AN3422
Application note

Migrating a microcontroller application from STM32F1 to STM32L1 series

1 Introduction

For designers of STM32 microcontroller applications, it is important to be able to easily replace one microcontroller type by another one in the same product family. Migrating an application to a different microcontroller is often needed, when product requirements grow, putting extra demands on memory size, or increasing the number of I/Os. On the other hand, cost reduction objectives may force you to switch to smaller components and shrink the PCB area.

This application note is written to help you and analyze the steps you need to migrate from an existing STM32F1 devices based design to STM32L1 devices. It groups together all the most important information and lists the vital aspects that you need to address.

To migrate your application from STM32 F1 series to L1 series, you have to analyze the hardware migration, the peripheral migration and the firmware migration.

To benefit fully from the information in this application note, the user should be familiar with the STM32 microcontroller family. You can refer to the following documents that are available from www.st.com.

- The STM32F1 family reference manuals (RM0008 and RM0041), the STM32F1 datasheets, and the STM32F1 Flash programming manuals (PM0075, PM0063 and PM0068).
- The STM32L1 family reference manual (RM0038), the STM32L1 datasheets, and the STM32F1 Flash and EEPROM programming manual (PM0062).

For an overview of the whole STM32 series and a comparison of the different features of each STM32 product series, please refer to AN3364 'Migration and compatibility guidelines for STM32 microcontroller applications

Contents

List of tables

List of figures

2 Hardware migration

The ultralow power STM32L and general-purpose STM32F1xxx families are pin-to-pin compatible. All peripherals shares the same pins in the two families, but there are some minor differences between packages.

In fact, the STM32 L1 series maintains a close compatibility with the whole STM32 F1 series. All power and functional pins are pin-to-pin compatible. The transition from the STM32 F1 series to the STM32 L1 series is simple as only a few pins are impacted (impacted pins are in bold in the table below).

Table 1. STM32 F1 series and STM32 L1 series pinout differences

STM32 F1 series				STM32 L1 series			
QFP48	**QFP64**	**QFP100**	**Pinout**	**QFP48**	**QFP64**	**QFP100**	**Pinout**
5	5	12	**PD0** - OSC_IN	5	5	12	**PH0** - OSC_IN
6	6	13	**PD1** - OSC_OUT	6	6	13	**PH1** - OSC_OUT
1	1	6	**VBAT**	1	1	6	**VLCD**
-	-	73	**NC**	-	-	73	**PH2**

The figures below show examples of board designs that are compatible with both the F1 and the L1 series.

Figure 1. Compatible board design: LQFP100

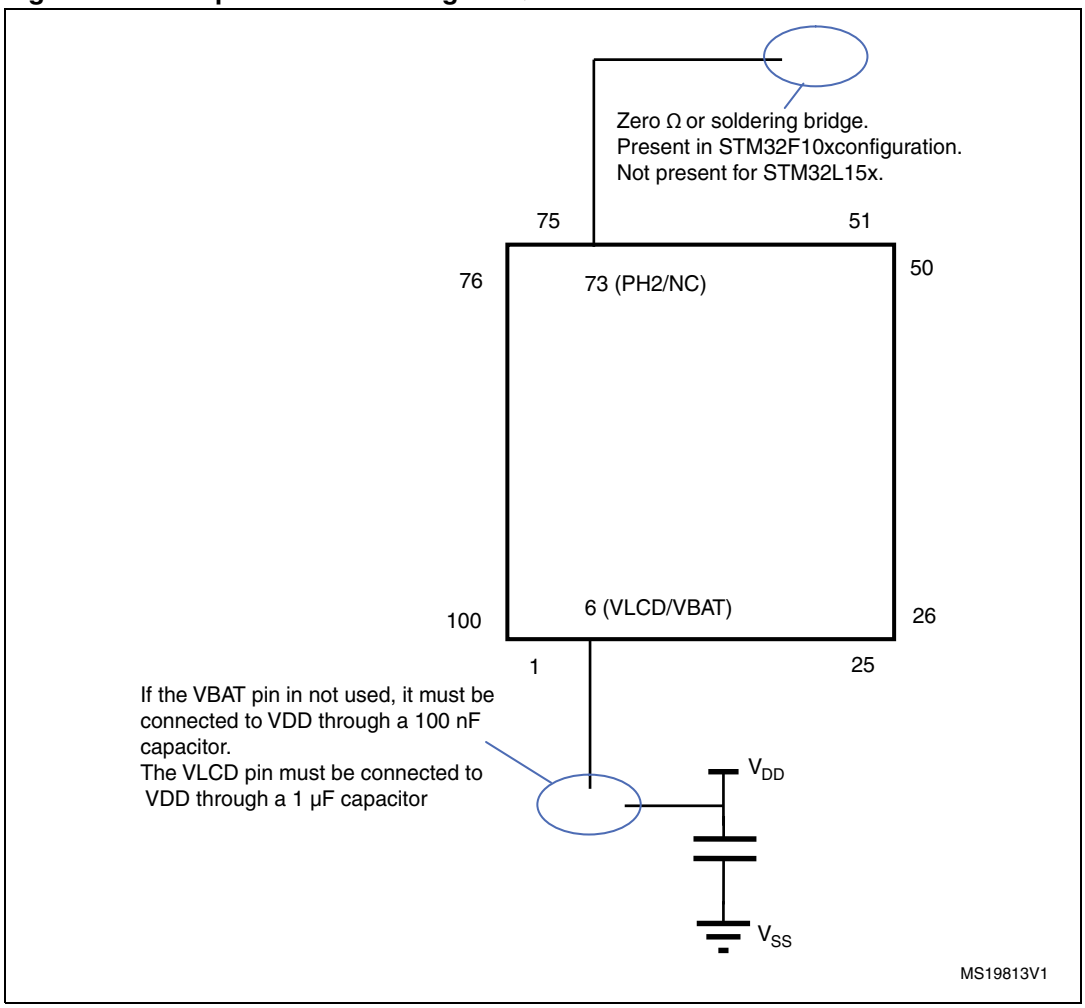


Figure 2. Compatible board design: LQFP64

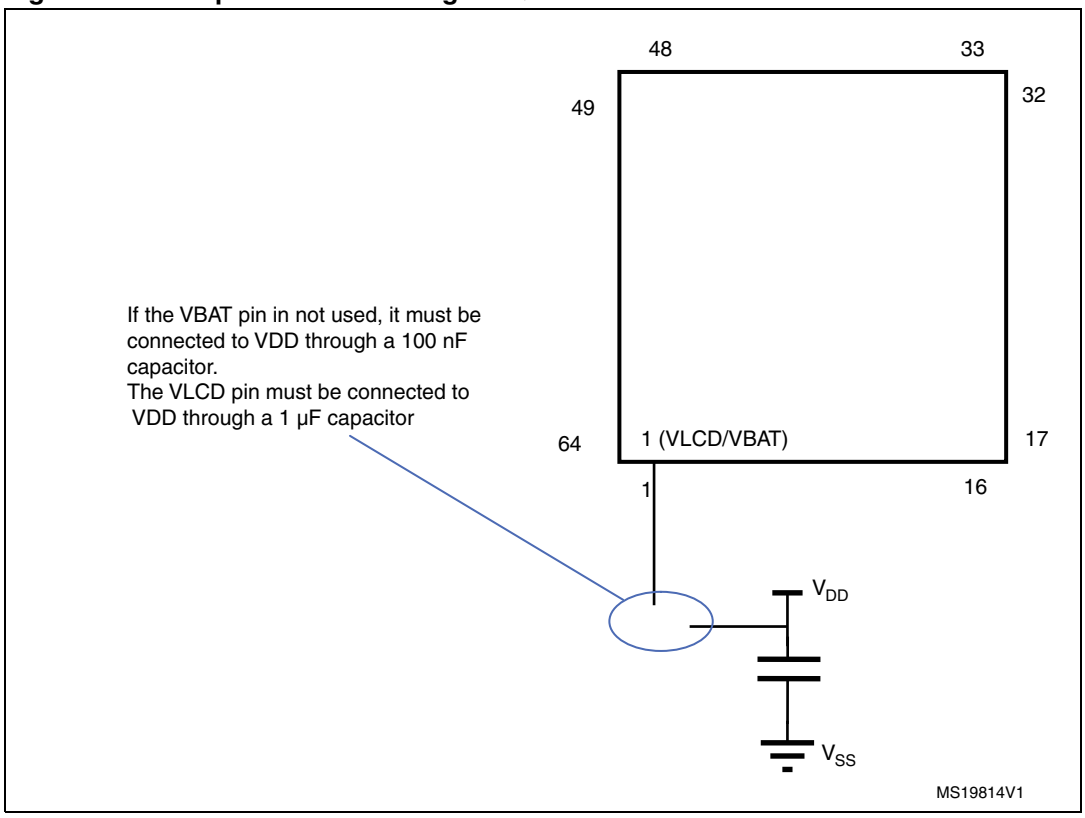
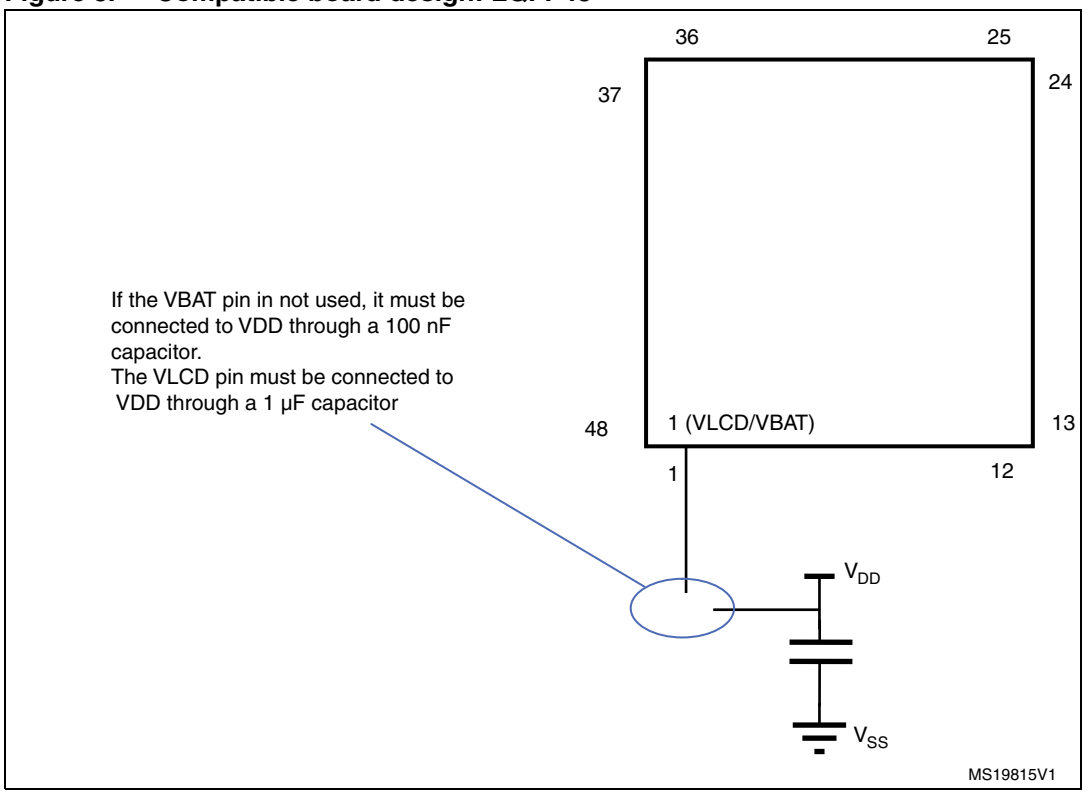


Figure 3. Compatible board design: LQFP48



3 Peripheral migration

As shown in *Table 2 on page 10*, there are three categories of peripherals. The common peripherals are supported with the dedicated firmware library without any modification, except if the peripheral instance is no longer present, you can change the instance and of course all the related features (clock configuration, pin configuration, interrupt/DMA request).

The modified peripherals such as: FLASH, ADC, RCC, PWR, GPIO and RTC are different from the F1 series ones and should be updated to take advantage of the enhancements and the new features in L1 series.

All these modified peripherals in the L1 series are enhanced to obtain lower power consumption, with features designed to meet new market requirements and to fix some limitations present in the F1 series.

3.1 STM32 product cross-compatibility

The STM32 series embeds a set of peripherals which can be classed in three categories:

- The first category is for the peripherals which are by definition common to all products. Those peripherals are identical, so they have the same structure, registers and control bits. There is no need to perform any firmware change to keep the same functionality at the application level after migration. All the features and behavior remain the same.
- The second category is for the peripherals which are shared by all products but have only minor differences (in general to support new features), so migration from one product to another is very easy and does not need any significant new development effort.
- The third category is for peripherals which have been considerably changed from one product to another (new architecture, new features...). For this category of peripherals, migration will require new development at application level.

Table 2 below gives a general overview of this classification:

Table 2. STM32 peripheral compatibility analysis F1 versus L1 series

Peripheral	F1 series	L1 series	Compatibility		
			Comments	Pinout	SW compatibility
SPI	Yes+	Yes	No I2S in L1 series L1 vs. F1: limitation fix	Identical	Full compatibility
WWDG	Yes	Yes	Same features	NA	Full compatibility
IWDG	Yes	Yes	Same features	NA	Full compatibility
DBGMCU	Yes	Yes	Same features	NA	Full compatibility
CRC	Yes	Yes	Same features	NA	Full compatibility
EXTI	Yes	Yes	Same features	Identical	Full compatibility
USB FS Device	Yes	Yes	Same features	Identical	Full compatibility
DMA	Yes	Yes	Same features	NA	Full compatibility
TIM	Yes	Yes	Same features	Identical	Full compatibility
PWR	Yes	Yes+	Enhancement	NA	Full compatibility for the same feature
RCC	Yes	Yes+	Enhancement	NA	Partial compatibility
USART	Yes	Yes+	Limitation fix / One Sample Bit method / Oversampling by 8	Identical	Full compatibility
I2C	Yes	Yes+	Limitation fix	Identical	Full compatibility
DAC	Yes	Yes+	DMA underrun interrupt	Identical	Full compatibility
ADC	Yes	Yes++	New peripheral	Identical	Partial compatibility
RTC	Yes	Yes++	New peripheral	Identical for the same feature	Not compatible
FLASH	Yes	Yes++	New peripheral	NA	Not compatible
GPIO	Yes	Yes++	New peripheral	Identical	Not compatible
CAN	Yes	NA	NA	NA	NA
CEC	Yes	NA	NA	NA	NA
Ethernet	Yes	NA	NA	NA	NA
SDIO	Yes	NA	NA	NA	NA
FSMC	Yes	NA	NA	NA	NA
LCD glass	NA	Yes	NA	NA	NA

Table 2. STM32 peripheral compatibility analysis F1 versus L1 series (continued)

Peripheral	F1 series	L1 series	Compatibility		
			Comments	Pinout	SW compatibility
COMP	NA	Yes	NA	NA	NA
SYSCFG	NA	Yes	NA	NA	NA

Color key:

= New feature or new architecture (Yes++)

= Same feature, but specification change or enhancement (Yes+)

= Feature not available (NA)

3.2 System architecture

The STM32L MCU family, based on the Cortex-M3 core, extends ST's ultra-low-power portfolio in performance, features, memory size and package pin count. It combines very high performance and ultra-low power consumption, through the use of an optimized architecture and ST's proprietary ultra-low leakage process, that is also used in the STM8L family. The STM32L family offers two different product lines (STM32L151 and STM32L152) so optimizing the STM32F and STM8L families for many applications requiring performance with special care on power savings.

3.3 Memory mapping

The peripheral address mapping has been changed in the L1 series vs. F1 series, the main change concerns the GPIOs which have been moved from the APB bus to the AHB bus to allow them to operate at maximum speed.

The tables below provide the peripheral address mapping correspondence between L1 and F1 series.

Table 3. IP bus mapping differences between STM32 F1 and STM32 L1 series

Peripheral	STM32 L1 series		STM32 F1 series	
	Bus	Base address	Bus	Base address
DMA1	AHB	0x40026000	AHB	0x40020000
Flash Interface		0x40023C00		0x40022000
RCC		0x40023800		0x40021000
CRC		0x40023000		0x40023000
GPIOH		0x40021400	NA	NA
GPIOE		0x40021000	APB2	0x40011800
GPIOD		0x40020C00		0x40011400
GPIOC		0x40020800		0x40011000
GPIOB		0x40020400		0x40010C00
GPIOA		0x40020000		0x40010800
USART1	APB2	0x40013800	APB2	0x40013800
SP1		0x40013000		0x40013000
ADC1		0x40012400		0x40012400
TIM11		0x40011000		0x40015400
TIM10		0x40010C00		0x40015000
TIM9		0x40010800		0x40014C00
EXTI		0x40010400		0x40010400
SYSCFG		0x40010000	NA	NA

Table 3. IP bus mapping differences between STM32 F1 and STM32 L1 series

Peripheral	STM32 L1 series		STM32 F1 series	
	Bus	Base address	Bus	Base address
COMP+RI	APB1	0x40007C00	NA	NA
DAC		0x40007400	APB1	0x40007400
PWR		0x40007000		0x40007000
USB device FS SRAM		0x40006000		0x40006000
USB device FS		0x40005C00		0x40005C00
I2C2		0x40005800		0x40005800
I2C1		0x40005400		0x40005400
USART3		0x40004800		0x40004800
USART2		0x40004400		0x40004400
SPI2		0x40003800		0x40003800
IWDG		0x40003000		0x40003000
WWDG		0x40002C00		0x40002C00
RTC		0x40002800 (inc. BKP registers)		0x40002800
LCD		0x40002400	NA	NA
TIM7		0x40001400	APB1	0x40001400
TIM6		0x40001000		0x40001000
TIM4		0x40000800		0x40000800
TIM3		0x40000400		0x40000400
TIM2		0x40000000		0x40000000
FSMC Registers	NA	NA	AHB	0xA0000000
USB OTG FS	NA	NA		0x50000000
ETHERNET MAC	NA	NA		0x40028000
DMA2	NA	NA		0x40020400
GPIOG	NA	NA	APB2	0x40012000
GPIOF	NA	NA		0x40011C00
SDIO	NA	NA	AHB	0x40018000
ADC2	NA	NA	APB2	0x40012800
ADC3	NA	NA		0x40013C00
TIM8	NA	NA		0x40013400
TIM1	NA	NA		0x40012C00

Table 3. IP bus mapping differences between STM32 F1 and STM32 L1 series

Peripheral	STM32 L1 series		STM32 F1 series	
	Bus	Base address	Bus	Base address
CAN2	NA	NA	APB1	0x40006800
CAN1	NA	NA		0x40006400
UART5	NA	NA		0x40005000
UART4	NA	NA		0x40004C00
SPI3/I2S3	NA	NA		0x40003C00
TIM14	NA	NA		0x40002000
TIM13	NA	NA		0x40001C00
TIM12	NA	NA		0x40001800
TIM5	NA	NA		0x40000C00
BKP registers	NA	NA		0x40006C00
AFIO	NA	NA	APB2	0x40010000

Color key:

= Same feature, but base address change

= Feature not available (NA)

3.4 RCC

The main differences related to the RCC (Reset and Clock Controller) in the STM32 L1 series vs. STM32 F1 series are presented in the table below.

Table 4. RCC differences between STM32 F1 and STM32 L1series

RCC main features	STM32 F1 series	STM32 L1series	Comments
MSI	NA	Multi Speed RC factory-trimmed (64 kHz /128 kHz / 256 kHz / 512 kHz / 1.02 MHz / 2.05 MHz / 4.1 MHz)	– Enable/disable RCC_CR[MSION] – Status flag RCC_CR[MSIRDY]
HSI	8 MHz RC factory-trimmed	16 MHz RC factory-trimmed	No change to SW configuration: – Enable/disable RCC_CR[HSION] – Status flag RCC_CR[HSIRDY]
LSI	40 KHz RC	37 KHz RC	No change to SW configuration: – Enable/disable RCC_CSR[LSION] – Status flag RCC_CSR[LSIRDY]

Table 4. RCC differences between STM32 F1 and STM32 L1series

RCC main features	STM32 F1 series	STM32 L1series	Comments
HSE	3 - 25 MHz Depending on the product line used	1 - 24 MHz	No change to SW configuration: – Enable/disable RCC_CR[HSEON] – Status flag RCC_CR[HSERDY]
LSE	32.768 kHz	32.768 kHz	LSE configuration/status bits are now in RCC_CSR register. – Enable/disable RCC_CSR[LSEON] – Status flag RCC_CSR[LSERDY] In L1 series the LSEON and LSERDY bits occupy bits RCC_CSR[9:8] respectively instead of bit RCC_BDCR[1:0] in F1 series.
PLL	– *Connectivity line:* main PLL + 2 PLLs for I2S, Ethernet and OTG FS clock – *Other product lines:* main PLL	– Main PLL for system	There is no change to PLL enable/disable RCC_CR[PLLON] and status flag RCC_CR[PLLRDY]. However, PLL configuration (clock source selection, multiplication/division factors) are different. In L1 series dedicated bits RCC_CFGR[PLLDIV] are used to configure the PLL divider parameters and the PLL multiplication factors are different. The PLL sources are only HSI and HSE.
System clock source	HSI, HSE or PLL	MSI, HSI, HSE or PLL	No change to SW configuration: – Selection bits RCC_CFGR[SW] – Status flag RCC_CFGR[SWS] However there is one more source, MSI, and the selection bit meanings are different.
System clock frequency	up to 72 MHz depending on the product line used 8 MHz after reset using HSI	32 MHz 2 MHz after reset using MSI	For STM32 L1 Flash wait states should be adapted according to the system frequency, the product voltage range V_{CORE} and the supply voltage range VDD.
AHB frequency	up to 72 MHz	up to 32 MHz	No change to SW configuration: configuration bits RCC_CFGR[HPRE]

Table 4. RCC differences between STM32 F1 and STM32 L1series

RCC main features	STM32 F1 series	STM32 L1series	Comments
APB1 frequency	up to 36 MHz	up to 32 MHz	No change to SW configuration: configuration bits RCC_CFGR[PPRE1].
APB2 frequency	up to 72 MHz	up to 32 MHz	No change to SW configuration: configuration bits RCC_CFGR[PPRE2].
RTC clock source	LSI, LSE or HSE/128	LSI, LSE or HSE clock divided by 2, 4, 8 or 16	RTC clock source configuration is done through the same bits (RTCSE[1:0] and RTCEN) but they are located in a different register. In L1 series the RTCSEL[1:0] bits occupy bits RCC_CSR[17:16] instead of bits RCC_BDCR[9:8] in F1 series. In L1 series the RTCEN bit occupies bit RCC_CSR[22] instead of bit RCC_BDCR[15] in F1 series. However, in L1 series when HSE is selected as RTC clock source, additional bits are used in CR register, RCC_CR[RTCPRE], to select the division factor to be applied to HSE clock.
MCO clock source	– MCO pin (PA8) – *Connectivity Line:* HSI, HSE, PLL/2, SYSCLK, PLL2, PLL3 or XT1 – *Other product lines:* HSI, HSE, PLL/2 or SYSCLK	– MCO pin (PA8): SYSCLK, HSI, HSE, PLLCLK, MSI, LSE or LSI With configurable prescaler, 1, 2, 4, 8 or 16 for each output.	MCO configuration in L1 series is different from F1: – For MCO, the prescaler is configured through bits RCC_CFGR[MCOPRE] and the selection of the clock to output through bits RCC_CFGR[MCOSEL]

Table 4. RCC differences between STM32 F1 and STM32 L1series

RCC main features	STM32 F1 series	STM32 L1series	Comments
Internal oscillator measurement / calibration	– LSI connected to TIM5 CH4 IC: can measure LSI w/ respect to HSI/HSE clock	– LSI connected to TIM10 CH1 IC: can measure LSI w/ respect to HSI/HSE clock – LSE connected to TIM10 CH1 IC: can measure LSE w/ respect to HSI/HSE clock – HSE connected to TIM11 CH1 IC: can measure HSE w/ respect to LSE/HSI clock – MSI connected to TIM11 CH1 IC: can measure MSI range w/ respect to HSI/HSE clock	There is no configuration to perform in RCC registers.
Interrupt	– CSS (linked to NMI IRQ) – LSIRDY, LSERDY, HSIRDY, HSERDY, PLLRDY, *PLL2RDY* and *PLL3RDY* (linked to RCC global IRQ)	– CSS (linked to IRQ) – LSIRDY, LSERDY, MSIRDY, HSIRDY, HSERDY and PLLRDY (linked to RCC global IRQ)	No change to SW configuration: interrupt enable, disable and pending bits clear are done in RCC_CIR register.

In addition to the differences described in the table above, the following additional adaptation steps may be needed for the migration:

1. *Performance versus V_{CORE} ranges:* The maximum system clock frequency and FLASH wait state depends on the selected voltage range V_{CORE} and also on V_{DD}. The following table gives the different clock source frequencies depending on the product voltage range.

Table 5. Performance versus V_{CORE} ranges

CPU performance	Power performance	V_{CORE} range	Typical Value (V)	Max frequency (MHz)		V_{DD} range
				1 WS	0 WS	
High	Low	1	1.8	32	16	2.0 - 3.6
Medium	Medium	2	1.5	16	8	1.65 - 3.6
Low	High	3	1.2	4	2	

2. *System clock configuration:* when moving from F1 series to L1 series only a few settings need to be updated in the system clock configuration code; mainly the Flash settings (configure the right wait states for the system frequency, prefetch

enable/disable, 64-bit access enable/disable...) or/and the PLL parameters configuration:

a) If the HSE or HSI is used directly as system clock source, in this case only the Flash parameters should be modified.

b) If PLL (clocked by HSE or HSI) is used as system clock source, in this case the Flash parameters and PLL configuration need to be updated.

Table 6 below provides an example of porting a system clock configuration from F1 to L1 series:

– STM32F105/7 Connectivity Line running at maximum performance: system clock at 72 MHz (PLL, clocked by the HSE, used as system clock source), Flash with 2 wait states and Flash prefetch queue enabled.

– L1 series running at maximum performance: system clock at 32 MHz (PLL, clocked by the HSE, used as system clock source), Flash with 1 wait state, Flash prefetch and 64-bit access enabled.

As shown in the table below, only the Flash settings and PLL parameters (code in ***Bold Italic***) need to be rewritten to run on L1 series. However, HSE, AHB prescaler and system clock source configuration are left unchanged, and APB prescalers are adapted to the maximum APB frequency in the L1 series.

Note: 1 The source code presented in the table below is intentionally simplified (time-out in wait loop removed) and is based on the assumption that the RCC and Flash registers are at their reset values.

2 For STM32L1xx you can use the clock configuration tool, STM32L1xx_Clock_Configuration.xls, to generate a customized system_stm32l1xx.c file containing a system clock configuration routine, depending on your application requirements. For more information, refer to AN3309 “Clock configuration tool for STM32L1xx microcontrollers”

Table 6. Example of migrating system clock configuration code from F1 to L1

STM32F105/7 running at 72 MHz (PLL as clock source) with 2 wait states	STM32L1xx running at 22 MHz (PLL as clock source) with 1 wait state

/* Enable HSE -----------------------------*/
RCC->CR |= ((uint32_t)RCC_CR_HSEON);
 	
/* Wait till HSE is ready */
while((RCC->CR & RCC_CR_HSERDY) == 0)
{
}
 	
/* Flash configuration -------------------*/
/* Prefetch ON, Flash 2 wait states */
FLASH->ACR |= FLASH_ACR_PRFTBE |
FLASH_ACR_LATENCY_2;
 	
/* AHB and APB prescaler configuration --*/
/* HCLK = SYSCLK */
RCC->CFGR |= RCC_CFGR_HPRE_DIV1;
 	
/* PCLK2 = HCLK */
RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;
 	
/* PCLK1 = HCLK */
RCC->CFGR |= RCC_CFGR_PPRE1_DIV2;
 	
/* PLL configuration -------------------*/
/* PLL2CLK = (HSE / 5) * 8 = 40 MHz
PREDIV1CLK = PLL2 / 5 = 8 MHz */
RCC->CFGR2 |= RCC_CFGR2_PREDIV2_DIV5 |
RCC_CFGR2_PLL2MUL8 |
RCC_CFGR2_PREDIV1SRC_PLL2 |
RCC_CFGR2_PREDIV1_DIV5;
/* Enable PLL2 */
RCC->CR |= RCC_CR_PLL2ON;
/* Wait till PLL2 is ready */
while((RCC->CR & RCC_CR_PLL2RDY) == 0)
{
}
/* PLLCLK = PREDIV1 * 9 = 72 MHz */
RCC->CFGR |= RCC_CFGR_PLLXTPRE_PREDIV1 |
RCC_CFGR_PLLSRC_PREDIV1 |
RCC_CFGR_PLLMULL9;
 	
/* Enable the main PLL */
RCC->CR |= RCC_CR_PLLON;
/* Wait till the main PLL is ready */
while((RCC->CR & RCC_CR_PLLRDY) == 0)
{
}
 	
/* Main PLL used as system clock source --*/
RCC->CFGR |= RCC_CFGR_SW_PLL;
/* Wait till the main PLL is used as system
clock source */
while ((RCC->CFGR & RCC_CFGR_SWS) !=
RCC_CFGR_SWS_PLL)
{
}
	/* Enable HSE -----------------------------*/
	RCC->CR |= ((uint32_t)RCC_CR_HSEON);
	
	/* Wait till HSE is ready */
	while((RCC->CR & RCC_CR_HSERDY) == 0)
	{
	}
	
	/* Flash configuration -------------------*/
	/* Flash prefetch and 64-bit access ON, Flash 1 wait state */
	FLASH->ACR |= FLASH_ACR_ACC64;
	
	FLASH->ACR |= FLASH_ACR_PRFTEN;
	
	FLASH->ACR |= FLASH_ACR_LATENCY;
	
	/* Power enable */
	RCC->APB1ENR |= RCC_APB1ENR_PWREN;
	
	/* Select the Voltage Range 1 (1.8 V) */
	PWR->CR = PWR_CR_VOS_0;
	
	/* Wait Until the Voltage Regulator is ready */
	while((PWR->CSR & PWR_CSR_VOSF) != RESET)
	{
	}
	
	/* AHB and APB prescaler configuration --*/
	/* HCLK = SYSCLK */
	RCC->CFGR |= RCC_CFGR_HPRE_DIV1;
	
	/* PCLK2 = HCLK / 1*/
	RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;
	
	/* PCLK1 = HCLK / 1*/
	RCC->CFGR |= RCC_CFGR_PPRE1_DIV1;
	
	/* PLL configuration ---------------------*/
	/* PLLCLK = (HSE * PLL_MUL) / PLL_DIV
	= (8 MHz * 12) / 3
	= 32MHz */
	RCC->CFGR = RCC_CFGR_PLLSRC_HSE | RCC_CFGR_PLLMUL12 | RCC_CFGR_PLLDIV3;
	
	/* Enable the main PLL */
	RCC->CR |= RCC_CR_PLLON;
	/* Wait till the main PLL is ready */
	while((RCC->CR & RCC_CR_PLLRDY) == 0)
	{
	}
	
	/* Main PLL used as system clock source --*/
	RCC->CFGR |= RCC_CFGR_SW_PLL;
	/* Wait till the main PLL is used as system
	clock source */
	while ((RCC->CFGR & RCC_CFGR_SWS) !=
	RCC_CFGR_SWS_PLL);
	{
	}

3. *Peripheral access configuration:* since the address mapping of some peripherals has been changed in L1 series vs. F1 series, you need to use different registers to [enable/disable] or [enter/exit] the peripheral [clock] or [from reset mode].

Table 7. RCC registers used for peripheral access configuration

Bus	Register	Comments
AHB	RCC_AHBRSTR	Used to [enter/exit] the AHB peripheral from reset
	RCC_AHBENR	Used to [enable/disable] the AHB peripheral clock
	RCC_AHBLPENR	Used to [enable/disable] the AHB peripheral clock in low power Sleep mode
APB1	RCC_APB1RSTR	Used to [enter/exit] the APB1 peripheral from reset
	RCC_APB1ENR	Used to [enable/disable] the APB1 peripheral clock
	RCC_APB1LPENR	Used to [enable/disable] the APB1 peripheral clock in low power Sleep mode
APB2	RCC_APB2RSTR	Used to [enter/exit] the APB2 peripheral from reset
	RCC_APB2ENR	Used to [enable/disable] the APB2 peripheral clock
	RCC_APB2LPENR	Used to [enable/disable] the APB2 peripheral clock in low power Sleep mode

To configure the access to a given peripheral you have first to know to which bus this peripheral is connected, refer to *Table 3 on page 12*, then depending on the action needed you have to program the right register as described in *Table 7* above. For example, USART1 is connected to APB2 bus, to enable the USART1 clock you have to configure APB2ENR register as follows:

```
RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
```

to disable USART1 clock during Sleep mode (to reduce power consumption) you have to configure APB2LPENR register as follows:

```
RCC->APB2LPENR |= RCC_APB2LPENR_USART1LPEN;
```

4. *Peripheral clock configuration*: some peripherals have a dedicated clock source independent from the system clock, and used to generate the clock required for their operation:

   a) *USB:* The USB 48 MHz clock is derived from the PLL VCO clock which should be at 9 6MHz

   b) LCD: The LCD Glass clock shares the same clock source as the RTC.

   c) *ADC:* in STM32 L1 series the ADC features two clock schemes:

   – Clock for the analog circuitry: ADCCLK. This clock is always the HSI oscillator clock. A divider by 1, 2 or 4 allows to adapt the clock frequency to the device operating conditions. This configuration is done using ADC_CCR[ADCPRE] bits. The ADC Clock depends also on the voltage range V_{CORE}. When product voltage

range 3 is selected (V_{CORE} = 1.2 V), the ADC is low speed (ADCCLK = 4 MHz, 250 Ksps).

– Clock for the digital interface (used for register read/write access). This clock is the APB2 clock. The digital interface clock can be enabled/disabled through the RCC_APB2ENR register (ADC1EN bit) and there is a bit to reset the ADC through RCC_APB2RSTR[ADCRST] bit.

3.5 DMA

STM32 F1 and STM32 L1 series uses the same DMA controller fully compatible.

STM32 F1 series embeds two DMA controllers, each controller has up to 7 channels. STM32 L1 series embeds only one DMA controller, with 7 channels. Each channel is dedicated to managing memory access requests from one or more peripherals. It has an arbiter for handling the priority between DMA requests.

The table below presents the correspondence between the DMA requests of the peripherals in STM32 F1 series and STM32 L1 series.

Table 8. DMA request differences between STM32 F1 series and STM32 L1 series

Peripheral	DMA request	STM32 F1 series	STM32 L1 series
ADC1	**ADC1**	DMA1_Channel1	DMA1_Channel1
ADC2	**ADC2**	NA	NA
ADC3	**ADC3**	DMA2_Channel5	NA
DAC	**DAC_Channel1** **DAC_Channel2**	DMA2_Channel3 / DMA1_Channel3(1) DMA2_Channel4 / DMA1_Channel4(1)	DMA1_Channel3 DMA1_Channel4
SPI1	**SPI1_Rx** **SPI1_Tx**	DMA1_Channel2 DMA1_Channel3	DMA1_Channel2 DMA1_Channel3
SPI2	**SPI2_Rx** **SPI2_Tx**	DMA1_Channel4 DMA1_Channel5	DMA1_Channel4 DMA1_Channel5
SPI3	**SPI3_Rx** **SPI3_Tx**	DMA2_Channel1 DMA2_Channel2	NA
USART1	**USART1_Rx** **USART1_Tx**	DMA1_Channel5 DMA1_Channel4	DMA1_Channel5 DMA1_Channel4
USART2	**USART2_Rx** **USART2_Tx**	DMA1_Channel6 DMA1_Channel7	DMA1_Channel6 DMA1_Channel7
USART3	**USART3_Rx** **USART3_Tx**	DMA1_Channel3 DMA1_Channel2	DMA1_Channel3 DMA1_Channel2
UART4	**UART4_Rx** **UART4_Tx**	DMA2_Channel3 DMA2_Channel5	NA
UART5	**UART5_Rx** **UART5_Tx**	DMA2_Channel4 DMA2_Channel1	NA

Table 8. DMA request differences between STM32 F1 series and STM32 L1 series (continued)

Peripheral	DMA request	STM32 F1 series	STM32 L1 series
I2C1	**I2C1_Rx** **I2C1_Tx**	DMA1_Channel7 DMA1_Channel6	DMA1_Channel7 DMA1_Channel6
I2C2	**I2C2_Rx** **I2C2_Tx**	DMA1_Channel5 DMA1_Channel4	DMA1_Channel5 DMA1_Channel4
SDIO	**SDIO**	DMA2_Channel4	NA
TIM1	**TIM1_UP** **TIM1_CH1** **TIM1_CH2** **TIM1_CH3** **TIM1_CH4** **TIM1_TRIG** **TIM1_COM**	DMA1_Channel5 DMA1_Channel2 DMA1_Channel3 DMA1_Channel6 DMA1_Channel4 DMA1_Channel4 DMA1_Channel4	NA
TIM8	**TIM8_UP** **TIM8_CH1** **TIM8_CH2** **TIM8_CH3** **TIM8_CH4** **TIM8_TRIG** **TIM8_COM**	DMA2_Channel1 DMA2_Channel3 DMA2_Channel5 DMA2_Channel1 DMA2_Channel2 DMA2_Channel2 DMA2_Channel2	NA
TIM2	**TIM2_UP** **TIM2_CH1** **TIM2_CH2** **TIM2_CH3** **TIM2_CH4**	DMA1_Channel2 DMA1_Channel5 DMA1_Channel7 DMA1_Channel1 DMA1_Channel7	DMA1_Channel2 DMA1_Channel5 DMA1_Channel7 DMA1_Channel1 DMA1_Channel7
TIM3	**TIM3_UP** **TIM3_CH1** **TIM3_TRIG** **TIM3_CH3** **TIM3_CH4**	DMA1_Channel3 DMA1_Channel6 DMA1_Channel6 DMA1_Channel2 DMA1_Channel3	DMA1_Channel3 DMA1_Channel6 DMA1_Channel6 DMA1_Channel2 DMA1_Channel3
TIM4	**TIM4_UP** **TIM4_CH1** **TIM4_CH2** **TIM4_CH3**	DMA1_Channel7 DMA1_Channel1 DMA1_Channel4 DMA1_Channel5	DMA1_Channel7 DMA1_Channel1 DMA1_Channel4 DMA1_Channel5
TIM5	**TIM5_UP** **TIM5_CH1** **TIM5_CH2** **TIM5_CH3** **TIM5_CH4** **TIM5_TRIG**	DMA2_Channel2 DMA2_Channel5 DMA2_Channel4 DMA2_Channel2 DMA2_Channel1 DMA2_Channel1	NA
TIM6	**TIM6_UP**	DMA2_Channel3 / DMA1_Channel3(1)	DMA1_Channel3
TIM7	**TIM7_UP**	DMA2_Channe4 / DMA1_Channel4 (1)	DMA1_Channel4

Table 8. DMA request differences between STM32 F1 series and STM32 L1 series (continued)

Peripheral	DMA request	STM32 F1 series	STM32 L1 series
TIM15	**TIM15_UP** **TIM15_CH1** **TIM15_TRIG** **TIM15_COM**	DMA1_Channel5 DMA1_Channel5 DMA1_Channel5 DMA1_Channel5	NA
TIM16	**TIM16_UP** **TIM16_CH1**	DMA1_Channel6 DMA1_Channel6	NA
TIM17	**TIM17_UP** **TIM17_CH1**	DMA1_Channel7 DMA1_Channel7	NA

1. For High-density value line devices, the DAC DMA requests are mapped respectively on DMA1 Channel 3 and DMA1 Channel 4

3.6 Interrupts

The table below presents the interrupt vectors in STM32 L1 series vs. F1 series

The changes in the interrupt vectors impact only a few peripherals:

1. ADC: in the F1 series there are two interrupt vectors for the ADCs; ADC1_2 and ADC3. However in L1 series there is a single interrupt vector for ADC1; ADC1_IRQ.
2. As in STM32 L1 series there are no CAN or TIM1 peripherals, their corresponding IRQs are now mapped to new peripherals: COMP, DAC, TIM9, TIM10, TIM11 and LCD.

Table 9. Interrupt vector differences between STM32 F1 series and STM32 L1 series

Position	STM32 F1 series	STM32 L1 series
0	WWDG	WWDG
1	PVD	PVD
2	TAMPER	TAMPER_ STAMP
3	RTC	RTC_WKUP
4	FLASH	FLASH
5	RCC	RCC
6	EXTI0	EXTI0
7	EXTI1	EXTI1
8	EXTI2	EXTI2
9	EXTI3	EXTI3
10	EXTI4	EXTI4
11	DMA1_Channel1	DMA1_Channel1
12	DMA1_Channel2	DMA1_Channel2
13	DMA1_Channel3	DMA1_Channel3
14	DMA1_Channel4	DMA1_Channel4
15	DMA1_Channel5	DMA1_Channel5

Table 9. Interrupt vector differences between STM32 F1 series and STM32 L1 series

Position	STM32 F1 series	STM32 L1 series
16	DMA1_Channel6	DMA1_Channel6
17	DMA1_Channel7	DMA1_Channel7
18	ADC1_2	ADC1
19	CAN1_TX / USB_HP_CAN_TX ((1)	USB_HP
20	CAN1_RX0 / USB_LP_CAN_RX0 (1)	USB_LP
21	CAN1_RX1	DAC
22	CAN1_SCE	COMP
23	EXTI9_5	EXTI9_5
24	TIM1_BRK / TIM1_BRK _TIM9 (1)	TIM9
25	TIM1_UP / TIM1_UP_TIM10 (1)	TIM10
26	TIM1_TRG_COM / TIM1_TRG_COM_TIM11 (1)	TIM11
27	TIM1_CC	LCD
28	TIM2	TIM2
29	TIM3	TIM3
30	TIM4	TIM4
31	I2C1_EV	I2C1_EV
32	I2C1_ER	I2C1_ER
33	I2C2_EV	I2C2_EV
34	I2C2_ER	I2C2_ER
35	SPI1	SPI1
36	SPI2	SPI2
37	USART1	USART1
38	USART2	USART2
39	USART3	USART3
40	EXTI15_10	EXTI15_10
41	RTC_Alarm	RTC_Alarm
42	OTG_FS_WKUP / USBWakeUp	USB_FS_WKUP
43	TIM8_BRK / TIM8_BRK_TIM12 (1)	TIM6
44	TIM8_UP / TIM8_UP_TIM13 (1)	TIM7
45	TIM8_TRG_COM / TIM8_TRG_COM_TIM14 (1)	NA
46	TIM8_CC	NA
47	ADC3	NA
48	FSMC	NA

Table 9. Interrupt vector differences between STM32 F1 series and STM32 L1 series

Position	STM32 F1 series	STM32 L1 series
49	SDIO	NA
50	TIM5	NA
51	SPI3	NA
52	UART4	NA
53	UART5	NA
54	TIM6 / TIM6_DAC (1)	NA, moved to 43
55	TIM7	NA, moved to 44
56	DMA2_Channel1	NA
57	DMA2_Channel2	NA
58	DMA2_Channel3	NA
59	DMA2_Channel4 / DMA2_Channel4_5(1)	NA
60	DMA2_Channel5	NA
61	ETH	NA
62	ETH_WKUP	NA
63	CAN2_TX	NA
64	CAN2_RX0	NA
65	CAN2_RX1	NA
66	CAN2_SCE	NA
67	OTG_FS	NA

Color key:

= Different Interrupt vector

= Interrupt Vector name changed but F1 peripheral still mapped on the same Interrupt Vector position in L1 series

= Feature not available (NA)

1. Depending on the product line used.

3.7 GPIO

The STM32 L1 GPIO peripheral embeds new features compared to F1 series, below the main features:

- GPIO mapped on AHB bus for better performance
- I/O pin multiplexer and mapping: pins are connected to on-chip peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected

to an I/O pin at a time. In this way, there can be no conflict between peripherals sharing the same I/O pin.

- More possibilities and features for I/O configuration

The L1 GPIO peripheral is a new design and thus the architecture, features and registers are different from the GPIO peripheral in the F1 series, so any code written for the F1 series using the GPIO needs to be rewritten to run on L1 series.

For more information about STM32 L1's GPIO programming and usage, please refer to the "I/O pin multiplexer and mapping" section in the GPIO chapter of the STM32L1xx Reference Manual (RM0038).

The table below presents the differences between GPIOs in the STM32 F1 series and STM32 L1 series.

Table 10. GPIO differences between STM32 F1 series and STM32 L1 series

GPIO	STM32 F1 series	STM32 L1 series
Input mode	Floating PU PD	Floating PU PD
General purpose output	PP OD	PP PP + PU PP + PD OD OD + PU OD + PD
Alternate Function output	PP OD	PP PP + PU PP + PD OD OD + PU OD + PD
Input / Output	Analog	Analog
Output speed	2 MHz 10 MHz 50 MHz	400KHz 2 MHz 10 MHz 40 MHz
Alternate function selection	To optimize the number of peripheral I/O functions for different device packages, it is possible to remap some alternate functions to some other pins (software remap).	Highly flexible pin multiplexing allows no conflict between peripherals sharing the same I/O pin.
Max IO toggle frequency	18 MHz	16 MHz

Alternate function mode

In STM32 F1 series

1. The configuration to use an I/O as alternate function depends on the peripheral mode used, for example the USART Tx pin should be configured as alternate function push-pull while USART Rx pin should be configured as input floating or input pull-up.
2. To optimize the number of peripheral I/O functions for different device packages (especially with those with low pin count), it is possible to remap some alternate functions to other pins by software, for example the USART2_RX pin can be mapped on PA3 (default remap) or PD6 (by software remap).

In STM32 L1 series

1. Whatever the peripheral mode used, the I/O must be configured as alternate function, then the system can use the I/O in the proper way (input or output).
2. The I/O pins are connected to on-chip peripherals/modules through a multiplexer that allows only one peripheral's alternate function to be connected to an I/O pin at a time. In this way, there can be no conflict between peripherals sharing the same I/O pin. Each I/O pin has a multiplexer with sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIOx_AFRL and GPIOx_AFRH registers:
   - After reset all I/Os are connected to the system's alternate function 0 (AF0)
   - The peripheral alternate functions are mapped by configuring AF1 to AF13
   - Cortex-M3 EVENTOUT is mapped by configuring AF15
3. In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped on different I/O pins to optimize the number of peripheral I/O functions for different device packages, for example the USART2_RX pin can be mapped on PA3 or PD6 pin

Note: Please refer to the "Alternate function mapping" table in the STM32L15x datasheet for the detailed mapping of the system and the peripheral alternate function I/O pins.

4. Configuration procedure
   - Configure the desired I/O as an alternate function in the GPIOx_MODER register
   - Select the type, pull-up/pull-down and output speed via the GPIOx_OTYPER, GPIOx_PUPDR and GPIOx_OSPEEDER registers, respectively
   - Connect the I/O to the desired AFx in the GPIOx_AFRL or GPIOx_AFRH register

3.8 EXTI source selection

In STM32 F1 the selection of EXTI line source is performed through EXTIx bits in AFIO_EXTICRx registers, while in L1 series this selection is done through EXTIx bits in SYSCFG_EXTICRx registers.

Only the mapping of the EXTICRx registers has been changed, without any changes to the meaning of the EXTIx bits. However, the maximum range of EXTIx bits values is 0b0101 as only 6 GPIO ports are supported in L1 (in F1 series the maximum value is 0b0110).

3.9 FLASH

The table below presents the difference between the FLASH interface of STM32 F1 series and STM32 L1 series, which can be grouped as follows:

- New interface, new technology
- New architecture,
- New read protection mechanism, 3 read protection levels with JTAG fuse

Consequently the L1 Flash programming procedures and registers are different from the F1 series, and any code written for the Flash interface in the F1 series needs to be rewritten to run on L1 series.

For more information on programming, erasing and protection of the L1 Flash memory, please refer to the STM32L1xx Flash programming manual (PM0062).

Table 11. FLASH differences between STM32 F1 series and STM32 L1 series

Feature		STM32 F1 series	STM32 L1 series
Wait State		up to 2	up to 1 (depending on the supply voltage)
Main/Program memory	**Start Address**	0x0800 0000	0x0800 0000
	End Address	up to 0x080F FFFF	up to 0x0801 FFFF
	Granularity	Page size = 2 Kbytes except for Low and Medium density page size = 1 Kbytes	Sector size = 4 Kbytes: 16 Pages of 256 bytes
EEPROM memory	**Start Address**	Available through SW emulation (1)	0x0808 0000
	End Address		0x0808 0FFF
System memory	**Start Address**	0x1FFF F000	0x1FF0 0000
	End Address	0x1FFF F7FF	0x1FF0 0FFF
Option Bytes	**Start Address**	0x1FFF F800	0x1FF8 0000
	End Address	0x1FFF F80F	0x1FF8001F
OTP	**Start Address**	NA	NA
	End Address		
Flash interface	**Start address**	0x4002 2000	0x4002 3C00
	Programming procedure	Same for all product lines	Different from F1 series
Erase granularity		Page (1 or 2 Kbytes)	Program memory: Page (256 bytes) DATA EEPROM memory: word / double word

Table 11. FLASH differences between STM32 F1 series and STM32 L1 series

Feature		STM32 F1 series	STM32 L1 series
Program mode		Half word	Program memory: word/ half page DATA EEPROM memory: byte / half word / word / Double word
Read Protection	**Unprotection**	Read protection disable RDP = 0xA55A	Level 0 no protection RDP = 0xAA
	Protection	Read protection enable RDP != 0xA55A	Level 1 memory protection RDP != (Level 2 & Level 0)
	JTAG fuse	NA	Level 2 RDP = 0xCC (2)
Write protection granularity		Protection by 4 Kbyte block	Protection by sector
User Option bytes		STOP	STOP
		STANDBY	STANDBY
		WDG	WDG
		NA	BOR level

Color key:

= New feature or new architecture

= Same feature, but specification change or enhancement

= Feature not available (NA)

1. For more details refer to “EEPROM emulation in STM32F10x microcontrollers (AN2594)
2. Memory read protection Level 2 is an irreversible operation. When Level 2 is activated, the level of protection cannot be decreased to Level 0 or Level 1.

3.10 ADC

The table below presents the differences between the ADC interface of STM32 F1 series and STM32 L1 series, these differences are the following:

- New digital interface
- New architecture and new features

Table 12. ADC differences between STM32 F1 series and STM32 L1 series

ADC	STM32 F1 series	STM32 L1series
ADC Type	SAR structure	SAR structure
Instances	ADC1 / ADC2 / ADC3	ADC1
Max Sampling freq	1 MSPS	1 MSPS
Number of channels	up to 21 channels	up to 24 Channels

Table 12. ADC differences between STM32 F1 series and STM32 L1 series (continued)

ADC	STM32 F1 series		STM32 L1series	
Resolution	12-bit		12-bit	
Conversion Modes	Single / continuous / Scan / Discontinuous / Dual Mode		Single / continuous / Scan / Discontinuous	
DMA	Yes		Yes	
External Trigger	Yes		Yes	
	External event for regular group For ADC1 and ADC2: TIM1 CC1 TIM1 CC2 TIM1 CC3 TIM2 CC2 TIM3 TRGO TIM4 CC4 EXTI line 11 / TIM8_TRGO For ADC3: TIM3 CC1 TIM2 CC3 TIM1 CC3 TIM8 CC1 TIM8 TRGO TIM5 CC1 TIM5 CC3	External event for injected group For ADC1 and ADC2: TIM1 TRGO TIM1 CC4 TIM2 TRGO TIM2 CC1 TIM3 CC4 TIM4 TRGO EXTI line15 / TIM8_CC4 For ADC3: TIM1 TRGO TIM1 CC4 TIM4 CC3 TIM8 CC2 TIM8 CC4 TIM5 TRGO TIM5 CC4	External event for regular group TIM9_CC2 TIM9_TRGO TIM2_CC3 TIM2_CC2 TIM3_TRGO TIM4_CC4 TIM2_TRGO TIM3_CC1 TIM3_CC3 TIM4_TRGO TIM6_TRGO EXTI line11	External event for injected group TIM9_CC1 TIM9_TRGO TIM2_TRGO TIM2_CC1 TIM3_CC4 TIM4_TRGO TIM4_CC1 TIM4_CC2 TIM4_CC3 TIM10_CC1 TIM7_TRGO EXTI line15
Supply requirement	2.4 V to 3.6 V		1.8 V to 3.6 V	
Input range	$V_{REF-} <= V_{IN} <= V_{REF+}$		Vref- <= Vin <= Vref+	

Color key:

= Same feature, but specification change or enhancement

3.11 PWR

In STM32 L1 series the PWR controller presents some differences vs. F1 series, these differences are summarized in the table below. However, the programming interface is unchanged.

Table 13. PWR differences between STM32 F1 series and STM32 L1 series

PWR	STM32 F1 series	STM32 L1 series
Power supplies	1. V_{DD} = 2.0 to 3.6 V: external power supply for I/Os and the internal regulator. Provided externally through V_{DD} pins. 2. V_{SSA}, V_{DDA} = 2.0 to 3.6 V: external analog power supplies for ADC, DAC, Reset blocks, RCs and PLL (minimum voltage to be applied to V_{DDA} is 2.4 V when the ADC or DAC is used). V_{DDA} and V_{SSA} must be connected to V_{DD} and V_{SS}, respectively. 3. V_{BAT} = 1.8 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when VDD is not present.	1. V_{DD} = 1.8 V (at power on) or 1.65 V (at power down) to 3.6 V when the BOR is available. V_{DD} = 1.65 V to 3.6 V, when BOR is not available. V_{DD} is the external power supply for I/Os and internal regulator. It is provided externally through V_{DD} pins. 2. V_{CORE} = 1.2 to 1.8 V V_{CORE} is the power supply for digital peripherals, SRAM and Flash memory. It is generated by a internal voltage regulator. Three V_{CORE} ranges can be selected by software depending on VDD. 3. V_{SSA}, V_{DDA} = 1.8 V (at power on) or 1.65 V (at power down) to 3.6 V, when BOR is available and V_{SSA}, V_{DDA} = 1.65 to 3.6 V, when BOR is not available. V_{DDA} is the external analog power supply for ADC, DAC, reset blocks, RC oscillators and PLL. The minimum voltage to be applied to V_{DDA} is 1.8 V when the ADC is used. 4. V_{LCD} = 2.5 to 3.6 V The LCD controller can be powered either externally through the V_{LCD} pin, or internally from an internal voltage generated by the embedded step-up converter.
Battery backup domain	– Backup registers – RTC – LSE – PC13 to PC15 I/Os **Note:** in F1 series the Backup registers are integrated in the BKP peripheral.	NA
RTC domain	NA	– RTC w/ backup registers – LSE – PC13 to PC15 I/Os **Note:** in L1 series the backup registers are integrated in the RTC peripheral
Power supply supervisor	Integrated POR / PDR circuitry Programmable Voltage Detector (PVD)	Integrated POR / PDR circuitry Programmable voltage detector (PVD)
	NA	Brownout reset (BOR)

Table 13. PWR differences between STM32 F1 series and STM32 L1 series

PWR	STM32 F1 series	STM32 L1 series
Low-power modes	Sleep mode Stop mode Standby mode (1.8V domain powered-off)	RUN Low Power Sleep mode + peripherals automatic clock gating Sleep Low Power mode + peripherals automatic clock gating Stop mode Standby mode (V_{CORE} domain powered off) **Note:** To further reduce power consumption in Sleep mode the peripheral clocks can be disabled prior to executing the WFI or WFE instructions.
Wake-up sources	Sleep mode – Any peripheral interrupt/wakeup event Stop mode – Any EXTI line event/interrupt Standby mode – WKUP pin rising edge – RTC alarm – External reset in NRST pin – IWDG reset	Sleep mode – Any peripheral interrupt/wakeup event Stop mode – Any EXTI line event/interrupt Standby mode – WKUP pin rising edge – RTC alarm A, RTC alarm B, RTC Wakeup, Tamper event, TimeStamp event – External reset in NRST pin – IWDG reset
Configuration	NA	In L1 some additional bits were added: – PWR_CR[ULP] used to switch off the VREFINT in STOP and STANDBY modes. – PWR_CR[FWU] used to ignore the VREFINT startup time when exiting from low power modes. – PWR_CR[VOS] used to select the product voltage range. – PWR_CR[LPRUN] used to select the RUN low power mode. – PWR_CSR[VREFINTRDY]: VREFTINT ready status – PWR_CSR[VOSF]: Internal Regulator change status – PWR_CSR[REGLP]: MCU is in Low power run mode – PWR_CSR[EWUP2] and PWR_CSR[EWUP3]: Wakeup Pin 2 and Wakeup Pin 3 Enable/Disable bits.

Color key:

= New feature or new architecture

= Same feature, but specification change or enhancement

3.12 RTC

The STM32 L1 series embeds a new RTC peripheral vs. F1 series; the architecture, features and programming interface are different.

As consequence the L1 RTC programming procedures and registers are different from the the F1 series, so any code written for the F1series using the RTC needs to be rewritten to run on L1 series.

The L1 RTC provides best-in-class features:

- BCD timer/counter
- Time-of-day clock/calendar with programmable daylight saving compensation
- Two programmable alarm interrupts
- Digital calibration circuit
- Time-stamp function for event saving
- Periodic programmable wakeup flag with interrupt capability
- Automatic wakeup unit to manage low power modes
- 20 backup registers (80 bytes) which are reset when a tamper detection event occurs.

For more information about STM32 L1's RTC features, please refer to RTC chapter of STM32L1xx Reference Manual (RM0038).

For advanced information about the RTC programming, please refer to Application Note AN3371 *Using the STM32 HW real-time clock (RTC)* .

4 Firmware migration using the library

This section describes how to migrate an application based on STM32F1xx Standard Peripherals Library in order to use the STM32L1xx Standard Peripherals Library.

The STM32F1xx and STM32L1xx libraries have the same architecture and are CMSIS compliant, they use the same driver naming and the same APIs for all compatible peripheral.

Only a few peripheral drivers need to be updated to migrate the application from an F1 series to an L1 series product.

Note: In the rest of this chapter (unless otherwise specified), the term "STM32L1xx Library" is used to refer to the STM32L1xx Standard Peripherals Library and the term of "STM32F10x Library" is used to refer to the STM32F10x Standard Peripherals Library.

4.1 Migration steps

To update your application code to run on STM32L1xx Library, you have to follow the steps listed below:

5. 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 STM32L1xxx devices. For more information please refer to your toolchain documentation.
   b) *Linker configuration and vector table location files:* these files are developed following the CMSIS standard and are included in the STM32L1xx Library install package under the following directory: *Libraries\CMSIS\CM3\DeviceSupport\ST\STM32L1xx*
6. Add STM32L1xx Library source files to the application sources
   a) Replace the *stm32f10x_conf.h* file of your application with *stm32l1xx_conf.h* provided in STM32L1xx Library.
   b) Replace the existing *stm32f10x_it.c/stm32f10x_it.h files* in your application with *stm32l1xx_it.c/stm32l1xx_it.h* provided in STM32L1xx Library.
7. Update the part of your application code that uses the RCC, PWR, GPIO, FLASH, ADC and RTC drivers. Further details are provided in the next section.

Note: The STM32L1xx Library comes with a rich set of examples (72 in total) demonstrating how to use the different peripherals (under Project\STM32L1xx_StdPeriph_Examples\).

4.2 RCC

1. *System clock configuration:* as presented in section *3.4: RCC* the STM32 L1 and F1 series have the same clock sources and configuration procedures. However, there are some differences related to the product voltage range, PLL configuration, maximum frequency and Flash wait state configuration. Thanks to the CMSIS layer, these differences are hidden from the application code; you only have to replace the *system_stm32f10x.c* file by *system_stm32l1xx.c* file. This file provides an

implementation of *SystemInit()* function used to configure the microcontroller system at start-up and before branching to the main() program.

Note: For STM32L1xx you can use the clock configuration tool, STM32L1xx_Clock_Configuration.xls, to generate a customized SystemInit() function *depending on your application requirements. For more information, refer to AN3309 "Clock configuration tool for STM32L1xx microcontrollers"*

2. *Peripheral access configuration:* as presented in section *3.4: RCC* you need to call different functions to [enable/disable] or [enter/exit] the peripheral [clock] or [from reset mode]. For example, GPIOA is mapped on AHB bus on L1 series (APB2 bus on F1 series), to enable its clock you have to use the
`RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);`
function instead of:
`RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);`
in the F1 series. Refer to *Table 3 on page 12* for the peripheral bus mapping changes between L1 and F1 series.

3. *Peripheral clock configuration*

a) *USB FS Device:* in STM32 L1 series the USB FS Device require a frequency of 48 MHz to work correctly. The following is an example of the main PLL configuration to obtain 32 MHz as system clock frequency and 48 MHz for the USB FS Device.

```
/* PLL_VCO = HSE_VALUE * PLL_MUL = 96 MHz */
/* USBCLK = PLL_VCO / 2= 48 MHz */
/* SYSCLK = PLL_VCO * PLL_DIV = 32 MHz */
RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLSRC_HSE | RCC_CFGR_PLLMUL12 | RCC_CFGR_PLLDIV3);
/* Enable PLL */
RCC->CR |= RCC_CR_PLLON;

/* Wait till PLL is ready */
while((RCC->CR & RCC_CR_PLLRDY) == 0)
{
}

/* Select PLL as system clock source */
RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;

/* Wait till PLL is used as system clock source */
while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)RCC_CFGR_SWS_PLL)
{
}
  ...
/* Enable USB FS Device's APB1 interface clock */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE);
```

b) *ADC:* in STM32 L1 series the ADC features two clock schemes:

– Clock for the analog circuitry: ADCCLK. This clock is generated always from the HSI clock divided by a programmable prescaler that allows the ADC to work at f_{HSI}/1, /2 or /4. This configuration is done using the ADC registers.

– Clock for the digital interface (used for register read/write access). This clock is equal to the APB2 clock. The digital interface clock can be enabled/disabled through the RCC APB2 peripheral clock enable register (RCC_APB2ENR).

```
/* Enable the HSI oscillator */
```

```
RCC_HSICmd(ENABLE);

/* Check that HSI oscillator is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
{
}
/* Enable ADC1 clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
```

4.3 FLASH

The table below presents the FLASH driver API correspondence between STM32F10x and STM32L1xx Libraries. You can easily update your application code by replacing STM32F10x functions by the corresponding function in STM32L1xx Library.

Table 14. STM32F10x and STM32L1xx FLASH driver API correspondence

	STM32F10x Flash driver API	STM32L1xx Flash driver API
Interface configuration	void FLASH_SetLatency(uint32_t FLASH_Latency);	void FLASH_SetLatency(uint32_t FLASH_Latency);
	void FLASH_PrefetchBufferCmd(uint32_t FLASH_PrefetchBuffer);	void FLASH_PrefetchBufferCmd(FunctionalState NewState);
	void FLASH_HalfCycleAccessCmd(uint32_t FLASH_HalfCycleAccess);	NA
	NA	void FLASH_ReadAccess64Cmd(FunctionalState NewState);
	NA	void FLASH_RUNPowerDownCmd(FunctionalState NewState);
	NA	void FLASH_SLEEPPowerDownCmd(FunctionalState NewState);
	void FLASH_ITConfig(uint32_t FLASH_IT, FunctionalState NewState);	void FLASH_ITConfig(uint32_t FLASH_IT, FunctionalState NewState);

Table 14. STM32F10x and STM32L1xx FLASH driver API correspondence (continued)

	STM32F10x Flash driver API	STM32L1xx Flash driver API
Memory Programming	void FLASH_Unlock(void);	void FLASH_Unlock(void);
	void FLASH_Lock(void);	void FLASH_Lock(void);
	FLASH_Status FLASH_ErasePage(uint32_t Page_Address);	FLASH_Status FLASH_ErasePage(uint32_t Page_Address);
	FLASH_Status FLASH_EraseAllPages(void);	NA
	FLASH_Status FLASH_EraseOptionBytes(void);	NA
	FLASH_Status FLASH_ProgramWord(uint32_t Address, uint32_t Data);	FLASH_Status FLASH_FastProgramWord(uint32_t Address, uint32_t Data);
	FLASH_Status FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data);	NA
	NA	FLASH_Status FLASH_ProgramHalfPage(uint32_t Address, uint32_t* pBuffer);
Option Byte Programming	NA	void FLASH_OB_Unlock(void);
	NA	void FLASH_OB_Lock(void);
	FLASH_Status FLASH_ProgramOptionByteData(uint32_t Address, uint8_t Data);	NA
	FLASH_Status FLASH_EnableWriteProtection(uint32_t FLASH_Pages);	FLASH_Status FLASH_OB_WRPConfig(uint32_t OB_WRP, FunctionalState NewState);
	FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState);	FLASH_Status FLASH_OB_RDPConfig(uint8_t OB_RDP);
	FLASH_Status FLASH_UserOptionByteConfig(uint16_t OB_IWDG, uint16_t OB_STOP, uint16_t OB_STDBY);	FLASH_Status FLASH_OB_UserConfig(uint8_t OB_IWDG, uint8_t OB_STOP, uint8_t OB_STDBY);
	NA	FLASH_Status FLASH_OB_BORConfig(uint8_t OB_BOR);
	NA	FLASH_Status FLASH_OB_Launch(void);
	uint32_t FLASH_GetUserOptionByte(void);	uint8_t FLASH_OB_GetUser(void);
	uint32_t FLASH_GetWriteProtectionOptionByte(void);	uint16_t FLASH_OB_GetWRP(void);
	FlagStatus FLASH_GetReadOutProtectionStatus(void);	FlagStatus FLASH_OB_GetRDP(void);
	NA	uint8_t FLASH_OB_GetBOR(void);

Table 14. STM32F10x and STM32L1xx FLASH driver API correspondence (continued)

	STM32F10x Flash driver API	STM32L1xx Flash driver API
FLAG management	FlagStatus FLASH_GetFlagStatus(uint32_t FLASH_FLAG);	FlagStatus FLASH_GetFlagStatus(uint32_t FLASH_FLAG);
	void FLASH_ClearFlag(uint32_t FLASH_FLAG);	void FLASH_ClearFlag(uint32_t FLASH_FLAG);
	FLASH_Status FLASH_GetStatus(void);	FLASH_Status FLASH_GetStatus(void);
	FLASH_Status FLASH_WaitForLastOperation(uint32_t Timeout);	FLASH_Status FLASH_WaitForLastOperation(void);
	FlagStatus FLASH_GetPrefetchBufferStatus(void);	NA
DATA EEPROM management	NA	void DATA_EEPROM_Unlock(void);
	NA	void DATA_EEPROM_Lock(void);
	NA	FLASH_Status DATA_EEPROM_EraseWord(uint32_t Address);
	NA	FLASH_Status DATA_EEPROM_EraseDoubleWord(uint32_t Address);
	NA	FLASH_Status DATA_EEPROM_FastProgramByte(uint32_t Address, uint8_t Data);
	NA	FLASH_Status DATA_EEPROM_FastProgramHalfWord(uint32_t Address, uint16_t Data);
	NA	FLASH_Status DATA_EEPROM_FastProgramWord(uint32_t Address, uint32_t Data);
	NA	FLASH_Status DATA_EEPROM_ProgramByte(uint32_t Address, uint8_t Data);
	NA	FLASH_Status DATA_EEPROM_ProgramHalfWord(uint32_t Address, uint16_t Data);
	NA	FLASH_Status DATA_EEPROM_ProgramWord(uint32_t Address, uint32_t Data);
	NA	FLASH_Status DATA_EEPROM_ProgramDoubleWord(uint32_t Address, uint64_t Data);

Color key:

= New function

= Same function, but API was changed

= Function not available (NA)

4.4 GPIO

This section explains how to update the configuration of the various GPIO modes when porting the application code from STM32 F1 series to L1 series.

4.4.1 Output mode

The example below shows how to configure an I/O in output mode (for example to drive a led) in STM32 F1 series:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_xxMHz; /* 2, 10 or 50 MHz */
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

In L1 series you have to update this code as follows:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; /* Push-pull or open drain */
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; /* None, Pull-up or pull-down */
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_xxMHz; /* 400 KHz, 2, 10 or 40MHz */
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

4.4.2 Input mode

The example below shows how to configure an I/O in input mode (for example to be used as an EXTI line) in STM32 F1 series:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

In L1 series you have to update this code as follows:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; /* None, Pull-up or pull-down */
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

4.4.3 Analog mode

The example below shows how to configure an I/O in analog mode (for example an ADC or DAC channel) in STM32 F1 series:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

In L1 series you have to update this code as follows:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_x ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOy, &GPIO_InitStructure);
```

4.4.4 Alternate function mode

In STM32 F1 series

1. The configuration to use an I/O as alternate function depends on the peripheral mode used, for example the USART Tx pin should be configured as alternate function push-pull while the USART Rx pin should be configured as input floating or input pull-up.
2. To optimize the number of peripheral I/O functions for different device packages, it is possible by software to remap some alternate functions to other pins, for example the USART2_RX pin can be mapped on PA3 (default remap) or PD6 (by software remap).

In STM32 L1 series

1. Whatever the peripheral mode used, the I/O must be configured as alternate function, then the system can use the I/O in the proper way (input or output).
2. The I/O pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral's alternate function to be connected to an I/O pin at a time. In this way, there can be no conflict between peripherals sharing the same I/O pin. Each I/O pin has a multiplexer with sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIO_PinAFConfig () function:
   – After reset all I/Os are connected to the system's alternate function 0 (AF0)
   – The peripherals' alternate functions are mapped by configuring AF1 to AF13
   – Cortex-M3 EVENTOUT is mapped by configuring AF15
3. In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripheral I/O functions for different device packages, for example the USART2_RX pin can be mapped on PA3 or PD6 pin
4. Configuration procedure
   – Connect the pin to the desired peripherals' Alternate Function (AF) using GPIO_PinAFConfig() function
   – Use GPIO_Init() function to configure the I/O pin:

     - Configure the desired pin in alternate function mode using GPIO_InitStructure->GPIO_Mode = GPIO_Mode_AF;

     - Select the type, pull-up/pull-down and output speed via GPIO_PuPd, GPIO_OType and GPIO_Speed members

The example below shows how to remap USART2 Tx/Rx I/Os on PD5/PD6 pins in STM32 F1 series:

```
/* Enable APB2 interface clock for GPIOD and AFIO (AFIO peripheral is used
   to configure the I/Os software remapping) */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_AFIO, ENABLE);

/* Enable USART2 I/Os software remapping [(USART2_Tx,USART2_Rx):(PD5,PD6)] */
GPIO_PinRemapConfig(GPIO_Remap_USART2, ENABLE);

/* Configure USART2_Tx as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOD, &GPIO_InitStructure);

/* Configure USART2_Rx as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

In L1 series you have to update this code as follows:

```
/* Enable GPIOD's AHB interface clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOD, ENABLE);

/* Select USART2 I/Os mapping on PD5/6 pins [(USART2_TX,USART2_RX):(PD5,PD6)] */
/* Connect PD5 to USART2_Tx */
GPIO_PinAFConfig(GPIOD, GPIO_PinSource5, GPIO_AF_USART2);
/* Connect PD6 to USART2_Rx*/
GPIO_PinAFConfig(GPIOD, GPIO_PinSource6, GPIO_AF_USART2);

/* Configure USART2_Tx and USART2_Rx as alternate function */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

4.5 EXTI

The example below shows how to configure the PA0 pin to be used as EXTI Line0 in STM32 F1 series:

```
/* Enable APB interface clock for GPIOA and AFIO */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);

/* Configure PA0 pin in input mode */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Connect EXTI Line0 to PA0 pin */
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);

/* Configure EXTI line0 */
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
```

In L1 series the configuration of the EXTI line source pin is performed in the SYSCFG peripheral (instead of AFIO in F1 series). As result, the source code should be updated as follows:

```
/* Enable GPIOA's AHB interface clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
/* Enable SYSCFG's APB interface clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

/* Configure PA0 pin in input mode */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Connect EXTI Line0 to PA0 pin */
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

/* Configure EXTI line0 */
```

```
  EXTI_InitStructure.EXTI_Line = EXTI_Line0;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
```

4.6 ADC

This section gives an example of how to port existing code from STM32 F1 series to L1 series.

The example below shows how to configure the ADC1 to convert continuously channel14 in STM32 F1 series:

```
  ...
  /* ADCCLK = PCLK2/4 */
  RCC_ADCCLKConfig(RCC_PCLK2_Div4);

  /* Enable ADC's APB interface clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* Configure ADC1 to convert continously channel14 */
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
  /* ADC1 regular channel14 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_55Cycles5);

  /* Enable ADC1's DMA interface */
  ADC_DMACmd(ADC1, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Enable ADC1 reset calibration register */
  ADC_ResetCalibration(ADC1);
  /* Check the end of ADC1 reset calibration register */
  while(ADC_GetResetCalibrationStatus(ADC1));

  /* Start ADC1 calibration */
  ADC_StartCalibration(ADC1);
  /* Check the end of ADC1 calibration */
  while(ADC_GetCalibrationStatus(ADC1));

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
  ...
```

In L1 series you have to update this code as follows:

```
  ...
/* Enable the HSI oscillator */
  RCC_HSICmd(ENABLE);

  /* Check that HSI oscillator is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET)
  {
  }
```

```
/* Enable ADC1 clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

/* Common configuration ***************************************************/
/* ADCCLK = HSI/1 */
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div1;
ADC_CommonInit(&ADC_CommonInitStructure);

/* ADC1 configuration */
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC1, &ADC_InitStructure);

/* ADC1 regular channel14 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_4Cycles);

/* Enable the request after last transfer for DMA Circular mode */
ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);

/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);

/* Wait until the ADC1 is ready */
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_ADONS) == RESET)
{
}

/* Start ADC1 Software Conversion */
ADC_SoftwareStartConv(ADC1);
...
```

The main changes in the source code/procedure in L1 series vs. F1 are described below:

1. ADC configuration is made through two functions ADC_CommonInit() and ADC_Init(): ADC_CommonInit() function is used to configure the ADC analog clock prescaler.
2. To enable the generation of DMA requests continuously at the end of the last DMA transfer, the ADC_DMARequestAfterLastTransferCmd() function should be used.
3. No calibration is needed

4.7 PWR

The table below presents the PWR driver API correspondence between STM32F10x and STM32L1xx Libraries. You can easily update your application code by replacing STM32F10x functions by the corresponding function in STM32L1xx Library.

Table 15. STM32F10x and STM32L1xx PWR driver API correspondence

	STM32F10x PWR driver API	STM32L1xx PWR driver API
Interface configuration	void PWR_DeInit(void);	void PWR_DeInit(void);
	void PWR_BackupAccessCmd(FunctionalState NewState);	void PWR_RTCAccessCmd(FunctionalState NewState);
PVD	void PWR_PVDLevelConfig(uint32_t PWR_PVDLevel);	void PWR_PVDLevelConfig(uint32_t PWR_PVDLevel);
	void PWR_PVDCmd(FunctionalState NewState);	void PWR_PVDCmd(FunctionalState NewState);
Wakeup	void PWR_WakeUpPinCmd(FunctionalState NewState);	void PWR_WakeUpPinCmd(uint32_t PWR_WakeUpPin, FunctionalState NewState);
	NA	void PWR_FastWakeUpCmd(FunctionalState NewState);
	NA	void PWR_UltraLowPowerCmd(FunctionalState NewState);
Power Management	NA	void PWR_VoltageScalingConfig(uint32_t PWR_VoltageScaling);
	NA	void PWR_EnterLowPowerRunMode(FunctionalState NewState);
	NA	void PWR_EnterSleepMode(uint32_t PWR_Regulator, uint8_t PWR_SLEEPEntry);
	void PWR_EnterSTOPMode(uint32_t PWR_Regulator, uint8_t PWR_STOPEntry);	void PWR_EnterSTOPMode(uint32_t PWR_Regulator, uint8_t PWR_STOPEntry);
	void PWR_EnterSTANDBYMode(void);	void PWR_EnterSTANDBYMode(void);
FLAG management	FlagStatus PWR_GetFlagStatus(uint32_t PWR_FLAG);	FlagStatus PWR_GetFlagStatus(uint32_t PWR_FLAG);
	void PWR_ClearFlag(uint32_t PWR_FLAG);	void PWR_ClearFlag(uint32_t PWR_FLAG);

Color key:

= New function

= Same function, but API was changed

= Function not available (NA)

4.8 Backup data registers

In STM32 F1 series the Backup data registers are managed through the BKP peripheral, while in L1 series they are a part of the RTC peripheral (there is no BKP peripheral).

The example below shows how to write to/read from Backup data registers in STM32 F1 series:

```
uint16_t BKPdata = 0;

  ...
  /* Enable APB2 interface clock for PWR and BKP */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_BKP, ENABLE);

  /* Enable write access to Backup domain */
  PWR_BackupAccessCmd(ENABLE);

  /* Write data to Backup data register 1 */
  BKP_WriteBackupRegister(BKP_DR1, 0x3210);

  /* Read data from Backup data register 1 */
  BKPdata = BKP_ReadBackupRegister(BKP_DR1);
```

In L1 series you have to update this code as follows:

```
uint16_t BKPdata = 0;

  ...
  /* PWR Clock Enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

  /* Enable write access to RTC domain */
  PWR_RTCAccessCmd(ENABLE);

  /* Write data to Backup data register 1 */
  RTC_WriteBackupRegister(RTC_BKP_DR1, 0x3220);

  /* Read data from Backup data register 1 */
  BKPdata = RTC_ReadBackupRegister(RTC_BKP_DR1);
```

The main changes in the source code in L1 series vs. F1 are described below:

1. There is no BKP peripheral
2. Write to/read from Backup data registers are done through RTC driver
3. Backup data registers naming changed from BKP_DRx to RTC_BKP_DRx, and numbering starts from 0 instead of 1

5 Revision history

Table 16. Document revision history

Date	Revision	Changes
20-Jul-2011	1	Initial release

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com