复旦微电子

FM33LC0 系列切换至 FM33LE0 系列软件变更说明_V1.0

1. 概述

为了方便用户从 FM33LC0xx 切换至 FM33LE0xx，本文主要说明 FM33LE0xx 相对 FM33LC0xx 减少或变化的部分，新增的功能不影响切换的情况下本文不做特别说明。

手册请参考 FM331E0 系列产品说明书 V1.1.0 后的版本（包括 V1.1.0）。

2. 版本记录

初版	2022.5	

3. 相同的模块

以下模块 FM33LE0xx 与 FM33LC0xx 完全相同：

GPIO、SPI、LPTIME32、BSTIM32、I2C、RTC、LCD、LPUART、TRNG、AES、CRC、HDIV、IWDT、WWDT、DMA、FLASH、RTC。

其中 TRNG 和 RTC 驱动和例程中各有一个函数修改，不影响使用。后文有描述。

FM33LE0xx 删除了 OPA 和 USB 两个模块。

4. 删除的模块

5. 修改的模块

5.1. RCC

驱动：

◆fm33le0xx 是删除了 USB 相关的部分，用户假如没有使用 USB 可以不需要修改程序

fm33le0xx_fl_rcc.h 删除了 USB 相关部分

fm33le0xx_fl_rcc.c 删除了 USB 相关部分

◆fm33le0xx 的 LPOSC 上电自动使能，软件可关闭，注意是写 0 使能，删除了 Chopper 使能和低功耗关闭使能

fm33le0xx 寄存器 RCC_LPOSCCR：

Bit	助记符	功能描述
31:1	--	RFU：未实现，读为 0
0	LPOSC_ENB	LPOSC 使能寄存器 0：使能 LPOSC 1：关闭 LPOSC

fm33lc0xx 寄存器 RCC_LPOSCCR：

位号	助记符	功能描述
31:3	-	RFU：未实现，读为 0
2	LPO_CHOP_EN	LPOSC Chopper 使能寄存器 (LPOSC chopper enable)
1	LPO_ENB	LPOSC 使能标志信号，只读，供软件查询 LPOSC 使能状态 (LPOSC Enable Bar,read only) 0：LPOSC 处于开启状态 1：LPOSC 处于关闭状态
0	LPM_LPO_OFF	软件禁止改写此寄存器，保持复位值

相关宏定义的修改，函数的修改不列举：

fm33le0xx_fl_rcc.h 与 fm33lc0xx_fl_rcc.h

fm33le

```
#define    RCC_LPOSCCR_LPOENB_Pos                          (0U)
#define    RCC_LPOSCCR_LPOENB_Msk                          (0x1U << RCC_LPOSCC
#define    RCC_LPOSCCR_LPOENB                              RCC_LPOSCCR_LPOENB_
```

fm33lc

```
#define    RCC_LPOSCCR_LPOENB_Pos                          (1U)
#define    RCC_LPOSCCR_LPOENB_Msk                          (0x1U << RC
#define    RCC_LPOSCCR_LPOENB                              RCC_LPOSCCR

#define    RCC_LPOSCCR_LPM_LPO_OFF_Pos                     (0U)
#define    RCC_LPOSCCR_LPM_LPO_OFF_Msk                     (0x1U << RC
#define    RCC_LPOSCCR_LPM_LPO_OFF                         RCC_LPOSCCR

#define    RCC_LPOSCCR_LPO_CHOP_EN_Pos                     (2U)
#define    RCC_LPOSCCR_LPO_CHOP_EN_Msk                     (0x1U << RC
#define    RCC_LPOSCCR_LPO_CHOP_EN                         RCC_LPOSCCR
```

◆fm33le0xx 的清除停振标志,bit0 写 1 就是清除 LFDETIF,bit1 写 1 就是清除 HFDETIF。

而 fm33lc0xx 的清除停振标志，bit0 写 1 就是清除 HFDETIF，bit1 写 1 就是清除 LFDETIF。用户需要注意将底层驱动中的函数修改下.

fm33le0xx_fl_rcc.h 与 fm33lc0xx_fl_rcc.h：

fm33le

```
/**
  * @brief    Clear XTHF Vibrating Flag
  * @rmtoll   ISR    HFDETIF    FL_FDET_ClearFlag_XTHFFail
  * @retval   None
  */
__STATIC_INLINE void FL_FDET_ClearFlag_XTHFFail(void)
{
    WRITE_REG(FDET->ISR, FDET_ISR_HFDETIF_Msk);
}

/**
  * @brief    Get XTLF Vibrating Output
  * @rmtoll   ISR    LFDETIF    FL_FDET_IsActiveFlag_XTLFFail
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_FDET_IsActiveFlag_XTLFFail(void)
{
    return (uint32_t)(READ_BIT(FDET->ISR, FDET_ISR_LFDETIF_Msk) == (FDET_ISR_LFDE
}

/**
  * @brief    Clear XTLF Vibrating Output
  * @rmtoll   ISR    LFDETIF    FL_FDET_ClearFlag_XTLFFail
  * @retval   None
  */
__STATIC_INLINE void FL_FDET_ClearFlag_XTLFFail(void)
{
    WRITE_REG(FDET->ISR, FDET_ISR_LFDETIF_Msk);
}
```

fm33lc

```
/**
  * @brief    Clear XTHF Vibrating Flag
  * @rmtoll   ISR    HFDETIF    FL_FDET_ClearFlag_XTHFFail
  * @retval   None
  */
__STATIC_INLINE void FL_FDET_ClearFlag_XTHFFail(void)
{
    WRITE_REG(FDET->ISR, FDET_ISR_LFDETIF_Msk);
}

/**
  * @brief    Get XTLF Vibrating Output
  * @rmtoll   ISR    LFDETIF    FL_FDET_IsActiveFlag_XTLFFail
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_FDET_IsActiveFlag_XTLFFail(void)
{
    return (uint32_t)(READ_BIT(FDET->ISR, FDET_ISR_LFDETIF_Msk) == (FDET_ISR_L
}

/**
  * @brief    Clear XTLF Vibrating Output
  * @rmtoll   ISR    LFDETIF    FL_FDET_ClearFlag_XTLFFail
  * @retval   None
  */
__STATIC_INLINE void FL_FDET_ClearFlag_XTLFFail(void)
{
    WRITE_REG(FDET->ISR, FDET_ISR_HFDETIF_Msk);
}
```

◆fm33le0xx 的 LPOSC 频率的赋值改为了 32768，而 fm33lc0xx 的为 32000

fm33le0xx_fl_rcc.c 与 fm33lc0xx_fl_rcc.c：

```
/* 系统时钟源为LSCLK */
case FL_RCC_SYSTEM_CLK_SOURCE_LSCLK:
    #ifdef USE_LSCLK_CLOCK_SRC_LPOSC
    frequency = 32768;
    #else
    frequency = XTLFClock;
    #endif
    break;

/* 系统时钟源为LPOSC */
case FL_RCC_SYSTEM_CLK_SOURCE_LPOSC:
    frequency = 32768;
```

fm331e

```
/* 系统时钟源为LSCLK */
case FL_RCC_SYSTEM_CLK_SOURCE_LSCLK:
    #ifdef USE_LSCLK_CLOCK_SRC_LPOSC
    frequency = 32000;
    #else
    frequency = XTLFClock;
    #endif
    break;
/* 系统时钟源为USB BCK */
case FL_RCC_SYSTEM_CLK_SOURCE_USBCLK:
    /* USB时钟频率获取 */
    frequency = FL_RCC_GetUSBClockFreqToSysclk();
    break;
/* 系统时钟源为LPOSC */
case FL_RCC_SYSTEM_CLK_SOURCE_LPOSC:
    frequency = 32000;
```

fm331c

例程：

PLL 的例程 FM33LE0xx 直接输入想要输出的时钟，不必要再减 1，相应的底下函数有修改。FM33LC0xx 中的 pll 例程也可以直接使用，但注意不要把两个 pll.c 中的函数混起来。

例程 PLL_SelRCHF 与 PLL_SelXTHF 中 main.c：

```
SelRCHFToPLL(FL_RCC_RCHF_FREQUENCY_8MHZ, 48);
```

1e

```
SelRCHFToPLL(FL_RCC_RCHF_FREQUENCY_8MHZ, 48 - 1);
```

1c

```
SelXTHFToPLL(48);
```

1e

```
SelXTHFToPLL(48 - 1);
```

1c

5.2. RMU

◆PDR 的档位变化，最低档位由 1.25V 变大 1.3V，相应的宏定义修改

fm33le0xx_fl_rmu.h：

```
#define    FL_RMU_PDR_THRESHOLD_1P5V                (0x0U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P3V                (0x1U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P35V               (0x2U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P4V                (0x3U << RMU_PDRCR_CFG_Pos)
```

fm33lc0xx_fl_rmu.h：

```
#define    FL_RMU_PDR_THRESHOLD_1P5V                (0x0U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P25V               (0x1U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P35V               (0x2U << RMU_PDRCR_CFG_Pos)
#define    FL_RMU_PDR_THRESHOLD_1P4V                (0x3U << RMU_PDRCR_CFG_Pos)
```

◆fm33le0xx 的 BOR 使能不区分上电复位和下电复位控制，上电复位功能上电时默认使能，上电复位完成后自动关闭。fm33le0xx 的下电复位的控制位在 bit1，fm33lc0xx 的下电复位控制位在 bit0.

宏定义的变化：

fm33le0xx_fl_rmu.h:

```
#define    RMU_BORCR_ENB_Pos                        (1U)
#define    RMU_BORCR_ENB_Msk                        (0x1U << RMU_BORCR_ENB_Pos)
#define    RMU_BORCR_ENB                            RMU_BORCR_ENB_Msk
```

fm33lc0xx_fl_rmu.h:

```
#define    RMU_BORCR_OFF_BOR_1P2_Pos                    (0U)
#define    RMU_BORCR_OFF_BOR_1P2_Msk                    (0x1U << RMU_BORCR_OFF_BOR_1P2_Pos)
#define    RMU_BORCR_OFF_BOR_1P2                        RMU_BORCR_OFF_BOR_1P2_Msk

#define    RMU_BORCR_OFF_BOR_1P0_Pos                    (1U)
#define    RMU_BORCR_OFF_BOR_1P0_Msk                    (0x1U << RMU_BORCR_OFF_BOR_1P0_Pos)
#define    RMU_BORCR_OFF_BOR_1P0                        RMU_BORCR_OFF_BOR_1P0_Msk
```

驱动函数的变化：

fm33le0xx_fl_rmu.h 与 fm33lc0xx_fl_rmu.h

fm33le

```
/**
  * @brief    Get BOR Power Down Reset Enable Status
  * @rmtoll   BORCR    ENB    FL_RMU_BORPowerDown_IsEnabled
  * @param    RMUx RMU instance
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_RMU_BORPowerDown_IsEnabled(RMU_Type *RMUx)
{
    return (uint32_t)!(READ_BIT(RMUx->BORCR, RMU_BORCR_ENB_Msk) == RMU_BORCR_ENB_Msk);
}

/**
  * @brief    Disable BOR Power Down Reset
  * @rmtoll   BORCR    ENB    FL_RMU_BORPowerDown_Disable
  * @param    RMUx RMU instance
  * @retval   None
  */
__STATIC_INLINE void FL_RMU_BORPowerDown_Disable(RMU_Type *RMUx)
{
    SET_BIT(RMUx->BORCR, RMU_BORCR_ENB_Msk);
}

/**
  * @brief    Enable BOR Power Down Reset
  * @rmtoll   BORCR    ENB    FL_RMU_BORPowerDown_Enable
  * @param    RMUx RMU instance
  * @retval   None
  */
__STATIC_INLINE void FL_RMU_BORPowerDown_Enable(RMU_Type *RMUx)
{
    CLEAR_BIT(RMUx->BORCR, RMU_BORCR_ENB_Msk);
}
```

fm331c

```
/**
  * @brief    Get BOR Power Down Reset Enable Status
  * @rmtoll   BORCR    OFF_BOR_1P2    FL_RMU_BORPowerDown_IsEnabled
  * @param    RMUx RMU instance
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_RMU_BORPowerDown_IsEnabled(RMU_Type *RMUx)
{
    return (uint32_t)!(READ_BIT(RMUx->BORCR, RMU_BORCR_OFF_BOR_1P2_Msk) == R
}

/**
  * @brief    Disable BOR Power Down Reset
  * @rmtoll   BORCR    OFF_BOR_1P2    FL_RMU_BORPowerDown_Disable
  * @param    RMUx RMU instance
  * @retval   None
  */
__STATIC_INLINE void FL_RMU_BORPowerDown_Disable(RMU_Type *RMUx)
{
    SET_BIT(RMUx->BORCR, RMU_BORCR_OFF_BOR_1P2_Msk);
}

/**
  * @brief    Enable BOR Power Down Reset
  * @rmtoll   BORCR    OFF_BOR_1P2    FL_RMU_BORPowerDown_Enable
  * @param    RMUx RMU instance
  * @retval   None
  */
__STATIC_INLINE void FL_RMU_BORPowerDown_Enable(RMU_Type *RMUx)
{
    CLEAR_BIT(RMUx->BORCR, RMU_BORCR_OFF_BOR_1P2_Msk);
}
```

◆工程没有 fm33le0xx_fl_rmu.c、fm33lc0xx_fl_rmu.c 文件不用关注

5.3. FLASH

◆Flash 功能没有变化，但容量的改变，注意相应宏定义的变化

fm33le0xx_fl_flash.h 与 fm33lc0xx_fl_flash.h

fm33le

```
#define    FL_FLASH_MAX_PAGE_NUM                        0x00000100U
#define    FL_FLASH_MAX_SECTOR_NUM                      0x00000040U
#define    FL_FLASH_SECTOR_SIZE_BYTE                    0x00000800U
#define    FL_FLASH_PAGE_SIZE_BYTE                      0x00000200U
#define    FL_FLASH_ADDR_MAXPROGRAM                     0x0001FFFFU
```

fm331c

```
#define    FL_FLASH_MAX_PAGE_NUM                        0x00000200U
#define    FL_FLASH_MAX_SECTOR_NUM                      0x00000080U
#define    FL_FLASH_SECTOR_SIZE_BYTE                    0x00000800U
#define    FL_FLASH_PAGE_SIZE_BYTE                      0x00000200U
#define    FL_FLASH_ADDR_MAXPROGRAM                     0x0003FFFFU
```

◆Flash 例程中有一处需要修正，后续 fm33lc0xx 的 flash 例程通样会进行更新

FLASH 读写例程的 flash.c

fm33le

```
void FlashRW(void)
{
    FL_FLASH_PageErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0x55, BUFFER_SIZE * 4);
    FL_FLASH_Program_Word(FLASH, FLASH_PROG_ADDR, DataBuffer[0]);

    FL_FLASH_PageErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0xAA, BUFFER_SIZE * 4);
    FL_FLASH_Program_Page(FLASH, FLASH_PROG_ADDR / FL_FLASH_PAGE_SIZE_BYTE, DataBuffe

    FL_FLASH_SectorErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0x00, BUFFER_SIZE * 4);
    FL_FLASH_Program_Sector(FLASH, FLASH_PROG_ADDR / FL_FLASH_SECTOR_SIZE_BYTE, DataB

    FL_FLASH_SectorErase(FLASH, FLASH_PROG_ADDR);
}
```

fm331c

```
void FlashTest(void)
{
    FL_FLASH_PageErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0x55, BUFFER_SIZE * 4);
    FL_FLASH_Program_Word(FLASH, FLASH_PROG_ADDR, DataBuffer[0]);

    FL_FLASH_PageErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0xAA, BUFFER_SIZE * 4);
    FL_FLASH_Program_Page(FLASH, FLASH_PROG_ADDR / FL_FLASH_MAX_PAGE_NUM, Da

    FL_FLASH_SectorErase(FLASH, FLASH_PROG_ADDR);
    memset(DataBuffer, 0x00, BUFFER_SIZE * 4);
    FL_FLASH_Program_Sector(FLASH, FLASH_PROG_ADDR / FL_FLASH_SECTOR_SIZE_BY

    FL_FLASH_SectorErase(FLASH, FLASH_PROG_ADDR);
}
```

5.4. PMU

驱动：

◆fm33le0xx 在 WKFR 寄存器空余的位置新增了些唤醒标志，不影响 fm33lc0xx 的程序。

例程：

◆FM33LC0XX 休眠例程中休眠时关闭 VREF 和 ADC 电源检测，在 FM33LE0XX 中已经删除，不需要再执行这两个函数。

fm33lc0xx 休眠例程中的休眠函数：

```
void Sleep(void)
{
    FL_RCC_RCMF_Disable();                  //关闭RCMF
    FL_RMU_PDR_Enable(RMU);                 //打开PDR
    FL_RMU_BORPowerDown_Disable(RMU);       //关闭BOR 2uA

    /*使用ADC时ADCMonitor功能以及Vref需同时开始，同时关闭*/
    FL_VREF_Disable(VREF);                  //关闭VREF1p2
    FL_SVD_DisableADCMonitor(SVD);          //关闭ADC电源检测
    FL_ADC_Disable(ADC);                    //关闭ADC使能

    FL_PMU_SetLowPowerMode(PMU, FL_PMU_POWER_MODE_SLEEP_OR_DEEPSLEEP);
    __WFI();
}
```

5.5. VREF

VREF 有个很大的变化是 fm33lc0xx 的 VREF 为 1.22V，它有使能开关和建立时间，用于 ADC、BOR、SVD。

fm33le0xx 的 VREF 为 1V，没有开关控制，上电就使能，3us 左右就建立完成。在 deepsleep 时自动关闭，唤醒会自动打开。所以 fm33le0xx 在 deepsleep 下不能使用 BOR 功能。

驱动：

◆fm33le0xx 的 VREF 没有使能和建立的标志，fm33le0xx_fl_vref.h 文件中没有相应的宏定义和函数

◆fm33le0xx 的 PTAT 控制寄存器在 VREF _PTATCR 寄存器，fm33lc0xx 在 VREF_CR 寄存器 fm33le0xx_fl_vref.h 与 fm33lc0xx_fl_vref.h

```
/**
  * @brief   Enable Temperatue Sensor
  * @rmtoll  CR    PTAT_EN    FL_VREF_EnableTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  None
  */
__STATIC_INLINE void FL_VREF_EnableTemperatureSensor(VREF_Type *VREFx)
{
    SET_BIT(VREFx->PTATCR, VREF_CR_PTAT_EN_Msk);
}
/**
  * @brief   Get Temperatue Sensor Enable Status
  * @rmtoll  CR    PTAT_EN    FL_VREF_IsEnabledTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_VREF_IsEnabledTemperatureSensor(VREF_Type *VREFx)
{
    return (uint32_t)(READ_BIT(VREFx->PTATCR, VREF_CR_PTAT_EN_Msk) == VREF_CR_PTAT_EN_Msk);
}
/**
  * @brief   Disable Temperatue Sensor
  * @rmtoll  CR    PTAT_EN    FL_VREF_DisableTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  None
  */
__STATIC_INLINE void FL_VREF_DisableTemperatureSensor(VREF_Type *VREFx)
{
    CLEAR_BIT(VREFx->PTATCR, VREF_CR_PTAT_EN_Msk);
}
```

1e

```
/**
  * @brief   Enable Temperatue Sensor
  * @rmtoll  CR    PTAT_EN    FL_VREF_EnableTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  None
  */
__STATIC_INLINE void FL_VREF_EnableTemperatureSensor(VREF_Type *VREFx)
{
    SET_BIT(VREFx->CR, VREF_CR_PTAT_EN_Msk);
}
/**
  * @brief   Get Temperatue Sensor Enable Status
  * @rmtoll  CR    PTAT_EN    FL_VREF_IsEnabledTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_VREF_IsEnabledTemperatureSensor(VREF_Type *VREFx)
{
    return (uint32_t)(READ_BIT(VREFx->CR, VREF_CR_PTAT_EN_Msk) == VREF_CR_PTAT_EN_Msk);
}
/**
  * @brief   Disable Temperatue Sensor
  * @rmtoll  CR    PTAT_EN    FL_VREF_DisableTemperatureSensor
  * @param   VREFx VREF instance
  * @retval  None
  */
__STATIC_INLINE void FL_VREF_DisableTemperatureSensor(VREF_Type *VREFx)
{
    CLEAR_BIT(VREFx->CR, VREF_CR_PTAT_EN_Msk);
}
```

1c

5.6. SVD

对现有方案影响比较大的变化是 fm33le0xx 的基准电压变为 1V，导致 SVD 各档位阈值发生

了变化，档位阈值具体请参考手册。
手册请参考 fm33le0 系列产品说明书 V2.0 后的版本（包括 V2.0）
◆fm33le0xx 的 SVD 增加了间歇使能的功能，结构体和增加了相应函数有所变化，但不影响原有的程序。间歇功能的使用方法可以参考例程。

fm33le0xx_fl_svd.h 与 fm33lc0xx_fl_svd.h：

```
typedef struct
{
    /* 工作模式 */
    uint32_t workMode;

    /* 间歇使能间隔 */
    uint32_t enablePeriod;

    /* 参考电压 */
    uint32_t referenceVoltage;

    /* 报警阈值 */
    uint32_t warningThreshold;

    /* 数字滤波 */
    uint32_t digitalFilter;

    /* SVS通道选择 */
    uint32_t SVSChannel;

} FL_SVD_InitTypeDef;
```

1e

```
typedef struct
{
    /* 参考电压 */
    uint32_t referenceVoltage;

    /* 报警阈值 */
    uint32_t warningThreshold;

    /* 数字滤波 */
    uint32_t digitalFilter;

    /* SVS通道选择 */
    uint32_t SVSChannel;

} FL_SVD_InitTypeDef;
```

1c

◆fm33le0xx 的 SVD 基准电压变为 1V，所以 SVD 比较基准档位有所变化,程序中相关宏定义和注释有变化
fm33le0xx_fl_svd.h:

```
#define     FL_SVD_REFERENCE_1P0V                                   (0x1U << 2U)
#define     FL_SVD_REFERENCE_0P95V                                  (0x1U << 1U)
#define     FL_SVD_REFERENCE_0P9V                                   (0x1U << 0U)
```

fm33lc0xx_fl_svd.h

```
#define     FL_SVD_REFERENCE_1P2V                                   (0x1U << 2U)
#define     FL_SVD_REFERENCE_1P1V                                   (0x1U << 1U)
#define     FL_SVD_REFERENCE_1P0V                                   (0x1U << 0U)
```

◆fm33le0xx 在 SVD_CR 寄存器增加了 SVSCFG，用于在使用外部检测时屏蔽内部电阻串的影响。而 fm33lc0xx 在此应用中外部电阻与内部电阻串是并联的，用户可以选择使不使用这个功能。默认是 0，兼容 fm33lc0xx。
fm33le0xx 寄存器 SVD_CR:

Bit	助记符	功能描述
31:10	--	RFU：未实现，读为 0
9	SVSCFG	SVS 兼容性配置 0：兼容 FM33LC0，外部电阻串与内部电阻串并联 1：内部电阻串不影响外部电阻分压
8	SVDTE	SVD 测试使能，避免写 1
7:2	--	RFU：未实现，读为 0
1	SVSEN	SVS 外部电源检测通道控制信号 0：SVS 通道关闭 1：SVS 通道使能 当 SVS_EN=1 时，根据 SVDLVL 寄存器可以设置 SVS 输入后是 经过内部电阻分压；如果 SVDLVL=1111，则 SVS 输入不做分压 如果 SVDLVL != 1111，则 SVS 输入经过内部电阻分压。
0	SVDEN	SVD 使能 1：启动 SVD 0：关闭 SVD

5.7. TRNG

FM33LE0XX 和 FM33LC0XX 完全一样没有修改，只是例程中修改了一处 BUG

例程 RNG 中 rng.c：

1e

```
FL_ErrorStatus Test_CRC32_MPEG2(void)
{
    uint32_t crc32;
    crc32 = GetCrc32(0x12345678);
    if(crc32 == 0xDF8A8A2B)
    {
        return FL_PASS;
    }

    return FL_FAIL;
}
```

1c

```
FL_ErrorStatus Test_CRC32_MPEG2(void)
{
    uint32_t crc32;
    crc32 = GetCrc32(0x12345678);
    if(crc32 == 0xDF8A8A2B)
    {
        return FL_FAIL;
    }

    return FL_PASS;
}
```

5.8. Comparator

◆LC0 的 COMP 的 buffen 放在 OPA 寄存器内，LE0 改到 COMP 中

fm33le0xx 寄存器 COMP_ICR:

Bit	助记符	功能描述
31:12	--	RFU：未实现，读为 0
11	BUFBYP	比较器 buffer bypass 0：不 bypass 比较器 buffer 1：bypass 比较器 buffer
10	BUFENB	比较器 buffer 使能 0：使能比较器 buffer 1：关闭比较器 buffer
		比较器 2 数字滤波使能

fm33lc0xx 寄存器 OPA1_CR:

位号	助记符	功能描述
31	BUFENB	VREF BUFFER 使能信号 (VREF buffer enable bar) 0：使能 VREF BUFFER 1：关闭 VREF BUFFER
30	BUFBYP	VREF BUFFER 旁路控制 (VREF buffer bypass enable) 0：不 bypass VREF BUFFER 1：bypass VREF BUFFER

驱动：

fm33le0xx_fl_comp.h：

```
#define    COMP_ICR_BUFBYP_Pos                                (11U)
#define    COMP_ICR_BUFBYP_Msk                                (0x1U << COMP_ICR_BUFBYP_Pos)
#define    COMP_ICR_BUFBYP                                    COMP_ICR_BUFBYP_Msk

#define    COMP_ICR_BUFENB_Pos                                (10U)
#define    COMP_ICR_BUFENB_Msk                                (0x1U << COMP_ICR_BUFENB_Pos)
#define    COMP_ICR_BUFENB                                    COMP_ICR_BUFENB
```

```
  * @brief    Enable VREF Buffer Bypass
  * @rmtoll   CR    BUFBYP    FL_COMP_EnableBypassVrefBuffer
  * @param    COMPx COMP instance
  * @retval   None
  */
__STATIC_INLINE void FL_COMP_EnableBypassVrefBuffer(COMP_COMMON_Type *COMPx)
{
    SET_BIT(COMPx->ICR, COMP_ICR_BUFBYP_Msk);
}

/**
  * @brief    Get VREF Buffer Bypass Enable Status
  * @rmtoll   CR    BUFBYP    FL_COMP_IsEnabledBypassVrefBuffer
  * @param    COMPx COMP instance
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_COMP_IsEnabledBypassVrefBuffer(COMP_COMMON_Type *COMPx)
{
    return (uint32_t)(READ_BIT(COMPx->ICR, COMP_ICR_BUFBYP_Msk) == COMP_ICR_BUFBYP_Msk);
}

/**
  * @brief    Disable VREF Buffer Bypass
  * @rmtoll   CR    BUFBYP    FL_COMP_DisableBypassVrefBuffer
  * @param    COMPx COMP instance
  * @retval   None
  */
__STATIC_INLINE void FL_COMP_DisableBypassVrefBuffer(COMP_COMMON_Type *COMPx)
{
    CLEAR_BIT(COMPx->ICR, COMP_ICR_BUFBYP_Msk);
}

/**
  * @brief    Enable VREF Buffer
  * @rmtoll   CR    BUFFEN    FL_COMP_EnableVrefBuffer
  * @param    COMPx COMP instance
  * @retval   None
  */
__STATIC_INLINE void FL_COMP_EnableVrefBuffer(COMP_COMMON_Type *COMPx)
{
    CLEAR_BIT(COMPx->ICR, COMP_ICR_BUFENB_Msk);
}
```

```
/**
  * @brief    Get VREF Buffer Enable Status
  * @rmtoll   CR    BUFFEN    FL_COMP_IsEnabledVrefBuffer
  * @param    COMPx COMP instance
  * @retval   State of bit (1 or 0).
  */
__STATIC_INLINE uint32_t FL_COMP_IsEnabledVrefBuffer(COMP_COMMON_Type *COMPx)
{
    return (uint32_t)!(READ_BIT(COMPx->ICR, COMP_ICR_BUFENB_Msk) == COMP_ICR_BUFENB_Msk);
}

/**
  * @brief    Disable VREF Buffer
  * @rmtoll   CR    BUFFEN    FL_COMP_DisableVrefBuffer
  * @param    COMPx COMP instance
  * @retval   None
  */
__STATIC_INLINE void FL_COMP_DisableVrefBuffer(COMP_COMMON_Type *COMPx)
{
    CLEAR_BIT(COMPx->ICR, COMP_ICR_BUFENB_Msk);
}
```

fm33le0xx_fl_comp.c 和 fm33lc0xx_fl_comp.c 中 FL_COMP_Init()函数除了修改 buffer 变化的更改，还在中断边沿选择处加了位移，位移部分后续 fm33lc0xx 同样会更新：

fm33le

```
FL_ErrorStatus FL_COMP_Init(COMP_Type *COMPx, FL_COMP_InitTypeDef *initStruct)
{
    /* 入口参数检查 */
    assert_param(IS_COMP_ALL_INSTANCE(COMPx));
    assert_param(IS_FL_COMP_EDGE(initStruct->edge));
    assert_param(IS_FL_COMP_POLARITY(initStruct->polarity));
    assert_param(IS_FL_COMP_POSITIVEINPUT(initStruct->positiveInput));
    assert_param(IS_FL_COMP_NEGATIVEINPUT(initStruct->negativeInput));
    assert_param(IS_FL_COMP_DIGITAL_FILTER(initStruct->digitalFilter));
    /* 使能时钟总线 */
    FL_RCC_EnableGroup1BusClock(FL_RCC_GROUP1_BUSCLK_ANAC);
    /* 比较器输出极性选择 */
    FL_COMP_SetOutputPolarity(COMPx, initStruct->polarity);
    /* 比较器正向输入选择 */
    FL_COMP_SetINPSource(COMPx, initStruct->positiveInput);
    /* 比较器反向输入选择 */
    FL_COMP_SetINNSource(COMPx, initStruct->negativeInput);
    /* 比较器使用vref 打开vref_buf */
    if((initStruct->negativeInput == FL_COMP_INN_SOURCE_VREF) || (initStruct->negativeI
    {
        FL_COMP_EnableVrefBuffer(COMP);/* 使能 */
        FL_COMP_DisableBypassVrefBuffer(COMP);/* 不bypass */
    }
    if(COMPx == COMP1)
    {
        /* 比较器中断边沿选择 */
        FL_COMP_SetComparator1InterruptEdge(COMP, ((initStruct->edge)<<COMP_ICR_CMP1SEL
    }
    else
    {
        /* 比较器中断边沿选择 */
        FL_COMP_SetComparator2InterruptEdge(COMP, ((initStruct->edge)<<COMP_ICR_CMP
    }
    /* 滤波 */
    if(initStruct->digitalFilter)
    {
        if(COMPx == COMP1)
        { FL_COMP_EnableComparator1OutputFilter(COMP); }
        else
        { FL_COMP_EnableComparator2OutputFilter(COMP); }
    }
```

fm33lc

```
FL_ErrorStatus FL_COMP_Init(COMP_Type *COMPx, FL_COMP_InitTypeDef *initSt
{
    /* 入口参数检查 */
    assert_param(IS_COMP_ALL_INSTANCE(COMPx));
    assert_param(IS_FL_COMP_EDGE(initStruct->edge));
    assert_param(IS_FL_COMP_POLARITY(initStruct->polarity));
    assert_param(IS_FL_COMP_POSITIVEINPUT(initStruct->positiveInput));
    assert_param(IS_FL_COMP_NEGATIVEINPUT(initStruct->negativeInput));
    assert_param(IS_FL_COMP_DIGITAL_FILTER(initStruct->digitalFilter));
    /* 使能时钟总线 */
    FL_RCC_EnableGroup1BusClock(FL_RCC_GROUP1_BUSCLK_ANAC);
    /* 比较器输出极性选择 */
    FL_COMP_SetOutputPolarity(COMPx, initStruct->polarity);
    /* 比较器正向输入选择 */
    FL_COMP_SetINPSource(COMPx, initStruct->positiveInput);
    /* 比较器反向输入选择 */
    FL_COMP_SetINNSource(COMPx, initStruct->negativeInput);
    /* 比较器使用vref 打开vref_buf */
    if((initStruct->negativeInput == FL_COMP_INN_SOURCE_VREF) || (initSt
    {
        FL_OPA_EnableVrefBuffer(OPA1);  /* 使能 */
        FL_OPA_DisableBypassVrefBuffer(OPA1);  /* 不bypass */
    }
    if(COMPx == COMP1)
    {
        /* 比较器中断边沿选择 */
        FL_COMP_SetComparator1InterruptEdge(COMP, initStruct->edge);
    }
    else
    {
        /* 比较器中断边沿选择 */
        FL_COMP_SetComparator2InterruptEdge(COMP, initStruct->edge);
    }
    /* 滤波 */
    if(initStruct->digitalFilter)
    {
        if(COMPx == COMP1)
        { FL_COMP_EnableComparator1OutputFilter(COMP); }
        else
        { FL_COMP_EnableComparator2OutputFilter(COMP); }
    }
```

◆Comp 正负端通道的变化

fm33le0xx 中寄存器 COMP1CR:

		1：输出极反
4:3	V1PSEL	比较器 1 正极输入选择 (Comparator1 positive input select) 00：COMP1_INP1 (PD4) 01：COMP1_INP2 (PD5) 10：COMP1_INP3 (PA13) 11：COMP1_INP4 (PA14)
2:1	V1NSEL	比较器 1 负极输入选择 (Comparator1 negative input select) 00：COMP1_INN1 01：RFU 10：VREF = 1.0V 11：VREF/2 = 0.5V

fm33le0xx 中寄存器 COMP2CR:

3	V2PSEL	比较器 2 正极输入选择 (Comparator1 positive input select) 0：COMP2_INP1 (PA8) 1：COMP2_INP2 (PA9)
2:1	V2NSEL	比较器 2 负极输入选择 (Comparator1 negative input select) 00：COMP2_INN1 (PA4) 01：COMP2_INN2 (PA5) 10：VREF = 1.0V 11：VREF/2 = 0.5V

fm33lc0xx 中寄存器 COMP1CR:

4:3	V1PSEL	比较器 1 正极输入选择 (Comparator1 positive input select) 00：COMP1_INP1 (PD4) 01：COMP1_INP2 (PD5) 10：COMP1_INP3 (PB12) 11：RFU
2:1	V1NSEL	比较器 1 负极输入选择 (Comparator1 negative input select) 00：COMP1_INN1 01：RFU 10：VREF = 1.2V 11：VREF/2 = 0.6V

fm33lc0xx 中寄存器 COMP2CR

3	V2PSEL	比较器 2 正极输入选择 (Comparator1 positive input select) 0：COMP2_INP1 (PA8) 1：COMP2_INP2 (PA9)
2:1	V2NSEL	比较器 2 负极输入选择 (Comparator1 negative input select) 00：COMP2_INN1 (PA4) 01：COMP2_INN2 (PA5) 10：VREF = 1.2V 11：VREF/2 = 0.6V

fm33le0xx_fl_comp.h 增加相关宏定义

```
#define    FL_COMP_INP_SOURCE_INP1                    (0x0U << COMP_CR_VPSEL_Pc
#define    FL_COMP_INP_SOURCE_INP2                    (0x1U << COMP_CR_VPSEL_Pc
#define    FL_COMP_INP_SOURCE_INP3                    (0x2U << COMP_CR_VPSEL_Pc
#define    FL_COMP_INP_SOURCE_INP4                    (0x3U << COMP_CR_VPSEL_Pc

#define    FL_COMP_INN_SOURCE_INN1                    (0x0U << COMP_CR_VNSEL_Pc
#define    FL_COMP_INN_SOURCE_INN2                    (0x1U << COMP_CR_VNSEL_Pc
#define    FL_COMP_INN_SOURCE_VREF                    (0x2U << COMP_CR_VNSEL_Pc
#define    FL_COMP_INN_SOURCE_VREF_DIV_2              (0x3U << COMP_CR_VNSEL_Pc
```

5.9. UART

◆UART 主要新增了 UART2，不影响原来程序的正常使用，另外 fm33le0xx 所有的 UART 都支持低功耗唤醒和接收超时功能

◆fm33le0xx_fl_uart.h 中 busy 的 bit 位需要修正，后续 fm33lc0xx 同样会进行更新：

fm33le

```
#define    UART_CSR_BUSY_Pos                  (24U)
#define    UART_CSR_BUSY_Msk                  (0x1U << UART_C
#define    UART_CSR_BUSY                      UART_CSR_BUSY_M

#define    UART_CSR_TXIREN_Pos                (17U)
#define    UART_CSR_TXIREN_Msk                (0x1U << UART_C
#define    UART_CSR_TXIREN                    UART_CSR_TXIREN

#define    UART_CSR_RXTOEN_Pos                (16U)
#define    UART_CSR_RXTOEN_Msk                (0x1U << UART_C
#define    UART_CSR_RXTOEN                    UART_CSR_RXTOEN
```

fm33lc

```
#define    UART_CSR_BUSY_Pos                  (21U)
#define    UART_CSR_BUSY_Msk                  (0x1U << UART_
#define    UART_CSR_BUSY                      UART_CSR_BUSY_

#define    UART_CSR_TXIREN_Pos                (17U)
#define    UART_CSR_TXIREN_Msk                (0x1U << UART_
#define    UART_CSR_TXIREN                    UART_CSR_TXIRE

#define    UART_CSR_RXTOEN_Pos                (16U)
#define    UART_CSR_RXTOEN_Msk                (0x1U << UART_
#define    UART_CSR_RXTOEN                    UART_CSR_RXTOE
```

5.10. U7816

◆fm33le0xx 删除了 U7816 模块的 U7816_CR 寄存器 bit1,IO 的强上拉使能。使用时可以使用 GPIO 模块中的上拉电阻或外部加上拉电阻。

fm33le0xx 中寄存器 U7816_CR：

位号	助记符	功能描述
3	CKOEN	U7816 时钟 CLK 输出使能控制位 (Clock output Enable) 1：7816 时钟输出使能 0：7816 时钟输出禁止
2:0	RFUI	保留位

fm33lc0xx 中寄存器 U7816_CR：

2	HPUAT	U7816 通道数据发送强上拉电阻自动有效控制位 (High-Pullup Automatically) 1：数据发送时上拉电阻自动有效，接收态上拉电阻无效 0：数据发送时上拉电阻自动有效功能禁止，上拉电阻由 HPUEN 控制
1	HPUEN	U7816 通道强上拉使能控制位 (High-Pullup Enable) 1：强上拉有效 0：强上拉无效
0	RFUI	保留位

fm33le0xx_fl_u7816.h 结构体中去除强上拉使能和自动上拉控制的相关定义

```
typedef struct
{
    /** 卡时钟频率 */
    uint32_t outputClockFreqence;

    /** 接口发送失败后自动重装使能 */
    uint32_t txAutoRetry;

    /** 自动重装次数 */
    uint32_t retryCnt;

    /** 插入BGT使能 */
    uint32_t blockGuard;

    /** 帧格式奇偶校验模式 */
    uint32_t parity;

    /** 帧格式接收Guard时间 */
    uint32_t rxGuardTime;

    /** 帧格式发送Guard时间 */
    uint32_t txGuardTime;
    /** 帧格式错误Guard时间 */
    uint32_t errorGuardTime;

    /** 帧错误信号长度 */
    uint32_t errorSignalWidth;

    /** 接收校验错误自动回发error signal使能 */
    uint32_t rxAutoErrorSignal;

    /** 传输BIT顺序 */
    uint32_t transferOrder;

    /** 通讯速率 */
    uint32_t baud;

    /** 额外发送Guard时间 */
    uint32_t extraGuardTime;

} FL_U7816_InitTypeDef;
```

1e

```
typedef struct
{
    /** 卡时钟频率 */
    uint32_t outputClockFreqence;

    /** 接口发送失败后自动重装使能 */
    uint32_t txAutoRetry;

    /** 自动重装次数 */
    uint32_t retryCnt;

    /** 插入BGT使能 */
    uint32_t blockGuard;

    /** 强上拉使能 */
    uint32_t strongPullUp;

    /** 帧格式奇偶校验模式 */
    uint32_t parity;

    /** 帧格式接收Guard时间 */
    uint32_t rxGuardTime;

    /** 帧格式发送Guard时间 */
    uint32_t txGuardTime;
    /** 帧格式错误Guard时间 */
    uint32_t errorGuardTime;

    /** 帧错误信号长度 */
    uint32_t errorSignalWidth;

    /** 接收校验错误自动回发error signal使能 */
    uint32_t rxAutoErrorSignal;

    /** 传输BIT顺序 */
    uint32_t transferOrder;

    /** 通讯速率 */
    uint32_t baud;

    /** 额外发送Guard时间 */
    uint32_t extraGuardTime;

} FL_U7816_InitTypeDef;
```

1c

fm33le0xx_fl_u7816.c 中 FL_ErrorStatus FL_U7816_Init()和 FL_U7816_StructInit()删除强上拉相关内容

```
assert_param(IS_FL_U7816_ERROR_SIGNALWIDTH(U7816_InitStruct->errorSignalWidth
assert_param(IS_FL_U7816_RX_AUTO_ERROR_SIG(U7816_InitStruct->rxAutoErrorSigna
assert_param(IS_FL_U7816_BIT_DIRECTION(U7816_InitStruct->transferOrder));

assert_param(IS_FL_U7816_TX_GUARD(U7816_InitStruct->txGuardTime));
/* 时钟使能 */
FL_RCC_EnableGroup3BusClock(FL_RCC_GROUP3_BUSCLK_U7816);
/* 卡时钟 */
Fclk = FL_RCC_GetAPB1ClockFreq();
tempClkdiv = Fclk / U7816_InitStruct->outputClockFreqence - 1;
FL_U7816_WriteClockDivision(U7816x, tempClkdiv);
/* 发送收到error signal后自动重发 */
if(U7816_InitStruct->txAutoRetry == FL_ENABLE)
{
    FL_U7816_EnableTXParityErrorAutoRetry(U7816x);
}
else
{
    FL_U7816_DisableTXParityErrorAutoRetry(U7816x);
}
/* 发送失败重试次数 */
FL_U7816_SetRetryCount(U7816x, U7816_InitStruct->retryCnt);
/* 发送一次之间的保护时间单位etu */
FL_U7816_SetTXGuardTime(U7816x, U7816_InitStruct->txGuardTime);

/* 块保护,插入block guard time */
if(U7816_InitStruct->blockGuard == FL_ENABLE)
{
    FL_U7816_EnableBlockGuardTime(U7816x);
}
else
```

1e

```
assert_param(IS_FL_U7816_ERROR_SIGNALWIDTH(U7816_InitStruct->errorSignalWidt
assert_param(IS_FL_U7816_RX_AUTO_ERROR_SIG(U7816_InitStruct->rxAutoErrorSig
assert_param(IS_FL_U7816_BIT_DIRECTION(U7816_InitStruct->transferOrder));
assert_param(IS_FL_U7816_AUTO_PULL(U7816_InitStruct->strongPullUp));
assert_param(IS_FL_U7816_TX_GUARD(U7816_InitStruct->txGuardTime));
/* 时钟使能 */
FL_RCC_EnableGroup3BusClock(FL_RCC_GROUP3_BUSCLK_U7816);
/* 卡时钟 */
Fclk = FL_RCC_GetAPB1ClockFreq();
tempClkdiv = Fclk / U7816_InitStruct->outputClockFreqence - 1;
FL_U7816_WriteClockDivision(U7816x, tempClkdiv);
/* 发送收到error signal后自动重发 */
if(U7816_InitStruct->txAutoRetry == FL_ENABLE)
{
    FL_U7816_EnableTXParityErrorAutoRetry(U7816x);
}
else
{
    FL_U7816_DisableTXParityErrorAutoRetry(U7816x);
}
/* 发送失败重试次数 */
FL_U7816_SetRetryCount(U7816x, U7816_InitStruct->retryCnt);
/* 发送一次之间的保护时间单位etu */
FL_U7816_SetTXGuardTime(U7816x, U7816_InitStruct->txGuardTime);
/* 强上拉*/
if(U7816_InitStruct->strongPullUp == FL_ENABLE)
{
    FL_U7816_EnablePullup(U7816x);
}
else
{
    FL_U7816_DisablePullup(U7816x);
}
/* 块保护,插入block guard time */
if(U7816_InitStruct->blockGuard == FL_ENABLE)
{
    FL_U7816_EnableBlockGuardTime(U7816x);
}
else
```

1c

```
void FL_U7816_StructInit(FL_U7816_InitTypeDef *U7816_InitStruct)
{
    U7816_InitStruct->outputClockFreqence   = 4000000;
    U7816_InitStruct->txAutoRetry           = FL_ENABLE;
    U7816_InitStruct->retryCnt              = FL_U7816_RETRY_COUNT_1;

    U7816_InitStruct->blockGuard            = FL_DISABLE;
    U7816_InitStruct->parity                = FL_U7816_PARITY_EVEN;
    U7816_InitStruct->rxGuardTime           = FL_U7816_RX_GUARD_TIME_2ETU;
    U7816_InitStruct->txGuardTime           = FL_U7816_TX_GUARD_TIME_2ETU;
    U7816_InitStruct->errorGuardTime        = FL_U7816_ERROR_GUARD_TIME_1ETU;
    U7816_InitStruct->errorSignalWidth      = FL_U7816_ERROR_SIGNAL_WIDTH_2ETU;
    U7816_InitStruct->rxAutoErrorSignal     = FL_ENABLE;
    U7816_InitStruct->transferOrder         = FL_U7816_BIT_ORDER_LSB_FIRST;
    U7816_InitStruct->baud                  = 372 - 1;
    U7816_InitStruct->extraGuardTime        = 0;
}
```

1e

```
void FL_U7816_StructInit(FL_U7816_InitTypeDef *U7816_InitStruct)
{
    U7816_InitStruct->outputClockFreqence   = 4000000;
    U7816_InitStruct->txAutoRetry           = FL_ENABLE;
    U7816_InitStruct->retryCnt              = FL_U7816_RETRY_COUNT_1;
    U7816_InitStruct->strongPullUp          = FL_ENABLE;
    U7816_InitStruct->blockGuard            = FL_DISABLE;
    U7816_InitStruct->parity                = FL_U7816_PARITY_EVEN;
    U7816_InitStruct->rxGuardTime           = FL_U7816_RX_GUARD_TIME_2ETU;
    U7816_InitStruct->txGuardTime           = FL_U7816_TX_GUARD_TIME_2ETU;
    U7816_InitStruct->errorGuardTime        = FL_U7816_ERROR_GUARD_TIME_1ETU;
    U7816_InitStruct->errorSignalWidth      = FL_U7816_ERROR_SIGNAL_WIDTH_2ETU;
    U7816_InitStruct->rxAutoErrorSignal     = FL_ENABLE;
    U7816_InitStruct->transferOrder         = FL_U7816_BIT_ORDER_LSB_FIRST;
    U7816_InitStruct->baud                  = 372 - 1;
    U7816_InitStruct->extraGuardTime        = 0;
}
```

1c

◆U7816 输出时钟范围修改，fm33lc0xx 驱动后续同样会更新

fm33le0xx_fl_u7816.c:

```
#define         IS_FL_U7816_CLOCK_FRQUENCE(__VALUE__)               (((__VALUE__) >=1000000)&&\
                                                                     ((__VALUE__) <= 5000000))
```

fm33lc0xx_fl_u7816.c:

```
#define         IS_FL_U7816_CLOCK_FRQUENCE(__VALUE__)               (((__VALUE__) >=1000000)||\
                                                                     ((__VALUE__) <= 5000000))
```

5.11. ATIM 与 GTIM

◆ATIM、GTIM 在功能上略做修改，输出比较模式寄存器 OCxM 中两个功能有变化需要注意：CCMR1 和 CCMR2 寄存器：

6:4	**OC1M**	输出比较 1 模式配置,此寄存器定义 OC1REF 信号的行为 (OC1 Mode) 000：输出比较寄存器 CCR1 和计数器 CNT 的比较结果不会影响输出 001：CCR1=CNT 时，将 OC1REF 置高 010：CCR1=CNT 时，将 OC1REF 置低 011：CCR1=CNT 时，翻转 OC1REF 100：OC1REF 固定为低（inactive） 101：OC1REF 固定为高（active） 110：PWM 模式 1 –在向上计数时，OC1REF 在 CNT<CCR1 时置高，否则置低；在向下计数时，OC1REF 在 CNT>CCR1 时置低，否则置高 111：PWM 模式 2 –在向上计数时，OC1REF 在 CNT<CCR1 时置低，否则置高；在向下计数时，OC1REF 在 CNT>CCR1 时置高，否则置低

001 CCR=CNT 时，将 OCREF 置高
010 CCR=CNT 时，将 OCREF 置低

FM33LC0xx，当 CNT=CCR 时 OCREF 置高或置低后立刻拉高或拉低,是一个窄脉冲
FM33LE0xx，当 CNT=CCR 时 OCREF 置高或置低后一直保持
其余完全相同

◆UDIS 的禁用更新事件函数需要修正，fm33lc0xx 驱动后续同样会更新
fm33le0xx_fl_atim.h 与 fm33lc0xx_fl_atim.h

fm33le
```
/**
  * @brief    禁用更新事件
  * @rmtoll   CR1    UDIS    FL_ATIM_DisableUpdateEvent
  * @param    TIMx TIM instance
  * @retval   None
  */
__STATIC_INLINE void FL_ATIM_DisableUpdateEvent(ATIM_Type *TIMx)
{
    SET_BIT(TIMx->CR1, ATIM_CR1_UDIS_Msk);
}
```

fm33lc
```
/**
  * @brief    禁用更新事件
  * @rmtoll   CR1    UDIS    FL_ATIM_DisableUpdateEvent
  * @param    TIMx TIM instance
  * @retval   None
  */
__STATIC_INLINE void FL_ATIM_DisableUpdateEvent(ATIM_Type *TIMx)
{
    CLEAR_BIT(TIMx->CR1, ATIM_CR1_UDIS_Msk);
}
```

BSTIM32 也有此问题，在此处一并说明，fm33lc0xx 驱动后续同样会更新

fm33le
```
/**
  * @brief    Update event disable
  * @rmtoll   CR1    UDIS    FL_BSTIM32_DisableUpdateEvent
  * @param    BSTIM32x BSTIM32 instance
  * @retval   None
  */
__STATIC_INLINE void FL_BSTIM32_DisableUpdateEvent(BSTIM32_Type *BSTIM32x)
{
    SET_BIT(BSTIM32x->CR1, BSTIM32_CR1_UDIS_Msk);
}
```

fm33lc
```
/**
  * @brief    Update event disable
  * @rmtoll   CR1    UDIS    FL_BSTIM32_DisableUpdateEvent
  * @param    BSTIM32x BSTIM32 instance
  * @retval   None
  */
__STATIC_INLINE void FL_BSTIM32_DisableUpdateEvent(BSTIM32_Type *BSTIM32x)
{
    CLEAR_BIT(BSTIM32x->CR1, BSTIM32_CR1_UDIS_Msk);
}
```

◆fm33lc0xx_fl_atim.h 中读写 DMA 地址函数需要修正，后续 fm33lc0xx_fl_atim.h 通样会进行更新

fm33le
```
/**
  * @brief    配置DMA burst访问寄存器
  * @rmtoll   DMAR        FL_ATIM_WriteDMAAddress
  * @param    TIMx TIM instance
  * @param    address
  * @retval   None
  */
__STATIC_INLINE void FL_ATIM_WriteDMAAddress(ATIM_Type *TIMx, uint32_t address)
{
    MODIFY_REG(TIMx->DMAR, (0xffffffffU << 0U), (address << 0U));
}

/**
  * @brief    读取DMA burst访问寄存器值
  * @rmtoll   DMAR        FL_ATIM_ReadDMAAddress
  * @param    TIMx TIM instance
  * @retval
  */
__STATIC_INLINE uint32_t FL_ATIM_ReadDMAAddress(ATIM_Type *TIMx)
{
    return (uint32_t)(READ_BIT(TIMx->DMAR, 0xffffffffU) >> 0U);
}
```

fm33lc
```
/**
  * @brief    配置DMA burst访问寄存器
  * @rmtoll   DMAR        FL_ATIM_WriteDMAAddress
  * @param    TIMx TIM instance
  * @param    address
  * @retval   None
  */
__STATIC_INLINE void FL_ATIM_WriteDMAAddress(ATIM_Type *TIMx, uint32_t address)
{
    MODIFY_REG(TIMx->DMAR, (0xffffU << 0U), (address << 0U));
}

/**
  * @brief    读取DMA burst访问寄存器值
  * @rmtoll   DMAR        FL_ATIM_ReadDMAAddress
  * @param    TIMx TIM instance
  * @retval
  */
__STATIC_INLINE uint32_t FL_ATIM_ReadDMAAddress(ATIM_Type *TIMx)
{
    return (uint32_t)(READ_BIT(TIMx->DMAR, 0xffffU) >> 0U);
}
```

5.12. RTC

RTC 初始化函数需要修正，fm33lc0xx 驱动后续同样会更新
fm33le0xx_fl_rtc.c 与 fm33lc0xx_fl_rtc.c:

fm331e

```
FL_ErrorStatus FL_RTC_DeInit(RTC_Type *RTCx)
{
    FL_ErrorStatus result = FL_PASS;
    /* Check the parameters */
    assert_param(IS_RTC_INSTANCE(RTCx));
    RTCx->IER       = 0x00000000U;
    RTCx->WER       = 0xACACACACU;
    RTCx->ADJUST    = 0x00000000U;
    RTCx->ADSIGN    = 0x00000000U;
    RTCx->ALARM     = 0x00000000U;
    RTCx->BCDDAY    = 0x00000000U;
    RTCx->BCDHOUR   = 0x00000000U;
    RTCx->BCDMIN    = 0x00000000U;
    RTCx->BCDMONTH  = 0x00000000U;
    RTCx->BCDSEC    = 0x00000000U;
    RTCx->BCDWEEK   = 0x00000000U;
    RTCx->BCDYEAR   = 0x00000000U;
    RTCx->SBSCNT    = 0x00000000U;
    RTCx->TMSEL     = 0x00000000U;
    RTCx->WER       = 0x00000000U;
    return result;
}
```

fm331c

```
FL_ErrorStatus FL_RTC_DeInit(RTC_Type *RTCx)
{
    FL_ErrorStatus result = FL_PASS;
    /* Check the parameters */
    assert_param(IS_RTC_INSTANCE(RTCx));
    RTCx->IER       = 0x00000000U;
    RTCx->WER       = 0x00000000U;
    RTCx->ADJUST    = 0x00000000U;
    RTCx->ADSIGN    = 0x00000000U;
    RTCx->ALARM     = 0x00000000U;
    RTCx->BCDDAY    = 0x00000000U;
    RTCx->BCDHOUR   = 0x00000000U;
    RTCx->BCDMIN    = 0x00000000U;
    RTCx->BCDMONTH  = 0x00000000U;
    RTCx->BCDSEC    = 0x00000000U;
    RTCx->BCDWEEK   = 0x00000000U;
    RTCx->BCDYEAR   = 0x00000000U;
    RTCx->SBSCNT    = 0x00000000U;
    RTCx->TMSEL     = 0x00000000U;
    RTCx->WER       = 0x00000000U;
    return result;
}
```

5.13. ADC

fm331e0xx 的 ADC 与 fm331c0xx 相比有较大的变动，建议完全替换驱动：

◆fm331e0xx 增加 ADC 校准（Calibration），使用前需要执行一次较准

fm331e0xx 中寄存器 ADC_CALR：

0	CALEN	Offset Calibration 使能 (Offset Calibration Enable) 软件写 1 启动校准周期，校准结束后自动清零并置位 EOCAL 寄存器。 Offset 校准分为比较器校准和通道校准，由 CALSEL 寄存器选择校准项目。两种校准都是由 CALEN 寄存器启动的。

fm331e0xx_fl_adc.c 中 FL_ADC_Init()增加 ADC 校准（Calibration）逻辑

```
    FL_VREF_EnableTemperatureSensor(VREF);
    FL_ADC_Enable(ADCx);
    FL_ADC_EnableCalibration(ADC);
    i = 0;
    do
    {
        Calibration_Flag = FL_ADC_IsActiveFlag_EndOfCalibration(ADC);
        i++;
    }while((i != 0xFFFFFFFFU) && (Calibration_Flag == 0U));          //等待转换完成
```

◆Fm331e0xx 删掉外部引脚转换功能相关函数接口

Fm33lc0xx 中寄存器 ADC_CR:

31:10	-	RFU：未实现，读为 0
9	EXSAMP	外部引脚控制采样时间 (External Sample time control) 1：由 GPIO 输入信号来控制 ADC 采样时间 0：由寄存器控制 ADC 采样时间
8	EXSYNC	外部引脚控制采样使能 (External Synchronization enable) 1：由 GPIO 输入信号启动 ADC 采样 0：由 START 寄存器启动 ADC 采样
7:2		RFU：未实现，读为 0

◆fm331e0xx_fl_adc.h 修改 ADC 启动转换寄存器名，函数调用名未变

```
__STATIC_INLINE void FL_ADC_EnableSWConversion(ADC_Type *ADCx)
{
    SET_BIT(ADCx->CR, ADC_CR_SWTRIG_Msk);
}
```

```
__STATIC_INLINE void FL_ADC_EnableSWConversion(ADC_Type *ADCx)
{
    SET_BIT(ADCx->CR, ADC_CR_START_Msk);
}
```

◆fm33le0xx 删掉软件控制采样函数接口

fm33lc0xx 中寄存器 ADC_SAMPT：

位号	助记符	功能描述
31:1	-	RFU：未实现，读为 0
0	SAMPT_S	软件控制采样信号，仅在 SMTSx=1101/1110/1111 时有效 (Sample time software control) 0：ADC 采样 1：ADC 停止采样

◆fm33le0xx 删掉采样通道切换等待时间函数接口

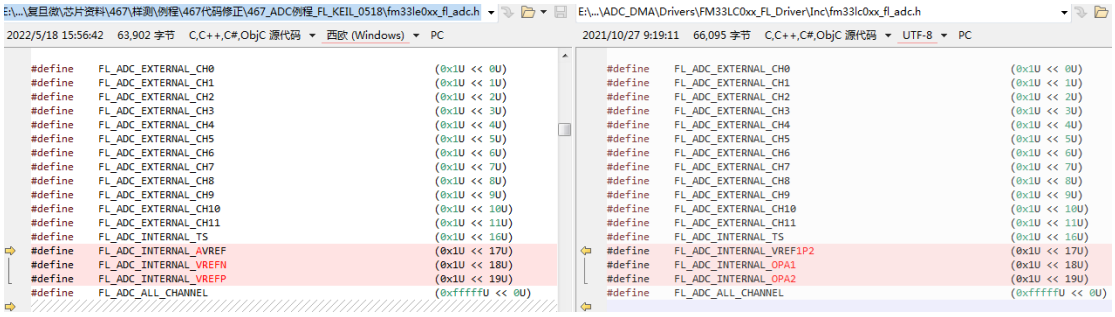
fm33lc0xx 中寄存器 ADC_SMTR：

位号	助记符	功能描述
31:12	-	RFU：未实现，读为 0
11:8	CHCG	ADC 采样通道切换等待时间，在当前通道采样周期完成后，等待 CHCG 时间（CHCG*ADC 工作时钟周期），再切换到下一个采样通道(Channel Clock Gating) 0000, 0001, 0010：2*T_{ADCLK} 0011：3*T_{ADCLK} 0100：4*T_{ADCLK} 0101：5*T_{ADCLK} 0110：6*T_{ADCLK} 0111：7*T_{ADCLK} 1000：8*T_{ADCLK} 1001：9*T_{ADCLK} 1010：10*T_{ADCLK} 1011~1111：11*T_{ADCLK}

fm33le0xx 删除采样通道切换等待时间以及外部引脚转换功能配置

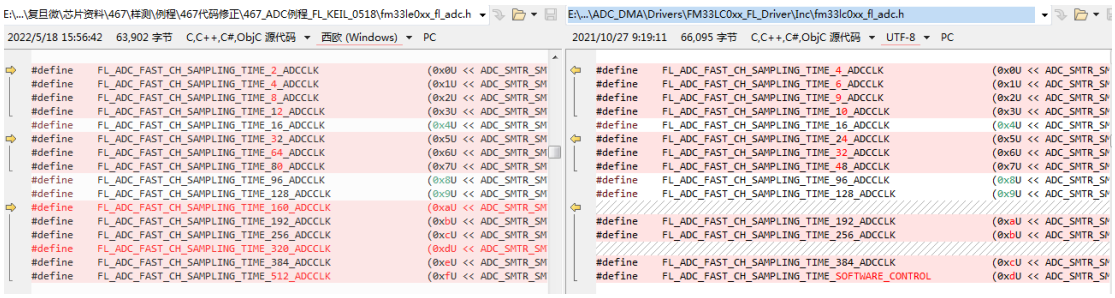
◆fm33le0xx 删掉内部 VREFP1P2、OPA 通道，增加 AVREF 通道

fm33le0xx_fl_adc.h 与 fm33lc0xx_fl_adc.h:



◆fm33le0xx 与 fm33lc0xx 采样时间配置选项不一样

fm33le0xx_fl_adc.h 与 fm33lc0xx_fl_adc.h:



◆fm33le0xx.c 删除 FL_ADC_Init 初始化中 VREFP1P2 使能逻辑

fm33le0xx 无 VREFP1P2，所以无下图软件逻辑

fm33lc0xx_fl_adc.c 中 FL_ADC_Init()函数

```
    if(!FL_VREF_IsEnabled(VREF))
    {
        FL_VREF_ClearFlag_Ready(VREF);
        FL_VREF_Enable(VREF);    /* 置位VREF_EN寄存器，使能VREF1p2模块 */
    }
    FL_VREF_EnableTemperatureSensor(VREF);  /* 置位PTAT_EN寄存器 */
    while(FL_VREF_IsActiveFlag_Ready(VREF) == 0)
    {
        if(i >= 128000)
        {
            break;
        }
        i++;
    }
```

例程：

Fm33le0xx 新增了使用简化公式计算信号电压值的例程：ADC 查询（简化计算公式）

5.14. GPIO

FL_WKUP_DeInit()函数需要修正，fm33lc0xx 驱动后续同样会更新

fm33le0xx_fl_gpio.c 与 fm33lc0xx_fl_gpio.c:

fm33le

```
FL_ErrorStatus FL_WKUP_DeInit(uint32_t wkupx)
{
    /* 入口参数检查 */
    assert_param(IS_FL_GPIO_WKUP_NUM(wkupx));
    FL_GPIO_DisableWakeup(GPIO, wkupx);
    return FL_PASS;
}
```

fm33lc

```
FL_ErrorStatus FL_WKUP_DeInit(uint32_t wkupx)
{
    /* 入口参数检查 */
    assert_param(IS_FL_GPIO_WKUP_NUM(wkupx));
    FL_GPIO_EnableWakeup(GPIO, wkupx);
    return FL_PASS;
}
```

5.15. 工程文件

◆fm33le0xx.h 和 fm33lc0xx.h

在 FM33LC0xx 的工程中，将 fm33lc0xx.h 中关于 ADC 和 VREF 的寄存器定义替换为 fm33le0xx.h 中的定义

VREF ：

fm33le

```
typedef struct
{
  __IO uint32_t PTATCR;             /*!< PTAT Control Register                */
  __IO uint32_t RSV0;               /*!< RESERVED REGISTER                    */
  __IO uint32_t RSV1;               /*!< RESERVED REGISTER    */
  __IO uint32_t BUFCR;              /*!< VREF BUFFER Control Register    */
} VREF_Type;
```

fm33lc

```
typedef struct
{
  __IO uint32_t CR;             /*!< VREF Control Register            */
  __IO uint32_t SR;             /*!< VREF Status Register             */
  __IO uint32_t IER;            /*!< VREF Interrupt Enable Register   */
  __IO uint32_t BUFCR;          /*!< VREF Interrupt Enable Register   */
} VREF_Type;
```

ADC:

fm33le

```
typedef struct
{
  __IO uint32_t ISR;            /*!< ADC Interrupt and Status Register,
  __IO uint32_t IER;            /*!< ADC Interrupt Enable Register,
  __IO uint32_t CR;             /*!< ADC Control Register,
  __IO uint32_t CALR;           /*!< ADC Calibration Register,
  __IO uint32_t CFGR;           /*!< ADC Config Register,
  __IO uint32_t SMTR;           /*!< ADC Sampling Time Register,
  __IO uint32_t CHER;           /*!< ADC Channel Enable Register,
  __IO uint32_t DR;             /*!< ADC Data Register,

  __IO uint32_t HLTR;           /*!< ADC Analog watchdog Threshold Register,
} ADC_Type;
```

fm33lc

```
typedef struct
{
  __IO uint32_t ISR;            /*!< ADC Interrupt and Status Register,
  __IO uint32_t IER;            /*!< ADC Interrupt Enable Register,
  __IO uint32_t CR;             /*!< ADC Control Register,

  __IO uint32_t CFGR;           /*!< ADC Config Register,
  __IO uint32_t SMTR;           /*!< ADC Sampling Time Register,
  __IO uint32_t CHER;           /*!< ADC Channel Enable Register,
  __IO uint32_t DR;             /*!< ADC Data Register,
  __IO uint32_t SAMPT;          /*!< ADC Calibration Register,
  __IO uint32_t HLTR;           /*!< ADC analog watchdog Threshold Register,
} ADC_Type;
```

- ◆ system_fm33lc0xx.h 不需要修改
- ◆ startup_fm33lc0xx.s 不需要修改
- ◆ system_fm33lc0xx.c

SystemInit()中 BORCR 寄存器的值修改，目的都是使能 BOR，并设置阈值为 1.75V。用户假如此处不修改 BORCR 寄存器的赋值仍为 0x0E，就是关闭 BOR 下电复位，因为打开了 PDR 下电复位，不影响芯片的工作。

system_fm33le0xx.c 与 system_fm33lc0xx.c:

fm33le

```
#endif  /* USE_LSCLK_CLOCK_SRC_XTLF */
#endif  /* MFANG */

/* PDR & BOR Configuration */
RMU->PDRCR = 0x1U;
RMU->BORCR = 0xCU;

/* Disable IWDT & WWDT, enable other peripherals(e.g. timers) under Debug Mod
DBG->CR = 0x3U;

/* Load clock trim value */
RCC->RCHFTR = RCHF8M_TRIM;
RCC->RCMFTR = RCMF4M_TRIM;
RCC->LPOSCTR = LPOSC_TRIM;
```

fm33lc

```
#endif  /* USE_LSCLK_CLOCK_SRC_XTLF */
#endif  /* MFANG */

/* PDR & BOR Configuration */
RMU->PDRCR = 0x1U;
RMU->BORCR = 0xEU;

/* Disable IWDT & WWDT, enable other peripherals(e.g. timers) under Debug M
DBG->CR = 0x3U;

/* Load clock trim value */
RCC->RCHFTR = RCHF8M_TRIM;
RCC->RCMFTR = RCMF4M_TRIM;
RCC->LPOSCTR = LPOSC_TRIM;
```

SystemCoreClockUpdate()函数中 LPOSC 的值修改为了 32768，这个函数是用于工程中 systick 延时函数用于获取系统时钟，系统中没有使用 LSCLK 或 LPOSC 做主时钟，此处不用修改

system_fm33le0xx.c 与 system_fm33lc0xx.c:

fm33le

```
void SystemCoreClockUpdate(void)
{
    switch ((RCC->SYSCLKCR >> 0) & 0x7)
    {
        case 1: /* XTHF */
            SystemCoreClock = XTHFClock;
            break;

        case 2: /* PLL */
            SystemCoreClock = SystemPLLClockUpdate();
            break;

        case 4: /* RCMF */
            switch ((RCC->RCMFCR >> 16) & 0x3)
            {
                case 0: /* output divided by 1 */
                    SystemCoreClock = 4000000;
                    break;

                case 1: /* output divided by 4 */
                    SystemCoreClock = 1000000;
                    break;

                case 2: /* output divided by 8 */
                    SystemCoreClock = 500000;
                    break;

                case 3: /* output divided by 16 */
                    SystemCoreClock = 250000;
                    break;
            }
            break;

        case 5: /* LSCLK */
            #ifdef USE_LSCLK_CLOCK_SRC_LPOSC
                SystemCoreClock = 32768;
            #else
                SystemCoreClock = XTLFClock;
            #endif
            break;

        case 6: /* LPOSC */
            SystemCoreClock = 32768;
```

fm33lc

```
void SystemCoreClockUpdate(void)
{
    switch ((RCC->SYSCLKCR >> 0) & 0x7)
    {
        case 1: /* XTHF */
            SystemCoreClock = XTHFClock;
            break;

        case 2: /* PLL */
            SystemCoreClock = SystemPLLClockUpdate();
            break;

        case 4: /* RCMF */
            switch ((RCC->RCMFCR >> 16) & 0x3)
            {
                case 0: /* output divided by 1 */
                    SystemCoreClock = 4000000;
                    break;

                case 1: /* output divided by 4 */
                    SystemCoreClock = 1000000;
                    break;

                case 2: /* output divided by 8 */
                    SystemCoreClock = 500000;
                    break;

                case 3: /* output divided by 16 */
                    SystemCoreClock = 250000;
                    break;
            }
            break;

        case 5: /* LSCLK */
            #ifdef USE_LSCLK_CLOCK_SRC_LPOSC
                SystemCoreClock = 32000;
            #else
                SystemCoreClock = XTLFClock;
            #endif
            break;

        case 6: /* LPOSC */
            SystemCoreClock = 32000;
            break;
```