武汉芯源半导体有限公司
WUHAN XINYUAN SEMICONDUCTOR CO., LTD

DMA function for synchronous triggered sampling of ATIM and ADC

Application note

Rev 1.0

www.whxy.com

Introduction

When working on BLDC motor control, it is necessary to match the sampling moment of the ADC with the PWM waveform generated by the timer in order to obtain accurate sampling values. This article introduces the CW32x030 (x is chip family, x = F, A, the below is same) series chip to implement the function of synchronous triggered sampling of the advanced timer and ADC by using the DMA function.

Contents

1 Configuration of PWM output

1.1 Output port configuration

According to the GPIO multiplexing function assignment table (for a complete table, please refer to Table 9-2 GPIO Multiplexing Function Assignment Table in the CW32x030 User Manual), select the pins that are expected to output complementary PWM waveforms, such as PA08, PA09, PA10, PB13, PB14 and PB15 in this example, as shown in the following table:

Table 1-1 GPIO multiplexing function assignment table

Pin name	Multiplexing function							
	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA08	GPIO		UART1_TXD	BTIM2_TOGN	MCO_OUT	LVD_OUT	GTIM3_ETR	ATIM_CH1A
PA09	GPIO	UART3_TXD	UART1_RXD	I2C1_SCL	BTIM1_TOGP	SPI1_CS	GTIM3_CH1	ATIM_CH2A
PA10	GPIO	UART3_RXD	UART1_CTS	I2C1_SDA	BTIM1_TOGN	SPI1_SCK	GTIM3_CH2	ATIM_CH3A
PB13	GPIO	GTIM2_TOGP	GTIM4_CH3	I2C2_SCL	SPI2_SCK	SPI1_SCK	GTIM1_TOGP	ATIM_CH1B
PB14	GPIO	GTIM2_CH1	GTIM4_CH2	I2C2_SDA	SPI2_MISO	SPI1_MISO	RTC_OUT	ATIM_CH2B
PB15	GPIO	GTIM2_CH2	GTIM4_CH1	BTIM2_TOGP	SPI2_MOSI	SPI1_MOSI	RTC_1Hz	ATIM_CH3B

PA08 and PB13 form a pair of complementary output channels CH1, PA09 and PB14 form a pair of complementary output channels CH2, and PA10 and PB15 form a pair of complementary outputs CH3.

The steps are as follows:

Step 1：Set the relevant GPIOs to output;

Step 2：Configure the GPIOs for ATIM's compare output multiplexing function.

The code is as follows:

```
void PWM_GPIOConfig(void)

{
    /* PA8   PB13   CH1A   CH1B
       PA9   PB14   CH2A   CH2B
       PA10   PB15   CH3A   CH3B
    */
    GPIO_InitTypeDef GPIO_InitStruct;

    __RCC_GPIOA_CLK_ENABLE();          //Enable the GPIO clock in order to configure the GPIO registers
    __RCC_GPIOB_CLK_ENABLE();

    GPIO_InitStruct.IT = GPIO_IT_NONE;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pins = GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOA, &GPIO_InitStruct);
```

```
    GPIO_InitStruct.Pins = GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_Init(CW_GPIOB, &GPIO_InitStruct);

    PA08_AFx_ATIMCH1A();        // GPIO port function multiplexing
    PA09_AFx_ATIMCH2A();
    PA10_AFx_ATIMCH3A();
    PB13_AFx_ATIMCH1B();
    PB14_AFx_ATIMCH2B();
    PB15_AFx_ATIMCH3B();

}
```

1.2 Configuration of the ATIM

The ATIM contains a 16-bit counter/timer and seven comparison units. six of the seven comparison units have a capture function, and the six capture/comparison units can be used in pairs to form a complementary output function.

In this article, the time base signal of the ATIM is selected as PCLK to generate a 20kHz PWM waveform with three complementary outputs required to drive a BLDC motor as an example.

In this example PCLK is 64MHz and is counted at 4MHz after 16 dividing by the ATIM's prescaler.

To facilitate the setting of the ADC sampling time, the ATIM uses the central alignment mode for counting. Setting the ATIM's Auto Reload Register (ARR) to 100, the ATIM's will first accumulate from 0 to 99 and then decrement from 100 to 1, so the counting period is 2 times the value of the ARR register, i.e. the PWM frequency is 20kHz.

By setting the COMP bit of the ATIM's control register (CR) to 1, so that the PWM is output in a complementary fashion, the pulse widths of CH1A and CH1B are determined by channel 1 compare/capture register A (CH1CCRA) and the output pulse width of CH1B is no longer determined by channel 1 compare/capture register B (CH1CCRB). The CH1CCRB can still be used to set the value of the CH1B compare match. CH2A and CH2B, CH3A and CH3B are similar to this.

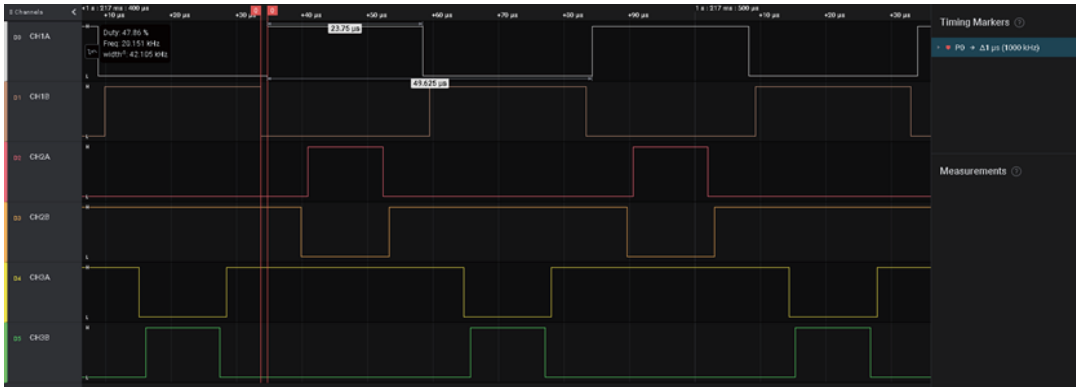
When setting the output PWM complementary outputs, a dead time can be added to the complementary channels, controlled by the Dead Time Register (DTR).

The ATIM is configured to output 3 pairs of complementary PWM waveforms with dead time, with the following detailed configuration code:

```
void ATIM_Config(void)
{
    ATIM_InitTypeDef ATIM_InitStruct;
    ATIM_OCInitTypeDef ATIM_OCInitStruct;

    __RCC_ATIM_CLK_ENABLE();      //Enable the ATIM configuration clock to configure the register
    ATIM_InitStruct.BufferState = DISABLE;      //Disable cache register
    ATIM_InitStruct.ClockSelect = ATIM_CLOCK_PCLK;      //Select PCLK clock count
    ATIM_InitStruct.CounterAlignedMode = ATIM_COUNT_MODE_CENTER_ALIGN;   //Centre alignment
    ATIM_InitStruct.CounterDirection = ATIM_COUNTING_UP;      //Counting upward; not valid for centre alignment
    ATIM_InitStruct.CounterOPMode = ATIM_OP_MODE_REPETITIVE;      //Continuous operate mode
    ATIM_InitStruct.OverFlowMask = ENABLE;      // Repeat counter overflow mask
    ATIM_InitStruct.Prescaler = ATIM_Prescaler_DIV16;      // 16 division, counted clock 4MHz
    ATIM_InitStruct.ReloadValue = 100;      // Add from 0 to ARR-1, then subtract from ARR to 1 (20kHz)
    ATIM_InitStruct.RepetitionCounter = 0;      // Repeat cycle 0
    ATIM_InitStruct.UnderFlowMask = ENABLE;      // Repeat counter underflow mask

    ATIM_Init(&ATIM_InitStruct);      // Configure the base configuration of ATIM

    ATIM_OCInitStruct.BufferState = DISABLE;      // Disable comparison output buffer
```

```
    ATIM_OCInitStruct.OCDMAState = DISABLE;       // No DMA function is triggered when comparing matches
    ATIM_OCInitStruct.OCInterruptSelect = ATIM_OC_IT_NONE;       // No interrupt is generated when comparing matches
    ATIM_OCInitStruct.OCInterruptState = DISABLE;       // Interrupt is disabled when comparing matches
    ATIM_OCInitStruct.OCMode = ATIM_OCMODE_PWM2;       // PWM2 mode for the comparison output
    ATIM_OCInitStruct.OCPolarity = ATIM_OCPOLARITY_NONINVERT;       // Output level is not inverted
    ATIM_OC1AInit(&ATIM_OCInitStruct);       // Configure CH1, CH2, CH3 output comparison
    ATIM_OC2AInit(&ATIM_OCInitStruct);
    ATIM_OC3AInit(&ATIM_OCInitStruct);
    ATIM_OC1BInit(&ATIM_OCInitStruct);
    ATIM_OC2BInit(&ATIM_OCInitStruct);
    ATIM_OC3BInit(&ATIM_OCInitStruct);

    ATIM_SetCompare1A(ATIM_InitStruct.ReloadValue * 50/100);       // Channel 1 outputs PWM with 50% duty cycle
    ATIM_SetCompare2A(ATIM_InitStruct.ReloadValue * 25/100);       // Channel 2 outputs PWM with 75% duty cycle
    ATIM_SetCompare3A(ATIM_InitStruct.ReloadValue * 75/100);       // Channel 3 outputs PWM with 25% duty cycle
    ATIM_PWMOutputConfig(OCREFA_TYPE_SINGLE, OUTPUT_TYPE_COMP, 2);       // Complementary output, insert dead time, current 4MHz clock, 1us dead time, see user manual for dead time calculation
    ATIM_CtrlPWMOutputs(ENABLE);       // Enable PWM output
    ATIM_Cmd(ENABLE);       // Enable ATIM
}
```

The PWM waveforms generated are as follows:



1.3 ADC configuration

In BLDC motor control applications, changes in busbar voltage, busbar current, phase voltage, phase current, Inverted electric potential, etc. need to be captured by an ADC, so the ADC needs to be able to convert multiple analogue quantities. The CW32x030's on-chip ADC has 13 external input channels to meet the demand for the number of samples.

1.3.1 Sequence sampling

When the required sampling channels are less than or equal to 4, this can be achieved through the sequence sampling mode of the ADC, and the sampling moment of the ADC can be set through any of the ATIM's channel 1 to 3 compare/capture registers B. All these operations can be done automatically by the hardware, reducing the workload of the CPU.

Take the 4 inputs from AIN0 to AIN3 as an example, and set the ADC sampling moment to be when the ATIM count reaches ARR, the reference code is as follows:

```
static void ADC_Config(void)
{
    ADC_InitTypeDef ADC_InitStruct;
    ADC_SerialChTypeDef ADC_SerialChStruct;
    ATIM_OCInitTypeDef ATIM_OCInitStruct;

    __RCC_ADC_CLK_ENABLE();

    ADC_InitStruct.ADC_AccEn = ADC_AccDisable;      // ADC accumulation conversion function not on
    ADC_InitStruct.ADC_Align = ADC_AlignRight;      // Sampling results are right-aligned, i.e. the results are stored in bit11~bit0
    ADC_InitStruct.ADC_ClkDiv = ADC_Clk_Div4;       // The sample clock of the ADC is 4 divisions of PCLK, i.e. ADCCLK = 16MHz
    ADC_InitStruct.ADC_DMAEn = ADC_DmaDisable;      // DMA is not triggered when ADC conversion is complete
    ADC_InitStruct.ADC_InBufEn = ADC_BufDisable;    // High speed sampling, ADC internal voltage follower not enabled
    ADC_InitStruct.ADC_OpMode = ADC_SerialChScanMode;     // Sequence scan conversion mode
    ADC_InitStruct.ADC_SampleTime = ADC_SampTime5Clk;     // Set to 5 sampling periods, to be adjusted as appropriate
    ADC_InitStruct.ADC_TsEn = ADC_TsDisable;        // Disable internal temperature sensor
    ADC_InitStruct.ADC_VrefSel = ADC_Vref_VDDA;     // The sampling reference voltage is selected as VDDA

    ADC_SerialChStruct.ADC_InitStruct = ADC_InitStruct;     // Basic configuration items for sequence sampling
    ADC_SerialChStruct.ADC_Sqr0Chmux = ADC_SqrCh0;      // Sequence 0 channel to be converted configured as AIN0
    ADC_SerialChStruct.ADC_Sqr1Chmux = ADC_SqrCh1;      // Sequence 1 channel to be converted configured as AIN1
```

```
    ADC_SerialChStruct.ADC_Sqr2Chmux = ADC_SqrCh2;      // Sequence 2 channel to be converted configured as AIN2
    ADC_SerialChStruct.ADC_Sqr3Chmux = ADC_SqrCh3;      // Sequence 3 channel to be converted configured as AIN3
    ADC_SerialChStruct.ADC_SqrEns = ADC_SqrEns03;       // Enable sequence 0~3

    ADC_SerialChScanModeCfg(&ADC_SerialChStruct);       // ADC sequence scan conversion mode initialisation
    ADC_ExtTrigCfg(ADC_TRIG_ATIM, ENABLE);      // Configure the trigger source for external trigger ADC start as ATIM
    /* Modify part of the ATIM configuration to enable the ADC to start conversion when a comparison match occurs on the CH1B channel of the ATIM */
    ATIM_ADCTriggerConfig(ATIM_ADC_TRIGGER_CH1B, ENABLE);       // Allows CH1B of the ATIM to trigger the ADC to start when comparing matches

    // Modify the CH1CR register of the ATIM to make CH1B trigger the ADC when the down-count comparison matches, note: 0~ARR-1 is the up-count direction
    // ARR~1 is the down-count direction
    REGBITS_MODIFY(CW_ATIM->CH1CR, ATIM_CH1CR_CISB_Msk, ATIM_OC_IT_DOWN_COUNTER << ATIM_CH1CR_CISB_Pos);
    ATIM_SetCompare1B(CW_ATIM->ARR);        // Set the CH1B comparison match value to ARR

    ADC_Enable();
}
```

The above method is implemented entirely in hardware, without CPU and interrupt involvement, and is very efficient in execution. The downside is that the sampling channels are limited to 4.

1.3.2 DMA extended sampling

If more than 4 analog quantities are to be sampled, a combination of DMA functions is required to achieve less CPU involvement. The idea is as follows:

1. The ADC is configured as a single channel single conversion mode, hardware triggered DMA after completion of ADC conversion;
2. Channel CH1 of the DMA is used to transfer the conversion result of the ADC to RAM, in this case 6 ADC channels will be sampled, so the number of transfers CNT is 6, the source address is fixed to the ADC's RESULT0 register and the destination address needs to be incremented;
3. Channel CH2 of the DMA is used to change the sampling channel of the ADC, when the ADC conversion is completed, the channel configuration parameters of the ADC are taken from RAM and the register value of the ADC is automatically configured, so the source address is RAM, the address is incremented and the destination address is the channel control register of the ADC;
4. Channel CH3 of the DMA is used to start the ADC again, because the ADC is configured for single conversion and the ADC automatically stops converting when the conversion is completed. So it needs to be set to the ADC's conversion start register via the DMA to start the ADC conversion again;
5. When the transfer of channel CH1 of the DMA is completed, the 6-way conversion of the ADC is also completed and the conversion result is transferred to RAM. The cyclic sampling of the multi-channel ADC can be achieved by reconfiguring the parameters of the DMA through the interrupt completed by the transfer of CH1;
6. The comparison channel 4 of the ATIM is used to trigger channel CH4 of the DMA, which sets the conversion start register of the ADC and starts the ADC.

Its reference code is as follows:

- ADC configuration:

```
static void ADC_Config(void)
{
    ADC_InitTypeDef ADC_InitStruct;

    __RCC_ADC_CLK_ENABLE();

    ADC_InitStruct.ADC_AccEn = ADC_AccDisable;          // ADC accumulation conversion function not on
    ADC_InitStruct.ADC_Align = ADC_AlignRight;          // Sampling results are right-aligned, i.e. the results are stored in bit11~bit0
    ADC_InitStruct.ADC_ClkDiv = ADC_Clk_Div4;           // The sample clock of the ADC is 4 divisions of PCLK, i.e. ADCCLK = 16MHz
    ADC_InitStruct.ADC_DMAEn = ADC_DmaEnable;           //DMA is triggered when ADC conversion is complete
    ADC_InitStruct.ADC_InBufEn = ADC_BufDisable;        // High speed sampling, ADC internal voltage follower not enabled
    ADC_InitStruct.ADC_OpMode = ADC_SingleChOneMode;    // Single channel single conversion mode
    ADC_InitStruct.ADC_SampleTime = ADC_SampTime5Clk;   // Set to 5 sampling periods, to be adjusted as appropriate
    ADC_InitStruct.ADC_TsEn = ADC_TsDisable;            // Disable internal temperature sensor
```

```
    ADC_InitStruct.ADC_VrefSel = ADC_Vref_VDDA;          // The sampling reference voltage is selected as VDDA

    ADC_Init(&ADC_InitStruct);          // Initialising the ADC configuration

    CW_ADC->CR1_f.CHMUX = 0;          // Initial sampling channel is AIN0
    ADC_Enable();          // Enable ADC
}
```

The ADC is configured for single channel single conversion mode, and DMA can be triggered when the sample is complete.

- DMA configuration:

```
uint16_t ADC_ResultBuff[6] = {0};          // For storing 6-way sampling results
uint8_t ADC_CR1Array[5] = {0x81, 0x82, 0x83, 0x84, 0x85};          // Automatic ADC channel switching by changing the ADC's CR1 register
uint8_t ADC_Start = 0x01;          // Configuration value of ADC_START register

void DMA_Config(void)
{
    // Uses 4 DMA channels: CH1, CH2, CH3, CH4
    // CH1 transfer ADC sample results for single channel single conversion mode into RAM (ADC_ResultBuff[6])
    // CH2 transfer the configuration value of the ADC's CR1 register from RAM (ADC_CR1Array) into the register
    // CH3 transfer the configuration value of the ADC's START register from RAM (ADC_Start) into the register
    // DMA channels CH1, CH2, CH3 triggered by ADC hardware
    // DMA channel CH4 is triggered by ATIM hardware and is used to start the ADC
    DMA_InitTypeDef DMA_InitStruct = {0};

    __RCC_DMA_CLK_ENABLE();

    DMA_InitStruct.DMA_DstAddress = (uint32_t)&ADC_ResultBuff[0];          // Destination address
    DMA_InitStruct.DMA_DstInc = DMA_DstAddress_Increase;          // Incremental destination address
    DMA_InitStruct.DMA_Mode = DMA_MODE_BLOCK;          // BLOCK transmission mode
    DMA_InitStruct.DMA_SrcAddress = (uint32_t)&CW_ADC->RESULT0;          // Source address: ADC's result register
    DMA_InitStruct.DMA_SrcInc = DMA_SrcAddress_Fix;          // Fixed source address
    DMA_InitStruct.DMA_TransferCnt = 0x6;          // Number of DMA transfers
    DMA_InitStruct.DMA_TransferWidth = DMA_TRANSFER_WIDTH_16BIT;          // Data bit width 16bit
    DMA_InitStruct.HardTrigSource = DMA_HardTrig_ADC_TRANSCOMPLETE;          // ADC conversion completion triggers DMA
```

```
DMA_InitStruct.TrigMode = DMA_HardTrig;          // Hardware trigger mode
DMA_Init(CW_DMACHANNEL1, &DMA_InitStruct);
DMA_Cmd(CW_DMACHANNEL1, ENABLE);

DMA_InitStruct.DMA_DstAddress = (uint32_t)&CW_ADC->CR1;          // Destination address
DMA_InitStruct.DMA_DstInc = DMA_DstAddress_Fix;          // Fixed destination address
DMA_InitStruct.DMA_Mode = DMA_MODE_BLOCK;          // BLOCK transmission mode
DMA_InitStruct.DMA_SrcAddress = (uint32_t)&ADC_CR1Array[0];          // Source address
DMA_InitStruct.DMA_SrcInc = DMA_SrcAddress_Increase;          // Incremental source address
DMA_InitStruct.DMA_TransferCnt = 0x5;          // Number of DMA transfers
DMA_InitStruct.DMA_TransferWidth = DMA_TRANSFER_WIDTH_8BIT;          // Data bit width 8bit
DMA_InitStruct.HardTrigSource = DMA_HardTrig_ADC_TRANSCOMPLETE;          // ADC conversion completion triggers DMA
DMA_InitStruct.TrigMode = DMA_HardTrig;          // Hardware trigger mode
DMA_Init(CW_DMACHANNEL2, &DMA_InitStruct);
DMA_Cmd(CW_DMACHANNEL2, ENABLE);
DMA_InitStruct.DMA_DstAddress = (uint32_t)&CW_ADC->START;          // Destination address
DMA_InitStruct.DMA_DstInc = DMA_DstAddress_Fix;          // Fixed destination address
DMA_InitStruct.DMA_Mode = DMA_MODE_BLOCK;          // BLOCK transmission mode
DMA_InitStruct.DMA_SrcAddress = (uint32_t)&ADC_Start;          // Source address
DMA_InitStruct.DMA_SrcInc = DMA_SrcAddress_Fix;          // Fixed source address
DMA_InitStruct.DMA_TransferCnt = 0x5;          // Number of DMA transfers
DMA_InitStruct.DMA_TransferWidth = DMA_TRANSFER_WIDTH_8BIT;          // Data bit width 8bit
DMA_InitStruct.HardTrigSource = DMA_HardTrig_ADC_TRANSCOMPLETE;          // ADC conversion completion triggers DMA
DMA_InitStruct.TrigMode = DMA_HardTrig;          // Hardware trigger mode
DMA_Init(CW_DMACHANNEL3, &DMA_InitStruct);
DMA_Cmd(CW_DMACHANNEL3, ENABLE);

DMA_InitStruct.DMA_DstAddress = (uint32_t)&CW_ADC->START;          // Destination address
DMA_InitStruct.DMA_DstInc = DMA_DstAddress_Fix;          // Fixed destination address
DMA_InitStruct.DMA_Mode = DMA_MODE_BLOCK;          // BLOCK transmission mode
DMA_InitStruct.DMA_SrcAddress = (uint32_t)&ADC_Start;          // Source address
DMA_InitStruct.DMA_SrcInc = DMA_SrcAddress_Fix;          // Fixed source address
DMA_InitStruct.DMA_TransferCnt = 0x1;          // Number of DMA transfers
DMA_InitStruct.DMA_TransferWidth = DMA_TRANSFER_WIDTH_8BIT;          // Data bit width 8bit
DMA_InitStruct.HardTrigSource = DMA_HardTrig_ATIM_CH1A2A3A4;          // ATIM hardware trigger
DMA_InitStruct.TrigMode = DMA_HardTrig;          // Hardware trigger mode
DMA_Init(CW_DMACHANNEL4, &DMA_InitStruct);
DMA_Cmd(CW_DMACHANNEL4, ENABLE);
```

```
    DMA_ITConfig(CW_DMACHANNEL1, DMA_IT_TC, ENABLE);          // Channel CH1 of the DMA triggers
    an interrupt when the transfer is complete
}
```

- ATIM requires the addition of a channel 4 setting to the previous configuration, with the following additional code:

```
{
    // Configure OC4 compare match trigger DMA
    CW_ATIM->MSCR_f.CCDS = 1;          // Allows the DMA master switch of the ATIM to be opened
    // Comparison channel 4 triggers DMA on comparison match when counting down
    ATIM_OC4Init(ENABLE, ATIM_OC_IT_DOWN_COUNTER, ENABLE, DISABLE, DISABLE);
    ATIM_SetCompare4(ATIM_InitStruct.ReloadValue);          // Trigger moment
}
```

- Once the CH1 channel of the DMA has been transferred, an interrupt service routine will be triggered, the code for which is as follows:

```
void DMACH1_IRQHandlerCallBack(void)
{
    CW_DMA->ICR_f.TC1 = 0x00;          // Clear interrupt flag
    CW_ATIM->ICR = 0x00;          // Clear the ATIM interrupt flag
    CW_ADC->ICR = 0x00;          // Clear the ADC interrupt flag
    CW_ADC->CR1 = 0x80;          // Restore the ADC's sampling channel to AIN0
    //-----------------------------------------------
    // DMA.CH1 : Restore to the initial setting, i.e. restore the ADC's sample memory address to the first
    position
    CW_DMACHANNEL1->CNT = 0x10006;          // Transfer 6
    CW_DMACHANNEL1->SRCADDR = (uint32_t)(&CW_ADC->RESULT0);
    CW_DMACHANNEL1->DSTADDR = (uint32_t)(&ADC_ResultBuff[0]);
    CW_DMACHANNEL1->CSR_f.EN = 1;



    //-----------------------------------------------
    // CH2 : Restore the next sampled channel to AIN1, reset the number of transfers to 5
    CW_DMACHANNEL2->CNT = 0x10005;          // Transfer 5
    CW_DMACHANNEL2->SRCADDR = (uint32_t)(&ADC_CR1Array[0]);
    CW_DMACHANNEL2->DSTADDR = (uint32_t)(&CW_ADC->CR1);
    CW_DMACHANNEL2->CSR_f.EN = 1;

    //-----------------------------------------------
    // DMA.CH3 : Reset the number of transfers to 5
```

```
    CW_DMACHANNEL3->CNT = 0x10005;          // Transfer 5
    CW_DMACHANNEL3->SRCADDR = (uint32_t)(&ADC_Start);
    CW_DMACHANNEL3->DSTADDR = (uint32_t)(&CW_ADC->START);
    CW_DMACHANNEL3->CSR_f.EN = 1;
    //-----------------------------------------------
    // DMA.CH4 : Reset the transfer number , wait for ATIM comparison match to trigger DMA transfer
    CW_DMACHANNEL4->CNT = 0x10001;          // Transfer 1
    CW_DMACHANNEL4->SRCADDR = (uint32_t)(&ADC_Start);
    CW_DMACHANNEL4->DSTADDR = (uint32_t)(&CW_ADC->START);
    CW_DMACHANNEL4->CSR_f.EN = 1;
}
```

This method allows more than 4 analogue channels to be sampled, the results are automatically stored in memory and the DMA configuration is reset only once after the last channel has been sampled, so the CPU overhead is relatively low and the timing of the samples can be set flexibly by comparing channel 4 of the ATIM.

2 Revision history

Table 2-1 Document revision history

Date	Revision	Changes
June 12, 2023	Rev 1.0	Initial release.