronous operations like Smartcard mode and SPI-like communication.

USART_Init was split up into two functions:

USART_Init

It is used for basic and asynchronous parameter configuration of USART1, USART2, USART3, UART4 and UART5:

```
USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
```

The *USART_InitTypeDef* structure definition is modified as follows:

```
typedef struct
{
  u32 USART_BaudRate;
  u16 USART_WordLength;
  u16 USART_StopBits;
  u16 USART_Parity;
  u16 USART_Mode;
  u16 USART_HardwareFlowControl;
} USART_InitTypeDef;
```

As a result, in code based on FWLib V1.0, if the application does not use the synchronous parameters, update the code as given in the following example:

USART configuration based on FWLib V1.0:

```
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No ;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStructure.USART_Clock = USART_Clock_Disable;
USART_InitStructure.USART_CPOL = USART_CPOL_Low;
USART_InitStructure.USART_CPHA = USART_CPHA_2Edge;
USART_InitStructure.USART_LastBit = USART_LastBit_Disable;
/* Configure USART1 */
USART_Init(USART1, &USART_InitStructure);
```

In FWLib V2.0, the USART_Clock, USART_CPOL, USART_CPHA and USART_LastBit structure members were removed:

```
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No ;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
/* Configure USART1 basic and asynchronous paramters */
USART_Init(USART1, &USART_InitStructure);
```

USART_ClockInit

It is used for the synchronous parameter configuration of USART1, USART2, and USART3:

```
USART_ClockInit(USART_TypeDef* USARTx,USART_ClockInitTypeDef*USART_ClockInitStruct)
```

The USART_ClockInitTypeDef structure is defined as shown below:

```
typedef struct
{
  u16 USART_Clock;
  u16 USART_CPOL;
  u16 USART_CPHA;
  u16 USART_LastBit;
} USART_ClockInitTypeDef;
```

As a result, in code based on FWLib V1.0, if the application uses the synchronous parameters, update the code as shown in the following example:

USART configuration based on FWLib V1.0:

```
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No ;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStructure.USART_Clock = USART_Clock_Enable;
USART_InitStructure.USART_CPOL = USART_CPOL_High;
USART_InitStructure.USART_CPHA = USART_CPHA_2Edge;
USART_InitStructure.USART_LastBit = USART_LastBit_Enable;
USART_Init(USART1, &USART_InitStructure);
/* Configure the USART1 */
USART_Init(USART1, &USART_InitStructure);
```

In FWLib V2.0, this configuration is done through two functions:

```
USART_ClockInitStructure.USART_Clock = USART_Clock_Enable;
USART_ClockInitStructure.USART_CPOL = USART_CPOL_High;
USART_ClockInitStructure.USART_CPHA = USART_CPHA_2Edge;
USART_ClockInitStructure.USART_LastBit = USART_LastBit_Enable;
/* Configure the USART1 synchronous paramters */
USART_ClockInit(USART1, &USART_ClockInitStructure);

USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No ;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
/* Configure USART1 basic and asynchronous paramters */
USART_Init(USART1, &USART_InitStructure);
```

Appendix A Timer driver changes

Table 4 gives the list of the changes made in the timer driver.

Table 4. Changes made in the timer driver

STM32 FWLib V1.0 stm32f10x_tim1.c, .h functions	STM32 FWLib V1.0 stm32f10x_tim.c, .h functions	STM32 FWLib V2.0 stm32f10x_tim.c, .h functions
void TIM1_DeInit(void)	void TIM_DeInit()	void TIM_DeInit()
void TIM1_TimeBaseInit()	void TIM_TimeBaseInit()	void TIM_TimeBaseInit()
void TIM1_OC1Init()	void TIM_OCInit()	void TIM_OC1Init()(1)
void TIM1_OC2Init()		void TIM_OC2Init()(1)
void TIM1_OC3Init()		void TIM_OC3Init()(1)
void TIM1_OC4Init()		void TIM_OC4Init()(1)
void TIM1_ICInit()	void TIM_ICInit()	void TIM_ICInit()
void TIM1_PWMIConfig()		void TIM_PWMIConfig()(1)
void TIM1_TimeBaseStructInit()	void TIM_TimeBaseStructInit()	void TIM_TimeBaseStructInit()
void TIM1_OCStructInit()	void TIM_OCStructInit()	void TIM_OCStructInit()
void TIM1_ICStructInit()	void TIM_ICStructInit()	void TIM_ICStructInit()
void TIM1_BDTRStructInit()		void TIM_BDTRStructInit()(2)
void TIM1_Cmd()	void TIM_Cmd()	void TIM_Cmd()
void TIM1_CtrlPWMOutputs()		void TIM_CtrlPWMOutputs()(2)
void TIM1_ITConfig()	void TIM_ITConfig()	void TIM_ITConfig()
void TIM1_DMAConfig()	void TIM_DMAConfig()	void TIM_DMAConfig()
void TIM1_DMACmd()	void TIM_DMACmd()	void TIM_DMACmd()
void TIM1_InternalClockConfig(void)	void TIM_InternalClockConfig()	void TIM_InternalClockConfig(void)
void TIM1_ETRClockMode1Config()	void TIM_ETRClockMode1Config()	void TIM_ETRClockMode1Config()
void TIM1_ETRClockMode2Config()	voidTIM_ETRClockMode2Config()	void TIM_ETRClockMode2Config()
void TIM1_ETRConfig()	void TIM_ETRConfig()	void TIM_ETRConfig()
void TIM1_ITRxExternalClockConfig()	void TIM_ITRxExternalClockConfig();	void TIM_ITRxExternalClockConfig()
void TIM1_TIxExternalClockConfig()	void TIM_TIxExternalClockConfig()	void TIM_TIxExternalClockConfig()
void TIM1_SelectInputTrigger()	void TIM_SelectInputTrigger()	void TIM_SelectInputTrigger()
void TIM1_UpdateDisableConfig()	void TIM_UpdateDisableConfig()	void TIM_UpdateDisableConfig()
void TIM1_UpdateRequestConfig()	void TIM_UpdateRequestConfig()	void TIM_UpdateRequestConfig()
void TIM1_SelectHallSensor()	void TIM_SelectHallSensor();	void TIM_SelectHallSensor()
void TIM1_SelectOnePulseMode()	void TIM_SelectOnePulseMode();	void TIM_SelectOnePulseMode()
void TIM1_SelectOutputTrigger()	void TIM_SelectOutputTrigger();	void TIM_SelectOutputTrigger()
void TIM1_SelectSlaveMode()	void TIM_SelectSlaveMode();	void TIM_SelectSlaveMode()

Table 4. Changes made in the timer driver (continued)

STM32 FWLib V1.0 stm32f10x_tim1.c, .h functions	STM32 FWLib V1.0 stm32f10x_tim.c, .h functions	STM32 FWLib V2.0 stm32f10x_tim.c, .h functions
void TIM1_SelectMasterSlaveMode()	void TIM_SelectMasterSlaveMode()	void TIM_SelectMasterSlaveMode()
void TIM1_EncoderInterfaceConfig()	void TIM_EncoderInterfaceConfig()	void TIM_EncoderInterfaceConfig()
void TIM1_PrescalerConfig()	void TIM_PrescalerConfig()	void TIM_PrescalerConfig()
void TIM1_CounterModeConfig()	void TIM_CounterModeConfig()	void TIM_CounterModeConfig()
void TIM1_ForcedOC1Config()	void TIM_ForcedOC1Config()	void TIM_ForcedOC1Config()
void TIM1_ForcedOC2Config()	void TIM_ForcedOC2Config()	void TIM_ForcedOC2Config()
void TIM1_ForcedOC3Config()	void TIM_ForcedOC3Config()	void TIM_ForcedOC3Config()
void TIM1_ForcedOC4Config()	void TIM_ForcedOC4Config()	void TIM_ForcedOC4Config()
void TIM1_ARRPreloadConfig()	void TIM_ARRPreloadConfig()	void TIM_ARRPreloadConfig()
void TIM1_SelectCOM()		void TIM_SelectCOM()
void TIM1_SelectCCDMA()	void TIM_SelectCCDMA()	void TIM_SelectCCDMA()
void TIM1_CCPreloadControl()		void TIM_CCPreloadControl()
void TIM1_OC1PreloadConfig()	void TIM_OC1PreloadConfig()	void TIM_OC1PreloadConfig()
void TIM1_OC2PreloadConfig()	void TIM_OC2PreloadConfig()	void TIM_OC2PreloadConfig()
void TIM1_OC3PreloadConfig()	void TIM_OC3PreloadConfig()	void TIM_OC3PreloadConfig()
void TIM1_OC4PreloadConfig()	void TIM_OC4PreloadConfig()	void TIM_OC4PreloadConfig()
void TIM1_OC1FastConfig()	void TIM_OC1FastConfig()	void TIM_OC1FastConfig()
void TIM1_OC2FastConfig()	void TIM_OC2FastConfig()	void TIM_OC2FastConfig()
void TIM1_OC3FastConfig()	void TIM_OC3FastConfig()	void TIM_OC3FastConfig()
void TIM1_OC4FastConfig()	void TIM_OC4FastConfig()	void TIM_OC4FastConfig()
void TIM1_ClearOC1Ref()	void TIM_ClearOC1Ref()	void TIM_ClearOC1Ref()
void TIM1_ClearOC2Ref()	void TIM_ClearOC2Ref()	void TIM_ClearOC2Ref()
void TIM1_ClearOC3Ref()	void TIM_ClearOC3Ref()	void TIM_ClearOC3Ref()
void TIM1_ClearOC4Ref()	void TIM_ClearOC4Ref()	void TIM_ClearOC4Ref()
void TIM1_GenerateEvent()	void TIM_GenerateEvent();	void TIM_GenerateEvent()
void TIM1_OC1PolarityConfig()	void TIM_OC1PolarityConfig()	void TIM_OC1PolarityConfig()
void TIM1_OC1NPolarityConfig()		void TIM_OC1NPolarityConfig()
void TIM1_OC2PolarityConfig()	void TIM_OC2PolarityConfig()	void TIM_OC2PolarityConfig()
void TIM1_OC2NPolarityConfig()		void TIM_OC2NPolarityConfig()
void TIM1_OC3PolarityConfig()	void TIM_OC3PolarityConfig()	void TIM_OC3PolarityConfig()
void TIM1_OC3NPolarityConfig()		void TIM_OC3NPolarityConfig()
void TIM1_OC4PolarityConfig()	void TIM_OC4PolarityConfig()	void TIM_OC4PolarityConfig()
void TIM1_CCxCmd()		void TIM_CCxCmd()
void TIM1_CCxNCmd()		void TIM_CCxNCmd()

Table 4. Changes made in the timer driver (continued)

STM32 FWLib V1.0 stm32f10x_tim1.c, .h functions	STM32 FWLib V1.0 stm32f10x_tim.c, .h functions	STM32 FWLib V2.0 stm32f10x_tim.c, .h functions
void TIM1_SelectOCxM()		void TIM_SelectOCxM()
void TIM1_SetCounter()	void TIM_SetCounter()	void TIM_SetCounter()
void TIM1_SetAutoreload()	void TIM_SetAutoreload()	void TIM_SetAutoreload()
void TIM1_SetCompare1()	void TIM_SetCompare1()	void TIM_SetCompare1()
void TIM1_SetCompare2()	void TIM_SetCompare2()	void TIM_SetCompare2()
void TIM1_SetCompare3()	void TIM_SetCompare3()	void TIM_SetCompare3()
void TIM1_SetCompare4()	void TIM_SetCompare4()	void TIM_SetCompare4()
void TIM1_SetIC1Prescaler()	void TIM_SetIC1Prescaler()	void TIM_SetIC1Prescaler()
void TIM1_SetIC2Prescaler()	void TIM_SetIC2Prescaler()	void TIM_SetIC2Prescaler()
void TIM1_SetIC3Prescaler()	void TIM_SetIC3Prescaler()	void TIM_SetIC3Prescaler()
void TIM1_SetIC4Prescaler()	void TIM_SetIC4Prescaler()	void TIM_SetIC4Prescaler()
void TIM1_SetClockDivision()	void TIM_SetClockDivision()	void TIM_SetClockDivision()
u16 TIM1_GetCapture1(void)s	u16 TIM_GetCapture1()	u16 TIM_GetCapture1(void)
u16 TIM1_GetCapture2(void)	u16 TIM_GetCapture2()	u16 TIM_GetCapture2(void)
u16 TIM1_GetCapture3(void)	u16 TIM_GetCapture3()	u16 TIM_GetCapture3(void)
u16 TIM1_GetCapture4(void)	u16 TIM_GetCapture4()	u16 TIM_GetCapture4(void)
u16 TIM1_GetCounter(void)	u16 TIM_GetCounter()	u16 TIM_GetCounter(void)
u16 TIM1_GetPrescaler(void)	u16 TIM_GetPrescaler()	u16 TIM_GetPrescaler(void)
FlagStatus TIM1_GetFlagStatus()	FlagStatus TIM_GetFlagStatus()	FlagStatus TIM_GetFlagStatus()
void TIM1_ClearFlag()	void TIM_ClearFlag()	void TIM_ClearFlag()
ITStatus TIM1_GetITStatus()	ITStatus TIM_GetITStatus()	ITStatus TIM_GetITStatus()
void TIM1_ClearITPendingBit()	void TIM_ClearITPendingBit()	void TIM_ClearITPendingBit()

1. Added for TIMx.
2. Available for TIM1 and TIM8.

Revision history

Table 5. Document revision history

Date	Revision	Changes
29-May-2008	1	Initial release.