Appendix A Code examples

A.1 Introduction

This appendix shows the code examples of the sequences described in this document.

These code examples are extracted from the STM32F0xx Snippet firmware package **STM32SnippetsF0** available on www.st.com.

These code examples use the peripheral bit and register description from the CMSIS header file (stm32f0xx.h).

Code lines starting with // should be uncommented if the given register has been modified before.

A.2 FLASH operation code examples

A.2.1 Flash memory unlocking sequence

```
/* (1) Wait till no operation is on going */
/* (2) Check that the flash memory is unlocked */
/* (3) Perform unlock sequence */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
  /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
  FLASH->KEYR = FLASH_FKEY1; /* (3) */
  FLASH->KEYR = FLASH_FKEY2;
}
```

A.2.2 Main flash memory programming sequence

```
/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write (half-word) at the desired address */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) clear it by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH->CR |= FLASH_CR_PG; /* (1) */
*(__IO uint16_t*)(flash_addr) = data; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
  /* For robust implementation, add here time-out management */
}
```

```
if ((FLASH->SR & FLASH_SR_EOP) != 0)  /* (4) */
{
  FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
  /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PG; /* (6) */
```

A.2.3 Page erase sequence

```
/* (1) Set the PER bit in the FLASH_CR register to enable page erasing */
/* (2) Program the FLASH_AR register to select a page to erase */
/* (3) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (4) Wait until the BSY bit is reset in the FLASH_SR register */
/* (5) Check the EOP flag in the FLASH_SR register */
/* (6) Clear EOP flag by software by writing EOP at 1 */
/* (7) Reset the PER Bit to disable the page erase */
FLASH->CR |= FLASH_CR_PER; /* (1) */
FLASH->AR =  page_addr; /* (2) */
FLASH->CR |= FLASH_CR_STRT; /* (3) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (4) */
{
  /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0)  /* (5) */
{
  FLASH->SR = FLASH_SR_EOP; /* (6)*/
}
else
{
  /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PER; /* (7) */
```

A.2.4 Mass erase sequence

```
/* (1) Set the MER bit in the FLASH_CR register to enable mass erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the mass erase */
FLASH->CR |= FLASH_CR_MER; /* (1) */
FLASH->CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
  /* For robust implementation, add here time-out management */
}

if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4)*/
{
  FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
  /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_MER; /* (6) */
```

A.2.5 Option byte unlocking sequence

```
/* (1) Wait till no operation is on going */
/* (2) Check that the flash memory is unlocked */
/* (3) Perform unlock sequence for flash memory */
/* (4) Check that the Option Bytes are unlocked */
/* (5) Perform unlock sequence for Option Bytes */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
  /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
  FLASH->KEYR = FLASH_FKEY1; /* (3) */
  FLASH->KEYR = FLASH_FKEY2;
}
if ((FLASH->CR & FLASH_CR_OPTWRE) == 0) /* (4) */
{
  FLASH->OPTKEYR = FLASH_OPTKEY1; /* (5) */
  FLASH->OPTKEYR = FLASH_OPTKEY2;
}
```

A.2.6 Option byte programming sequence

```
/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear the EOP flag by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH->CR |= FLASH_CR_OPTPG; /* (1) */
*opt_addr = data; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
  /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0)  /* (4) */
{
  FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
  /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_OPTPG; /* (6) */
```

A.2.7 Option byte erasing sequence

```
/* (1) Set the OPTER bit in the FLASH_CR register to enable option byte
       erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the page erase */
FLASH->CR |= FLASH_CR_OPTER; /* (1) */
FLASH->CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
  /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0)  /* (4) */
{
  FLASH->SR = FLASH_SR_EOP; /* (5)*/
}
else
{
  /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_OPTER; /* (6) */
```

A.3 Clock controller code examples

A.3.1 HSE start sequence

```
/**
  * Description: This function enables the interrupt on HSE ready,
  *              and start the HSE as external clock.
  */
__INLINE void StartHSE(void)
{
  /* Configure NVIC for RCC */
  /* (1) Enable Interrupt on RCC */
  /* (2) Set priority for RCC */
  NVIC_EnableIRQ(RCC_CRS_IRQn); /* (1)*/
  NVIC_SetPriority(RCC_CRS_IRQn,0); /* (2) */

  /* (1) Enable interrupt on HSE ready */
  /* (2) Enable the CSS
         Enable the HSE and set HSEBYP to use the external clock
         instead of an oscillator
         Enable HSE */
  /* Note : the clock is switched to HSE in the RCC_CRS_IRQHandler ISR */
  RCC->CIR |= RCC_CIR_HSERDYIE; /* (1) */
  RCC->CR |= RCC_CR_CSSON | RCC_CR_HSEBYP | RCC_CR_HSEON; /* (2) */
}

/**
  * Description: This function handles RCC interrupt request
  *              and switch the system clock to HSE.
  */
void RCC_CRS_IRQHandler(void)
{
  /* (1) Check the flag HSE ready */
  /* (2) Clear the flag HSE ready */
  /* (3) Switch the system clock to HSE */

  if ((RCC->CIR & RCC_CIR_HSERDYF) != 0) /* (1) */
  {
    RCC->CIR |= RCC_CIR_HSERDYC; /* (2) */
    RCC->CFGR = ((RCC->CFGR & (~RCC_CFGR_SW)) | RCC_CFGR_SW_0); /* (3) */
  }
  else
  {
    /* Report an error */
  }
}
```

A.3.2 PLL configuration modification

```
/* (1)  Test if PLL is used as System clock */
/* (2)  Select HSI as system clock */
/* (3)  Wait for HSI switched */
/* (4)  Disable the PLL */
/* (5)  Wait until PLLRDY is cleared */
/* (6)  Set the PLL multiplier to 6 */
/* (7)  Enable the PLL */
/* (8)  Wait until PLLRDY is set */
/* (9)  Select PLL as system clock */
/* (10) Wait until the PLL is switched on */
if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_PLL) /* (1) */
{
  RCC->CFGR &= (uint32_t) (~RCC_CFGR_SW); /* (2) */
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI) /* (3) */
  {
    /* For robust implementation, add here time-out management */
  }
}
RCC->CR &= (uint32_t)(~RCC_CR_PLLON);/* (4) */
while((RCC->CR & RCC_CR_PLLRDY) != 0) /* (5) */
{
  /* For robust implementation, add here time-out management */
}
RCC->CFGR = RCC->CFGR & (~RCC_CFGR_PLLMUL) | (RCC_CFGR_PLLMUL6); /* (6) */
RCC->CR |= RCC_CR_PLLON; /* (7) */
while((RCC->CR & RCC_CR_PLLRDY) == 0) /* (8) */
{
  /* For robust implementation, add here time-out management */
}
RCC->CFGR |= (uint32_t) (RCC_CFGR_SW_PLL); /* (9) */
while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL) /* (10) */
{
  /* For robust implementation, add here time-out management */
}
```

A.3.3 MCO selection

```
/* Select system clock to be output on the MCO without prescaler */
RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;
```

A.3.4 Clock measurement configuration with TIM14

```
/**
  * Description: This function configures the TIM14 as input capture
  *              and enables the interrupt on TIM14
  */
__INLINE void ConfigureTIM14asInputCapture(void)
{
  /* (1) Enable the peripheral clock of Timer 14 */
  /* (2) Select the active input TI1,Program the input filter, and prescaler
         */
  /* (3) Enable interrupt on Capture/Compare */
  RCC->APB1ENR |= RCC_APB1ENR_TIM14EN; /* (1) */
  TIM14->CCMR1 |= TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1 \
                | TIM_CCMR1_CC1S_0 | TIM_CCMR1_IC1PSC_1; /* (2)*/
  TIM14->DIER |= TIM_DIER_CC1IE; /* (3) */

  /* Configure NVIC for TIM14 */
  /* (4) Enable Interrupt on TIM14 */
  /* (5) Set priority for TIM14 */
  NVIC_EnableIRQ(TIM14_IRQn); /* (4) */
  NVIC_SetPriority(TIM14_IRQn,0); /* (5) */

  /* (6) Select HSE/32 as input on TI1 */
  /* (7) Enable counter */
  /* (8) Enable capture */
  TIM14->OR |= TIM14_OR_TI1_RMP_1; /* (6) */
  TIM14->CR1 |= TIM_CR1_CEN; /* (7) */
  TIM14->CCER |= TIM_CCER_CC1E; /* (8) */
}
```

Note: The measurement is done in the TIM14 interrupt subroutine.

A.4 GPIO code examples

A.4.1 Lock sequence

```
/**
  * Description: This function locks the targeted pins of Port A
                 configuration
                 This function can be easily modified to lock Port B
  * Parameter: lock contains the port pin mask to be locked
  */
void LockGPIOA(uint16_t lock)
{
  /* (1) Write LCKK bit to 1 and set the pin bits to lock */
  /* (2) Write LCKK bit to 0 and set the pin bits to lock */
  /* (3) Write LCKK bit to 1 and set the pin bits to lock */
  /* (4) Read the Lock register */
  /* (5) Check the Lock register (optionnal) */
  GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (1) */
  GPIOA->LCKR = lock; /* (2) */
  GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (3) */
  GPIOA->LCKR; /* (4) */
  if ((GPIOA->LCKR & GPIO_LCKR_LCKK) == 0) /* (5) */
  {
    /* Manage an error */
  }
}
```

A.4.2 Alternate function selection sequence

```
/* This sequence select AF2 for GPIOA4, 8 and 9. This can be easily adapted
   with another port by changing all GPIOA references by another GPIO port,
   and the alternate function number can be changed by replacing 0x04 or
0x02 for
   each pin by the targeted alternate function in the 2 last code lines. */
/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select alternate function mode on GPIOA pin 4, 8 and 9 */
/* (3) Select AF4 on PA4 in AFRL for TIM14_CH1 */
/* (4) Select AF2 on PA8 and PA9 in AFRH for TIM1_CH1 and TIM1_CH2 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER4 | GPIO_MODER_MODER8
| GPIO_MODER_MODER9)) | GPIO_MODER_MODER4_1
| GPIO_MODER_MODER8_1 | GPIO_MODER_MODER9_1; /* (2) */
GPIOA->AFR[0] |= 0x04 << GPIO_AFRL_AFRL4_Pos; /* (3) */
GPIOA->AFR[1] |= (0x02 << GPIO_AFRL_AFRH8_Pos) | (0x02 <<
GPIO_AFRL_AFRH9_Pos); /* (4) */
```

A.4.3 Analog GPIO configuration

```
/* (1) Enable the peripheral clock of GPIOA, GPIOB and GPIOC */
/* (2) Select analog mode for PA1 */
/* (3) Select analog mode for PB1 */
/* (4) Select analog mode for PC0 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN
             | RCC_AHBENR_GPIOCEN; /* (1) */
GPIOA->MODER |= GPIO_MODER_MODER1; /* (2) */
GPIOB->MODER |= GPIO_MODER_MODER1; /* (3) */
GPIOC->MODER |= GPIO_MODER_MODER0; /* (4) */
```

A.5 DMA code examples

A.5.1 DMA channel configuration sequence

```
/* The following example is given for the ADC. It can be easily ported on
   any peripheral supporting DMA transfer taking of the associated channel
   to the peripheral, this must check in the datasheet. */
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs on channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = 3; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                    | DMA_CCR_TEIE | DMA_CCR_TCIE ; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
/* Configure NVIC for DMA */
/* (1) Enable Interrupt on DMA Channel 1 */
/* (2) Set priority for DMA Channel 1 */
NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* (1) */
NVIC_SetPriority(DMA1_Channel1_IRQn,0); /* (2) */
```

A.6 Interrupts and event code examples

A.6.1 NVIC initialization

```
/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC to 2*/
NVIC_EnableIRQ(ADC1_IRQn); /* (1) */
NVIC_SetPriority(ADC1_IRQn,2); /* (2) */
```

A.6.2 External interrupt selection

```
/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select Port A for pin 0 external interrupt by writing 0000 in
       EXTI0 (reset value)*/
/* (3) Configure the corresponding mask bit in the EXTI_IMR register */
/* (4) Configure the Trigger Selection bits of the Interrupt line on
       rising edge*/
/* (5) Configure the Trigger Selection bits of the Interrupt line on
       falling edge*/
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
//SYSCFG->EXTICR[1] &= (uint16_t)~SYSCFG_EXTICR1_EXTI0_PA; /* (2) */
EXTI->IMR = 0x0001; /* (3) */
EXTI->RTSR = 0x0001; /* (4) */
EXTI->FTSR = 0x0001; /* (5) */
/* Configure NVIC for External Interrupt */
/* (1) Enable Interrupt on EXTI0_1 */
/* (2) Set priority for EXTI0_1 */
NVIC_EnableIRQ(EXTI0_1_IRQn); /* (1) */
NVIC_SetPriority(EXTI0_1_IRQn,0); /* (2) */
```

A.7 ADC code examples

A.7.1 ADC calibration

```
/* (1) Ensure that ADEN = 0 */
/* (2) Clear ADEN by setting ADDIS*/
/* (3) Clear DMAEN */
/* (4) Launch the calibration by setting ADCAL */
/* (5) Wait until ADCAL=0 */
if ((ADC1->CR & ADC_CR_ADEN) != 0) /* (1) */
{
  ADC1->CR |= ADC_CR_ADDIS; /* (2) */
}
while ((ADC1->CR & ADC_CR_ADEN) != 0)
{
/* For robust implementation, add here time-out management */
}
ADC1->CFGR1 &= ~ADC_CFGR1_DMAEN; /* (3) */
ADC1->CR |= ADC_CR_ADCAL; /* (4) */
while ((ADC1->CR & ADC_CR_ADCAL) != 0) /* (5) */
{
  /* For robust implementation, add here time-out management */
}
```

A.7.2 ADC enable sequence

```
/* (1) Ensure that ADRDY = 0 */
/* (2) Clear ADRDY */
/* (3) Enable the ADC */
/* (4) Wait until ADC ready */
if ((ADC1->ISR & ADC_ISR_ADRDY) != 0) /* (1) */
{
  ADC1->ISR |= ADC_CR_ADRDY; /* (2) */
}
ADC1->CR |= ADC_CR_ADEN; /* (3) */
while ((ADC1->ISR & ADC_ISR_ADRDY) == 0) /* (4) */
{
  /* For robust implementation, add here time-out management */
}
```

A.7.3 ADC disable sequence

```
/* (1) Stop any ongoing conversion */
/* (2) Wait until ADSTP is reset by hardware i.e. conversion is stopped */
/* (3) Disable the ADC */
/* (4) Wait until the ADC is fully disabled */
ADC1->CR |= ADC_CR_ADSTP; /* (1) */
while ((ADC1->CR & ADC_CR_ADSTP) != 0) /* (2) */
{
  /* For robust implementation, add here time-out management */
}
ADC1->CR |= ADC_CR_ADDIS; /* (3) */
while ((ADC1->CR & ADC_CR_ADEN) != 0) /* (4) */
{
  /* For robust implementation, add here time-out management */
}
```

A.7.4 ADC clock selection

```
/* This code selects the HSI14 as clock source. */
/* (1) Enable the peripheral clock of the ADC */
/* (2) Start HSI14 RC oscillator */
/* (3) Wait HSI14 is ready */
/* (4) Select HSI14 by writing 00 in CKMODE (reset value) */
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; /* (1) */
RCC->CR2 |= RCC_CR2_HSI14ON; /* (2) */
while ((RCC->CR2 & RCC_CR2_HSI14RDY) == 0) /* (3) */
{
  /* For robust implementation, add here time-out management */
}
//ADC1->CFGR2 &= (~ADC_CFGR2_CKMODE); /* (4) */
```

A.7.5 Single conversion sequence - software trigger

```
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select CHSEL0, CHSEL9, CHSEL10 andCHSEL17 for VRefInt */
/* (3) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater
       than 17.1us */
/* (4) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
//ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL9
             | ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (2) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (3) */
ADC->CCR |= ADC_CCR_VREFEN; /* (4) */
while (1)
{
  /* Performs the AD conversion */
  ADC1->CR |= ADC_CR_ADSTART; /* Start the ADC conversion */
  for (i=0; i < 4; i++)
  {
    while ((ADC1->ISR & ADC_ISR_EOC) == 0) /* Wait end of conversion */
    {
      /* For robust implementation, add here time-out management */
    }
    ADC_Result[i] = ADC1->DR; /* Store the ADC conversion result */
  }
  ADC1->CFGR1 ^= ADC_CFGR1_SCANDIR; /* Toggle the scan direction */
}
```

A.7.6 Continuous conversion sequence - software trigger

```
/* This code example configures the AD conversion in continuous mode and in
   backward scan. It also enable the interrupts. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and scanning direction */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
       17.1us */
/* (5) Enable interrupts on EOC, EOSEQ and overrun */
/* (6) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
//ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_CONT | ADC_CFGR1_SCANDIR; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL9
             | ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (3) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (4) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (5) */
ADC->CCR |= ADC_CCR_VREFEN; /* (6) */

/* Configure NVIC for ADC */
/* (7) Enable Interrupt on ADC */
/* (8) Set priority for ADC */
NVIC_EnableIRQ(ADC1_IRQn); /* (7) */
NVIC_SetPriority(ADC1_IRQn,0); /* (8) */
```

A.7.7 Single conversion sequence - hardware trigger

```
/* Configure the ADC, the ADC and its clock having previously been
   enabled. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on falling edge and external trigger on
       TIM15_TRGO */
/* (3) Select CHSEL0, 1, 2 and 3 */
//ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_0 | ADC_CFGR1_EXTSEL_2; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
             | ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3) */
```

A.7.8 Continuous conversion sequence - hardware trigger

```
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO (EXTSEL = 100),falling
       edge (EXTEN = 10), the continuous mode (CONT = 1)*/
/* (3) Select CHSEL0/1/2/3 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
//ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_1 | ADC_CFGR1_EXTSEL_2
             | ADC_CFGR1_CONT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
             | ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3)*/
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

/* Configure NVIC for ADC */
/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC */
NVIC_EnableIRQ(ADC1_IRQn); /* (1) */
NVIC_SetPriority(ADC1_IRQn,0); /* (2) */
```

A.7.9 DMA one shot mode sequence

```
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC - DMACFG is kept at 0
       for one shot mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs
       on DMA channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                    | DMA_CCR_TEIE | DMA_CCR_TCIE ; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
```

A.7.10 DMA circular mode sequence

```
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC and circular mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs
       on DMA channel 1 */
/* (6) Configure increment, size, interrupts and circular mode */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN | ADC_CFGR1_DMACFG; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                    | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
```

A.7.11 Wait mode sequence

```
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and the wait mode */
/* (3) Select CHSEL1/2/3 */
ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_CONT | ADC_CFGR1_WAIT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
             | ADC_CHSELR_CHSEL3; /* (3)*/
ADC1->CR |= ADC_CR_ADSTART; /* start the ADC conversions */
```

A.7.12 Auto Off and no wait mode sequence

```
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and rising edge
       and auto off */
/* (3) Select CHSEL1/2/3/4 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_1 | ADC_CFGR1_EXTSEL_2
             | ADC_CFGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
             | ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */
```

A.7.13 Auto Off and wait mode sequence

```
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and falling edge,
       the continuous mode, scanning direction and auto off */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_EXTEN_0 | ADC_CFGR1_EXTSEL_2
             | ADC_CFGR1_SCANDIR | ADC_CFGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
             | ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */
```

A.7.14 Analog watchdog

```
/* (1) Select the continuous mode
       and configure the Analog watchdog to monitor only CH17 */
/* (2) Define analog watchdog range : 16b-MSW is the high limit
       and 16b-LSW is the low limit */
/* (3) Enable interrupt on Analog Watchdog */
ADC1->CFGR1 |= ADC_CFGR1_CONT
             | (17 << 26) | ADC_CFGR1_AWDEN | ADC_CFGR1_AWDSGL; /* (1) */
ADC1->TR = (vrefint_high << 16) + vrefint_low; /* (2)*/
ADC1->IER = ADC_IER_AWDIE; /* (3) */
```

A.7.15 Temperature configuration

```
/* (1) Select CHSEL16 for temperature sensor */
/* (2) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
       17.1us */
/* (3) Wake-up the Temperature sensor (only for VBAT, Temp sensor and
       VRefInt) */
ADC1->CHSELR = ADC_CHSELR_CHSEL16; /* (1) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (2) */
ADC->CCR |= ADC_CCR_TSEN; /* (3) */
```

A.7.16 Temperature computation

```
/* Temperature sensor calibration value address */
#define TEMP30_CAL_ADDR ((uint16_t*) ((uint32_t) 0x1FFFF7B8))
#define VDD_CALIB ((uint32_t) (3300))
#define VDD_APPLI ((uint32_t) (3000))
#define AVG_SLOPE ((uint32_t) (5336)) //AVG_SLOPE in ADC conversion step
(@3.3V)/°C multiplied by 1000 for precision on the division
int32_t temperature; /* will contain the temperature in degrees Celsius */
temperature = ((uint32_t) ADC1->DR * VDD_APPLI / VDD_CALIB)) * 1000
            - ((uint32_t) *TEMP30_CAL_ADDR;
temperature = (temperature / AVG_SLOPE) + 30;
```

A.8 Timers

A.8.1 Upcounter on TI2 rising edge

```
/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 9 */
/* (4) Select TIM1_CH2 on PA9 by enabling AF2 for pin 9 in GPIOA AFRH
       register */
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER9))
             | (GPIO_MODER_MODER9_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((9-8)*4); /* (4) */
/* (1) Configure channel 2 to detect rising edges on the TI2 input by
       writing CC2S = ‘01’, and configure the input filter duration by
       writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter
       is needed, keep IC2F=0000).*/
/* (2) Select rising edge polarity by writing CC2P=0 in the TIMx_CCER
       register (reset value). */
/* (3) Configure the timer in external clock mode 1 by writing SMS=111
       Select TI2 as the trigger input source by writing TS=110
       in the TIMx_SMCR register.*/
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_IC2F_0 | TIM_CCMR1_IC2F_1
             | TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC2P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
```

A.8.2 Up counter on each 2 ETR rising edges

```
/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 12 */
/* (4) Select TIM1_ETR on PA12 by enabling AF2 for pin 12 in GPIOA AFRH
       register */
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER12))
             | (GPIO_MODER_MODER12_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((12-8)*4); /* (4) */
/* (1) As no filter is needed in this example, write ETF[3:0]=0000
       in the TIMx_SMCR register. Keep the reset value.
       Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR
       register.
       Select rising edge detection on the ETR pin by writing ETP=0
       in the TIMx_SMCR register. Keep the reset value.
       Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR
       register. */
/* (2) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->SMCR |= TIM_SMCR_ETPS_0 | TIM_SMCR_ECE; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */
```

A.8.3 Input capture configuration

```
/* (1) Select the active input TI1 (CC1S = 01),
       program the input filter for 8 clock cycles (IC1F = 0011),
       select the rising edge on CC1 (CC1P = 0, reset value)
       and prescaler at each valid transition (IC1PS = 00, reset value) */
/* (2) Enable capture by setting CC1E */
/* (3) Enable interrupt on Capture/Compare */
/* (4) Enable counter */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0
             | TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1; /* (1)*/
TIMx->CCER |= TIM_CCER_CC1E; /* (2) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
```

A.8.4 Input capture data management

```
This code must be inserted in the Timer interrupt subroutine.
if ((TIMx->SR & TIM_SR_CC1IF) != 0)
{
  if ((TIMx->SR & TIM_SR_CC1OF) != 0) /* Check the overflow */
  {
    /* Overflow error management */
    gap = 0; /* Reinitialize the laps computing */
    TIMx->SR &= ~(TIM_SR_CC1OF | TIM_SR_CC1IF); /* Clear the flags */
    return;
  }
  if (gap == 0) /* Test if it is the first rising edge */
  {
    counter0 = TIMx->CCR1; /* Read the capture counter which clears the
                              CC1ICF */
   gap = 1; /* Indicate that the first rising edge has yet been detected */
  }
  else
  {
    counter1 = TIMx->CCR1; /* Read the capture counter which clears the
                              CC1ICF */
    if (counter1 > counter0) /* Check capture counter overflow */
    {
      Counter = counter1 - counter0;
    }
    else
    {
      Counter = counter1 + 0xFFFF - counter0 + 1;
    }
    counter0 = counter1;
  }
}
else
{
  /* Unexpected Interrupt */
  /* Manage an error for robust application */
}
```

Note: This code manages only a single counter overflow. To manage many counter overflows the update interrupt must be enabled (UIE = 1) and properly managed.

A.8.5 PWM input configuration

```
/* (1) Select the active input TI1 for TIMx_CCR1 (CC1S = 01),
       select the active input TI1 for TIMx_CCR2 (CC2S = 10) */
/* (2) Select TI1FP1 as valid trigger input (TS = 101)
       configure the slave mode in reset mode (SMS = 100) */
/* (3) Enable capture by setting CC1E and CC2E
      select the rising edge on CC1 and CC1N (CC1P = 0 and CC1NP = 0, reset
       value),
       select the falling edge on CC2 (CC2P = 1). */
/* (4) Enable interrupt on Capture/Compare 1 */
/* (5) Enable counter */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_1; /* (1)*/
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_TS_0
            | TIM_SMCR_SMS_2; /* (2) */
TIMx->CCER |= TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC2P; /* (3) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */
```

A.8.6 PWM input with DMA configuration

```
/* (1) Enable the peripheral clock on DMA */
/* (2) Configure the peripheral data register address for DMA channel x */
/* (3) Configure the memory address for DMA channel x */
/* (4) Configure the number of DMA transfers to be performed
       on DMA channel x */
/* (5) Configure no increment (reset value), size (16-bits), interrupts,
       transfer from peripheral to memory and circular mode
       for DMA channel x */
/* (6) Enable DMA Channel x */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel2->CPAR = (uint32_t) (&(TIM1->CCR1)); /* (2) */
DMA1_Channel2->CMAR = (uint32_t)(&Period); /* (3) */
DMA1_Channel2->CNDTR = 1; /* (4) */
DMA1_Channel2->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
| DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (6) */
/* repeat (2) to (6) for channel 3 */
DMA1_Channel3->CPAR = (uint32_t) (&(TIM1->CCR2)); /* (2) */
DMA1_Channel3->CMAR = (uint32_t)(&DutyCycle); /* (3) */
DMA1_Channel3->CNDTR = 1; /* (4) */
DMA1_Channel3->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                    | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3->CCR |= DMA_CCR_EN; /* (6) */

/* Configure NVIC for DMA */
/* (7) Enable Interrupt on DMA Channels x */
/* (8) Set priority for DMA Channels x */
NVIC_EnableIRQ(DMA1_Channel2_3_IRQn); /* (7) */
NVIC_SetPriority(DMA1_Channel2_3_IRQn,3); /* (8) */
```

A.8.7 Output compare configuration

```
/* (1) Set prescaler to 3, so APBCLK/4 i.e 12MHz */
/* (2) Set ARR = 12000 -1 */
/* (3) Set CCRx = ARR, as timer clock is 12MHz, an event occurs each 1 ms */
/* (4) Select toggle mode on OC1 (OC1M = 011),
       disable preload register on OC1 (OC1PE = 0, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (6) Enable output (MOE = 1)*/
/* (7) Enable counter */
TIMx->PSC |= 3; /* (1) */
TIMx->ARR = 12000 - 1; /* (2) */
TIMx->CCR1 = 12000 - 1; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5)*/
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
```

A.8.8 Edge-aligned PWM configuration example

```
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, , the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
       enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (6) Enable output (MOE = 1)*/
/* (7) Enable counter (CEN = 1)
       select edge aligned mode (CMS = 00, reset value)
       select direction as upcounter (DIR = 0, reset value) */
/* (8) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
             | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */
```

A.8.9 Center-aligned PWM configuration example

```
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz and center-aligned counting,
       the period is 16 us */
/* (3) Set CCRx = 7, the signal will be high during 14 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
       enable preload register on OC1 (OC1PE = 1, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (6) Enable output (MOE = 1)*/
/* (7) Enable counter (CEN = 1)
       select center-aligned mode 1 (CMS = 01) */
/* (8) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 7; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
             | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CMS_0 | TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */
```

A.8.10 ETR configuration to clear OCxREF

```
/* This code is similar to the edge-aligned PWM configuration but it enables
   the clearing on OC1 for ETRclearing (OC1CE = 1) in CCMR1 (5) and ETR is
   configured in SMCR (7).*/
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, , the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
       enable preload register on OC1 (OC1PE = 1),
       enable clearing on OC1 for ETR clearing (OC1CE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Select ETR as OCREF clear source (OCCS = 1),
       select External Trigger Prescaler off (ETPS = 00, reset value),
       disable external clock mode 2 (ECE = 0, reset value),
       select active at high level (ETP = 0, reset value) */
/* (8) Enable counter (CEN = 1),
       select edge aligned mode (CMS = 00, reset value),
       select direction as upcounter (DIR = 0, reset value) */
/* (9) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1PE
             | TIM_CCMR1_OC1CE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->SMCR |= TIM_SMCR_OCCS; /* (7) */
TIMx->CR1 |= TIM_CR1_CEN; /* (8) */
TIMx->EGR |= TIM_EGR_UG; /* (9) */
```

A.8.11 Encoder interface

```
/* (1) Configure TI1FP1 on TI1 (CC1S = 01),
       configure TI1FP2 on TI2 (CC2S = 01) */
/* (2) Configure TI1FP1 and TI1FP2 non inverted (CC1P = CC2P = 0, reset
       value) */
/* (3) Configure both inputs are active on both rising and falling edges
       (SMS = 011) */
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_0; /* (1)*/
TIMx->CCER &= (uint16_t)(~(TIM_CCER_CC21 | TIM_CCER_CC2P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_0 | TIM_SMCR_SMS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
```

A.8.12 Reset mode

```
/* (1) Configure channel 1 to detect rising edges on the TI1 input
       by writing CC1S = '01',
       and configure the input filter duration by writing the IC1F[3:0]
       bits in the TIMx_CCMR1 register (if no filter is needed, keep
       IC1F=0000).*/
/* (2) Select rising edge polarity by writing CC1P=0 in the TIMx_CCER
       register
       Not necessary as it keeps the reset value. */
/* (3) Configure the timer in reset mode by writing SMS=100
       Select TI1 as the trigger input source by writing TS=101
       in the TIMx_SMCR register.*/
/* (4) Set prescaler to 48000-1 in order to get an increment each 1ms */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1)*/
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC1P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIM1->PSC = 47999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */
```

A.8.13 Gated mode

```
/* (1) Configure channel 1 to detect low level on the TI1 input
       by writing CC1S = '01',
       and configure the input filter duration by writing the IC1F[3:0]
       bits in the TIMx_CCMR1 register (if no filter is needed,
       keep IC1F=0000). */
/* (2) Select polarity by writing CC1P=1 in the TIMx_CCER register */
/* (3) Configure the timer in gated mode by writing SMS=101
       Select TI1 as the trigger input source by writing TS=101
       in the TIMx_SMCR register. */
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1)*/
TIMx->CCER |= TIM_CCER_CC1P; /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0
            | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIMx->PSC = 11999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */
```

A.8.14 Trigger mode

```
/* (1) Configure channel 2 to detect rising edge on the TI2 input
       by writing CC2S = '01',
       and configure the input filter duration by writing the IC1F[3:0]
       bits in the TIMx_CCMR1 register (if no filter is needed,
       keep IC1F=0000). */
/* (2) Select polarity by writing CC2P=0 (reset value) in the TIMx_CCER
       register */
/* (3) Configure the timer in trigger mode by writing SMS=110
       Select TI2 as the trigger input source by writing TS=110
       in the TIMx_SMCR register. */
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1)*/
TIMx->CCER &= ~TIM_CCER_CC2P; /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
            | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
TIM1->PSC = 11999; /* (4) */
```

A.8.15 External clock mode 2 + trigger mode

```
/* (1) Configure no input filter (ETF=0000, reset value)
       configure prescaler disabled (ETPS = 0, reset value)
       select detection on rising edge on ETR (ETP = 0, reset value)
       enable external clock mode 2 (ECE = 1) */
/* (2) Configure no input filter (IC1F=0000, reset value)
       select input capture source on TI1 (CC1S = 01) */
/* (3) Select polarity by writing CC1P=0 (reset value) in the TIMx_CCER
       register */
/* (4) Configure the timer in trigger mode by writing SMS=110
       Select TI1 as the trigger input source by writing TS=101
       in the TIMx_SMCR register. */
TIMx->SMCR |= TIM_SMCR_ECE; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (2)*/
TIMx->CCER &= ~TIM_CCER_CC1P; /* (3) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
            | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (4) */
/* Use TI2FP2 as trigger 1 */
/* (1) Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register */
/* (2) TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP=0
       in the TIMx_CCER register (keep the reset value) */
/* (3) Configure TI2FP2 as trigger for the slave mode controller (TRGI)
       by writing TS=110 in the TIMx_SMCR register,
       TI2FP2 is used to start the counter by writing SMS to '110'
       in the TIMx_SMCR register (trigger mode) */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1) */
//TIMx->CCER &= ~(TIM_CCER_CC2P | TIM_CCER_CC2NP); /* (2) */
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_TS_1
            | TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1; /* (3) */
```

A.8.16 One-Pulse mode

```
/* The OPM waveform is defined by writing the compare registers */
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 7, as timer clock is 1MHz the period is 8 us */
/* (3) Set CCRx = 5, the burst will be delayed for 5 us (must be > 0) */
/* (4) Select PWM mode 2 on OC1 (OC1M = 111),
       enable preload register on OC1 (OC1PE = 1, reset value)
       enable fast enable (no delay) if PULSE_WITHOUT_DELAY is set */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Write '1 in the OPM bit in the TIMx_CR1 register to stop the counter
       at the next update event (OPM = 1),
       enable auto-reload register(ARPE = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 7; /* (2) */
TIMx->CCR1 = 5; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_0
             | TIM_CCMR1_OC1PE
#if PULSE_WITHOUT_DELAY > 0
             | TIM_CCMR1_OC1FE
#endif
             ; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_OPM | TIM_CR1_ARPE; /* (7) */
```

A.8.17 Timer prescaling another timer

```
/* TIMy is slave of TIMx */
/* (1) Select Update Event as Trigger output (TRG0) by writing MMS = 010
       in TIMx_CR2. */
/* (2) Configure TIMy in slave mode using ITR1 as internal trigger
       by writing TS = 000 in TIMy_SMCR (reset value)
       Configure TIMy in external clock mode 1, by writing SMS=111 in the
       TIMy_SMCR register. */
/* (3) Set TIMx prescaler to 47999 in order to get an increment each 1ms */
/* (4) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV)
       each second */
/* (5) Set TIMx Autoreload to 24*3600-1 in order to get an overflow each 24-
       hour */
/* (6) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
/* (7) Enable the counter by writing CEN=1 in the TIMy_CR1 register. */
TIMx->CR2 |= TIM_CR2_MMS_1; /* (1) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1 | TIM_SMCR_SMS_0; /* (2) */
TIMx->PSC = 47999; /* (3) */
TIMx->ARR = 999; /* (4) */
TIMy->ARR = (24 * 3600) - 1; /* (5) */
TIMx->CR1 |= TIM_CR1_CEN; /* (6) */
TIMy->CR1 |= TIM_CR1_CEN; /* (7) */
```

A.8.18 Timer enabling another timer

```
/* TIMy is slave of TIMx */
/* (1) Configure Timer x master mode to send its Output Compare 1 Reference
       (OC1REF) signal as trigger output
       (MMS=100 in the TIM1_CR2 register). */
/* (2) Configure the Timer x OC1REF waveform (TIM1_CCMR1 register)
       Channel 1 is in PWM mode 1 when the counter is less than the
       capture/compare register (write OC1M = 110) */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger
       by writing TS = 000 in TIMy_SMCR (reset value)
       Configure TIMy in gated mode, by writing SMS=101 in the
       TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV)
       each 100ms */
/* (7) Set capture compare register to a value between 0 and 999 */
TIMx->CR2 |= TIM_CR2_MMS_2; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 999; /* (6) */
TIMx-> CCR1 = 700; /* (7) */
/* Configure the slave timer to generate toggling on each count */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 1 */
/* (3) Set capture compare register to 1 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 1; /* (2) */
TIMy-> CCR1 = 1; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1
       in the TIMy_CR1 register. */
/* (2) Enable the master counter by writing CEN=1
       in the TIMx_CR1 register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */
```

A.8.19 Master and slave synchronization

```
/* (1) Configure Timer x master mode to send its enable signal
       as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure the Timer x Channel 1 waveform (TIM1_CCMR1 register)
       is in PWM mode 1 (write OC1M = 110) */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger
       by writing TS = 000 in TIMy_SMCR (reset value)
       Configure TIMy in gated mode, by writing SMS=101 in the
       TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV)
       each 10ms */
/* (7) Set capture compare register to a value between 0 and 99 */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 99; /* (6) */
TIMx-> CCR1 = 25; /* (7) */
/* Configure the slave timer Channel 1 as PWM as Timer
   to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 99 */
/* (3) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 99; /* (2) */
TIMy-> CCR1 = 25; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Reset Timer x by writing '1 in UG bit (TIMx_EGR register) */
/* (2) Reset Timer y by writing '1 in UG bit (TIMy_EGR register) */
TIMx->EGR |= TIM_EGR_UG; /* (1) */
TIMy->EGR |= TIM_EGR_UG; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1 in the TIMy_CR1
       register.
       TIMy will start synchronously with the master timer */
/* (2) Start the master counter by writing CEN=1
       in the TIMx_CR1 register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */
```

A.8.20 Two timers synchronized by an external trigger

```
/* (1) Configure TIMx master mode to send its enable signal
       as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure TIMx in slave mode to get the input trigger from TI1
       by writing TS = 100 in TIMx_SMCR
       Configure TIMx in trigger mode, by writing SMS=110 in the
       TIMx_SMCR register.
       Configure TIMx in Master/Slave mode by writing MSM = 1
       in TIMx_SMCR */
/* (3) Configure TIMy in slave mode to get the input trigger from Timer1
       by writing TS = 000 in TIMy_SMCR (reset value)
       Configure TIMy in trigger mode, by writing SMS=110 in the
       TIMy_SMCR register. */
/* (4) Reset Timer x counter by writing '1 in UG bit (TIMx_EGR register) */
/* (5) Reset Timer y counter by writing '1 in UG bit (TIMy_EGR register) */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1)*/
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
            | TIM_SMCR_MSM; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1; /* (3) */
TIMx->EGR |= TIM_EGR_UG; /* (4) */
TIMy->EGR |= TIM_EGR_UG; /* (5) */
/* Configure the Timer Channel 2 as PWM */
/* (1) Configure the Timer x Channel 2 waveform (TIM1_CCMR1 register)
       is in PWM mode 1 (write OC2M = 110) */
/* (2) Set TIMx prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV)
       each 10ms */
/* (4) Set capture compare register to a value between 0 and 99 */
TIMx->CCMR1 |= TIM_CCMR1_OC2M_2 | TIM_CCMR1_OC2M_1; /* (1) */
TIMx->PSC = 2; /* (2) */
TIMx->ARR = 99; /* (3) */
TIMx->CCR2 = 25; /* (4) */
/* Configure the slave timer Channel 1 as PWM as Timer
   to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 */
/* (4) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->PSC = 2; /* (2) */
TIMy->ARR = 99; /* (3) */
TIMy-> CCR1 = 25; /* (4) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (2) Enable output (MOE = 1)*/
TIMx->CCER |= TIM_CCER_CC2E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (2) Enable output (MOE = 1)*/
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
```

A.8.21 DMA burst feature

```
/* In this example TIMx has been previously configured
   in PWM center-aligned */
/* Configure DMA Burst Feature */
/* Configure the corresponding DMA channel */
/* (1) Set DMA channel peripheral address is the DMAR register address */
/* (2) Set DMA channel memory address is the address of the buffer
       in the RAM containing the data to be transferred by DMA
       into CCRx registers */
/* (3) Set the number of data transfer to sizeof(Duty_Cycle_Table) */
/* (4) Configure DMA transfer in CCR register,
       enable the circular mode by setting CIRC bit (optional),
       set memory size to 16_bits MSIZE = 01,
       set peripheral size to 32_bits PSIZE = 10,
       enable memory increment mode by setting MINC,
       set data transfer direction read from memory by setting DIR. */
/* (5) Configure TIMx_DCR register with DBL = 3 transfers
       and DBA = (@TIMx->CCR2 - @TIMx->CR1) >> 2 = 0xE */
/* (6) Enable the TIMx update DMA request by setting UDE bit in DIER
       register */
/* (7) Enable TIMx */
/* (8) Enable DMA channel */
DMA1_Channel2->CPAR = (uint32_t)(&(TIMx->DMAR)); /* (1) */
DMA1_Channel2->CMAR = (uint32_t)(Duty_Cycle_Table); /* (2) */
DMA1_Channel2->CNDTR = 10*3; /* (3) */
DMA1_Channel2->CCR |= DMA_CCR_CIRC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_1
                    | DMA_CCR_MINC | DMA_CCR_DIR; /* (4) */
TIMx->DCR = (3 << 8)
          + (((uint32_t)(&TIMx->CCR2))
          - ((uint32_t)(&TIMx->CR1))) >> 2); /* (5) */
TIMx->DIER |= TIM_DIER_UDE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (8) */
```

A.9 IRTIM code examples

A.9.1 TIM16 and TIM17 configuration

```
/* The following configuration is for RC5 standard */
/* TIM16 is used for the enveloppe while TIM17 is used for the carrier */
#define TIM_ENV TIM16
#define TIM_CAR TIM17
/* (1) Enable the peripheral clocks of Timer 16 and 17 and SYSCFG */
/* (2) Enable the peripheral clock of GPIOB */
/* (3) Select alternate function mode on GPIOB pin 9 */
/* (4) Select AF0 on PB9 in AFRH for IR_OUT (reset value) */
/* (5) Enable the high sink driver capability by setting I2C_PB9_FM+ bit
       in SYSCFG_CFGR1 */
RCC->APB2ENR |= RCC_APB2ENR_TIM16EN | RCC_APB2ENR_TIM17EN
              | RCC_APB2ENR_SYSCFGCOMPEN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOBEN; /* (2) */
GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER9)
             | GPIO_MODER_MODER9_1; /* (3) */
GPIOB->AFR[1] &= ~(0x0F << ((9 - 8) * 4)); /* (4) */
SYSCFG->CFGR1 |= SYSCFG_CFGR1_I2C_FMP_PB9; /* (5) */
/* Configure TIM_CAR as carrier signal */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 1333, as timer clock is 48MHz the frequency is 36kHz */
/* (3) Set CCRx = 1333/4, , the signal will bhave a 25% duty cycle */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
       enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1)*/
/* (6) Enable output (MOE = 1)*/
TIM_CAR->PSC = v; /* (1) */
TIM_CAR->ARR = 1333; /* (2) */
TIM_CAR->CCR1 = (uint16_t)(1333 / 4); /* (3) */
TIM_CAR->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
                | TIM_CCMR1_OC1PE; /* (4) */
TIM_CAR->CCER |= TIM_CCER_CC1E; /* (5) */
TIM_CAR->BDTR |= TIM_BDTR_MOE; /* (6) */
/* Configure TIM_ENV is the modulation enveloppe */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 42627, as timer clock is 48MHz the period is 888 us */
/* (3) Select Forced inactive on OC1 (OC1M = 100) */
/* (4) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (5) Enable output (MOE = 1) */
/* (6) Enable Update interrupt (UIE = 1) */
TIM_ENV->PSC = 0; /* (1) */
TIM_ENV->ARR = 42627; /* (2) */
TIM_ENV->CCMR1 |= TIM_CCMR1_OC1M_2; /* (3) */
TIM_ENV->CCER |= TIM_CCER_CC1E; /* (4) */
TIM_ENV->BDTR |= TIM_BDTR_MOE; /* (5) */
TIM_ENV->DIER |= TIM_DIER_UIE; /* (6) */
/* Enable and reset TIM_CAR only */
/* (1) Enable counter (CEN = 1),
       select edge aligned mode (CMS = 00, reset value),
       select direction as upcounter (DIR = 0, reset value) */
```

```
/* (2) Force update generation (UG = 1) */
TIM_CAR->CR1 |= TIM_CR1_CEN; /* (1) */
TIM_CAR->EGR |= TIM_EGR_UG; /* (2) */
/* Configure TIM_ENV interrupt */
/* (1) Enable Interrupt on TIM_ENV */
/* (2) Set priority for TIM_ENV */
NVIC_EnableIRQ(TIM_ENV_IRQn); /* (1) */
NVIC_SetPriority(TIM_ENV_IRQn,0); /* (2) */
```

A.9.2 IRQHandler for IRTIM

```
/**
  * Description: This function handles TIM_16 interrupt request.
  *              This interrupt subroutine computes the laps between 2
  *              rising edges on T1IC.
  *              This laps is stored in the "Counter" variable.
  */
void TIM16_IRQHandler(void)
{
  uint8_t bit_msg = 0;

  if ((SendOperationReady == 1)
      && (BitsSentCounter < (RC5_GlobalFrameLength * 2)))
  {
    if (BitsSentCounter < 32)
    {
      SendOperationCompleted = 0x00;
      bit_msg = (uint8_t)((ManchesterCodedMsg >> BitsSentCounter)& 1);

      if (bit_msg== 1)
      {
        /* Force active level - OC1REF is forced high */
        TIM_ENV->CCMR1 |= TIM_CCMR1_OC1M_0;
      }
      else
      {
        /* Force inactive level - OC1REF is forced low */
        TIM_ENV->CCMR1 &= (uint16_t)(~TIM_CCMR1_OC1M_0);
      }
    }
    BitsSentCounter++;
  }
  else
  {
    SendOperationCompleted = 0x01;
    SendOperationReady = 0;
    BitsSentCounter = 0;
  }
  /* Clear TIM_ENV update interrupt */
  TIM_ENV->SR &= (uint16_t)(~TIM_SR_UIF);
}
```

A.10 DBG code examples

A.10.1 DBG read device ID

```
/* Read MCU Id, 32-bit access */
MCU_Id = DBGMCU->IDCODE;
```

A.10.2 DBG debug in Low-power mode

```
/* To be able to debug in stop mode */
DBGMCU->CR |= DBGMCU_CR_DBG_STOP;
```

A.11 I2C code examples

A.11.1 I2C configured in master mode to receive

```
/* (1) Timing register value is computed with the AN4235 xls file,
       fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
       fall time = 40ns */
/* (2) Periph enable, receive interrupt enable */
/* (3) Slave address = 0x5A, read transfer, 1 byte to receive, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_RXIE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1<<16) | I2C_CR2_RD_WRN
          | (I2C1_OWN_ADDRESS << 1); /* (3) */
```

A.11.2 I2C configured in master mode to transmit

```
/* (1) Timing register value is computed with the AN4235 xls file,
       fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
       fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 1 byte to transmit, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1 << 16) | (I2C1_OWN_ADDRESS << 1); /* (3) */
```

A.11.3 I2C configured in slave mode

```
/* (1) Timing register value is computed with the AN4235 xls file,
       fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
       fall time = 40ns */
/* (2) Periph enable, address match interrupt enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */
```

A.11.4 I2C master transmitter

```
/* Check Tx empty */
if ((I2C2->ISR & I2C_ISR_TXE) == I2C_ISR_TXE)
{
  I2C2->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
  I2C2->CR2 |= I2C_CR2_START; /* Go */
}
```

A.11.5 I2C master receiver

```
if ((I2C2->ISR & I2C_ISR_RXNE) == I2C_ISR_RXNE)
{
  /* Read receive register, will clear RXNE flag */
  if (I2C2->RXDR == I2C_BYTE_TO_SEND)
  {
    /* Process */
  }
}
```

A.11.6 I2C slave transmitter

```
uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
/* Check address match */
if ((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)
{
  I2C1->ICR |= I2C_ICR_ADDRCF; /* Clear address match flag */
  /* Check if transfer direction is read (slave transmitter) */
  if ((I2C1->ISR & I2C_ISR_DIR) == I2C_ISR_DIR)
  {
    I2C1->CR1 |= I2C_CR1_TXIE; /* Set transmit IT */
  }
}
else if ((I2C_InterruptStatus & I2C_ISR_TXIS) == I2C_ISR_TXIS)
{
  I2C1->CR1 &=~ I2C_CR1_TXIE; /* Disable transmit IT */
  I2C1->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
}
```

A.11.7 I2C slave receiver

```
uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
if ((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)
{
  I2C1->ICR |= I2C_ICR_ADDRCF; /* Address match event */
}
else if ((I2C_InterruptStatus & I2C_ISR_RXNE) == I2C_ISR_RXNE)
{
  /* Read receive register, will clear RXNE flag */
  if (I2C1->RXDR == I2C_BYTE_TO_SEND)
  {
    /* Process */
  }
}
```

A.11.8 I2C configured in master mode to transmit with DMA

```
/* (1) Timing register value is computed with the AN4235 xls file,
       fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
       fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 2 bytes to transmit,
       autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_TXDMAEN; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (SIZE_OF_DATA << 16)
          | (I2C1_OWN_ADDRESS << 1); /* (3) */
```

A.11.9 I2C configured in slave mode to receive with DMA

```
/* (1) Timing register value is computed with the AN4235 xls file,
       fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
       fall time = 40ns */
/* (2) Periph enable, receive DMA enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_RXDMAEN | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */
```

A.12 IWDG code examples

A.12.1 IWDG configuration

```
/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Refresh counter */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
  /* add time out here for a robust application */
}
IWDG->KR = IWDG_REFRESH; /* (6) */
```

A.12.2 IWDG configuration with window

```
/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Set a 50ms window, this will refresh the IWDG */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
  /* add time out here for a robust application */
}
IWDG->WINR = IWDG_RELOAD >> 1; /* (6) */
```

A.13 RTC code examples

A.13.1 RTC calendar configuration

```
/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) set prescaler, 40kHz/128 => 312 Hz, 312Hz/312 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Disable write access for RTC registers */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
  /* add time out here for a robust application */
}
RTC->PRER = 0x007F0137; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &=~ RTC_ISR_INIT; /* (6) */
RTC->WPR = 0xFE; /* (7) */
RTC->WPR = 0x64; /* (7) */
```

A.13.2 RTC alarm configuration

```
/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &=~ RTC_CR_ALRAE; /* (2) */
while ((RTC->ISR & RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */
{
  /* add time out here for a robust application */
}
RTC->ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
            | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
RTC->CR = RTC_CR_ALRAIE | RTC_CR_ALRAE; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */
```

A.13.3 RTC WUT configuration

```
/* (1) Write access for RTC registers */
/* (2) Disable wake up timerto modify it */
/* (3) Wait until it is allow to modify wake up reload value */
/* (4) Modify wake upvalue reload counter to have a wake up each 1Hz */
/* (5) Enable wake up counter and wake up interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &= ~RTC_CR_WUTE; /* (2) */
while ((RTC->ISR & RTC_ISR_WUTWF) != RTC_ISR_WUTWF) /* (3) */
{
  /* add time out here for a robust application */
}
RTC->WUTR = 0x9C0; /* (4) */
RTC->CR = RTC_CR_WUTE | RTC_CR_WUTIE; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */
```

A.13.4 RTC read calendar

```
if((RTC->ISR & RTC_ISR_RSF) == RTC_ISR_RSF)
{
  TimeToCompute = RTC->TR; /* get time */
  DateToCompute = RTC->DR; /* need to read date also */
}
```

A.13.5 RTC calibration

```
/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) set prescaler, 40kHz/125 => 320 Hz, 320Hz/320 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Wait until it's allow to modify calibartion register */
/* (8) Set calibration to around +20ppm, which is a standard value @25°C */
/* Note: the calibration is relevant when LSE is selected for RTC clock */
/* (9) Disable write access for RTC registers */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
  /* add time out here for a robust application */
}
RTC->PRER = (124<<16) | 319; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &=~ RTC_ISR_INIT; /* (6) */
while((RTC->ISR & RTC_ISR_RECALPF) == RTC_ISR_RECALPF) /* (7) */
{
  /* add time out here for a robust application */
}
RTC->CALR = RTC_CALR_CALP | 482; /* (8) */
RTC->WPR = 0xFE; /* (9) */
RTC->WPR = 0x64; /* (9) */
```

A.13.6 RTC tamper and time stamp configuration

```
/* Tamper configuration:
   - Disable precharge (PU)
   - RTCCLK/256 tamper sampling frequency
   - Activate time stamp on tamper detection
   - input rising edge trigger detection on RTC_TAMP2 (PA0)
   - Tamper interrupt enable */
RTC->TAFCR = RTC_TAFCR_TAMPPUDIS | RTC_TAFCR_TAMPFREQ | RTC_TAFCR_TAMPTS
           | RTC_TAFCR_TAMP2E | RTC_TAFCR_TAMPIE;
```

A.13.7 RTC tamper and time stamp

```
/* Check tamper and timestamp flag */
if (((RTC->ISR & (RTC_ISR_TAMP2F)) == (RTC_ISR_TAMP2F))
    && ((RTC->ISR & (RTC_ISR_TSF)) == (RTC_ISR_TSF)))
{
  RTC->ISR &= ~RTC_ISR_TAMP2F; /* clear tamper flag */
  EXTI->PR = EXTI_PR_PR19; /* clear exti line 19 flag */
  TimeToCompute = RTC->TSTR; /* get tamper time in timestamp register */
  RTC->ISR &= ~RTC_ISR_TSF; /* clear timestamp flag */
}
```

A.13.8 RTC clock output

```
/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt,
       enable calibration output (1Hz) */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &=~ RTC_CR_ALRAE; /* (2) */
while ((RTC->ISR & RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */
{
  /* add time out here for a robust application */
}
RTC->ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
            | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
RTC->CR = RTC_CR_ALRAIE | RTC_CR_ALRAE | RTC_CR_COE
        | RTC_CR_COSEL; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */
```

A.14 SPI code examples

A.14.1 SPI master configuration

```
/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
       set to the minimum) CPOL and CPHA at zero (rising first edge) */
/* (2) Slave select output enabled, RXNE IT, 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1->CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1->CR2 = SPI_CR2_SSOE | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
          | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1->CR1 |= SPI_CR1_SPE; /* (3) */
```

A.14.2 SPI slave configuration

```
/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) RXNE IT, 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_RXNEIE | SPI_CR2_FRXTH
          | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */
```

A.14.3 SPI full duplex communication

```
if ((SPI1->SR & SPI_SR_TXE) == SPI_SR_TXE) /* Test Tx empty */
{
  /* Will inititiate 8-bit transmission if TXE */
  *(__IO uint8_t *)&(SPI1->DR) = SPI1_DATA;
}
```

A.14.4 SPI interrupt

```
if ((SPI1->SR & SPI_SR_RXNE) == SPI_SR_RXNE)
{
  SPI1_Data = (uint8_t)SPI1->DR; /* receive data, clear flag */
  /* Process */
}
```

A.14.5 SPI master configuration with DMA

```
/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
       set to the minimum)
       CPOL and CPHA at zero (rising first edge) */
/* (2) TX and RX with DMA,
       enable slave select output,
       enable RXNE interrupt,
       select 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1->CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN | SPI_CR2_SSOE
          | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
          | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1->CR1 |= SPI_CR1_SPE; /* (3) */
```

A.14.6 SPI slave configuration with DMA

```
/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) Select TX and RX with DMA,
       enable RXNE interrupt,
       select 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN
          | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
          | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */
```

A.15 USART code examples

A.15.1 USART transmitter configuration

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_TE | USART_CR1_UE; /* (2) */
```

A.15.2 USART transmit byte

```
/* Start USART transmission */
USART1->TDR = stringtosend[send++]; /* Will inititiate TC if TXE is set*/
```

A.15.3 USART transfer complete

```
if ((USART1->ISR & USART_ISR_TC) == USART_ISR_TC)
{
  if (send == sizeof(stringtosend))
  {
    send=0;
    USART1->ICR |= USART_ICR_TCCF; /* Clear transfer complete flag */
  }
  else
  {
    /* clear transfer complete flag and fill TDR with a new char */
    USART1->TDR = stringtosend[send++];
  }
}
```

A.15.4 USART receiver configuration

```
/* (1) oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity, reception mode */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE; /* (2) */
```

A.15.5 USART receive byte

```
if ((USART1->ISR & USART_ISR_RXNE) == USART_ISR_RXNE)
{
  chartoreceive = (uint8_t)(USART1->RDR); /* Receive data, clear flag */
}
```

A.15.6 USART synchronous mode

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) Synchronous mode
       CPOL and CPHA = 0 => rising first edge
       Last bit clock pulse
       Most significant bit first in transmit/receive */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity
       Transmission enabled, reception enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR2 = USART_CR2_MSBFIRST | USART_CR2_CLKEN
            | USART_CR2_LBCL; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
            | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission w/o clock */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
  /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

A.15.7 USART DMA

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) Enable DMA in reception and transmission */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
       transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_DMAT | USART_CR3_DMAR; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
  /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

A.15.8 USART hardware flow control

```
/* (1) oversampling by 16, 9600 baud */
/* (2) RTS and CTS enabled */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
       transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_RTSE | USART_CR3_CTSE; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
            | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
  /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

A.16 WWDG code examples

A.16.1 WWDG configuration

```
/* (1) Set prescaler to have a roll-over each about 5.5ms,
       set window value (about 2.25ms) */
/* (2) Refresh WWDG before activate it */
/* (3) Activate WWDG */
WWDG->CFR = 0x60; /* (1) */
WWDG->CR = WWDG_REFRESH; /* (2) */
WWDG->CR |= WWDG_CR_WDGA; /* (3) */
```