AN0011
Application Note

AT32 3ADC simultaneous trigger

Introduction

This application note introduces the method of triggering three ADC conversions through the same trigger source on the AT32 microcontrollers to realize synchronous action of three ADC channels at any time, so as to meet the application that requires synchronous conversion of three ADCs.

Note: The corresponding code in this application note is developed on the basis of V2.x.x BSP provided by Artery. For other versions of BSP, please pay attention to the differences in usage.

Applicable products:

Part number	AT32F403 series
	AT32F403A series
	AT32F407 series

Contents

List of Tables

List of Figures

1 ADC structure

The ADC is a peripheral that converts an analog input signal into a 12-bit digital signal. Its sampling rate is as high as 2 MSPS. It has up to 18 channels (16 internal channels and 2 external channels) for sampling and conversion. It has the following functions:

1) Support single, repetition, sequence, automatic preempted group conversion and partition modes;

2) Both ordinary and preempted channels support trigger by software and external trigger, and external trigger has multiple optional trigger sources to choose from;

3) Converted data can be stored with left or right alignment, and preempted channels support data offset setting;

4) Voltage monitoring feature allows applications to monitor input voltage for exceeding the user-defined high/low thresholds.

5) Preempted channels conversion end, channels conversion end and voltage monitoring out of range have their respective interrupt enable bits;

6) Adjustable ADC clock (derived from PCLK2, maximum frequency of ACCLK: 28 MHz);

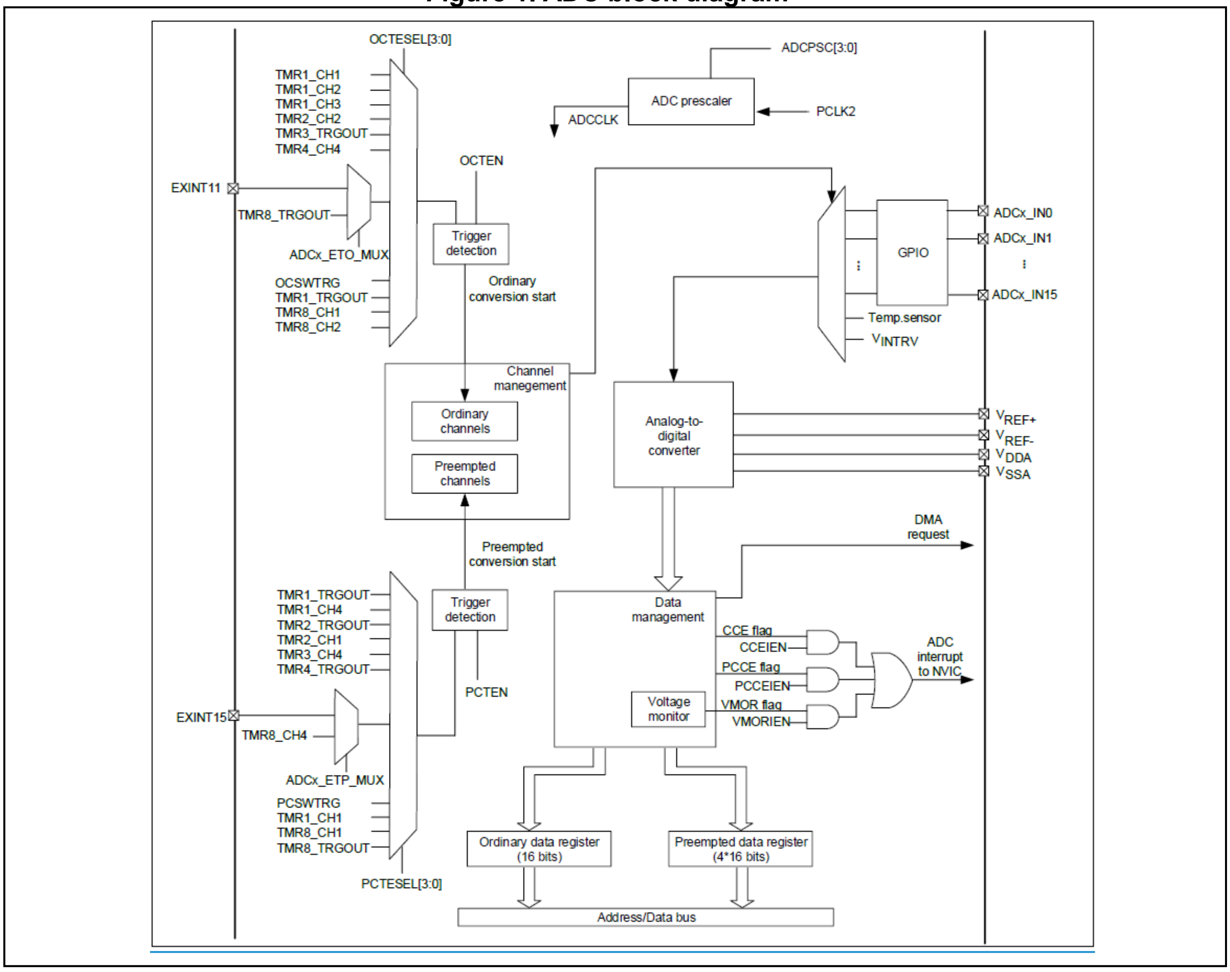
7) Support up to 20 channels sampling conversions (ordinary group: 16 channels; preempted group: 4 channels), and the sampling period is adjustable as required;

8) Ordinary channels conversion data can be transferred through DMA; when multiple channels are selected, DMA must be used to obtain conversion data;

9) Support multiple master/slave modes linking ADC1 and ADC2.

The structure of one ADC is shown below.

Figure 1. ADC block diagram



2 Principle of 3 ADC simultaneous trigger

In the regular simultaneous mode of ADC master/slave mode, ADC2 will fully synchronize ADC1 action, and ADC1 and ADC3 have multiple identical trigger sources. Therefore, the application can combine ADC1 and ADC 2 into master/slave mode (regular simultaneous mode), and ADC3 and ADC1 use the same trigger source so as to perform 3ADC simultaneous actions.

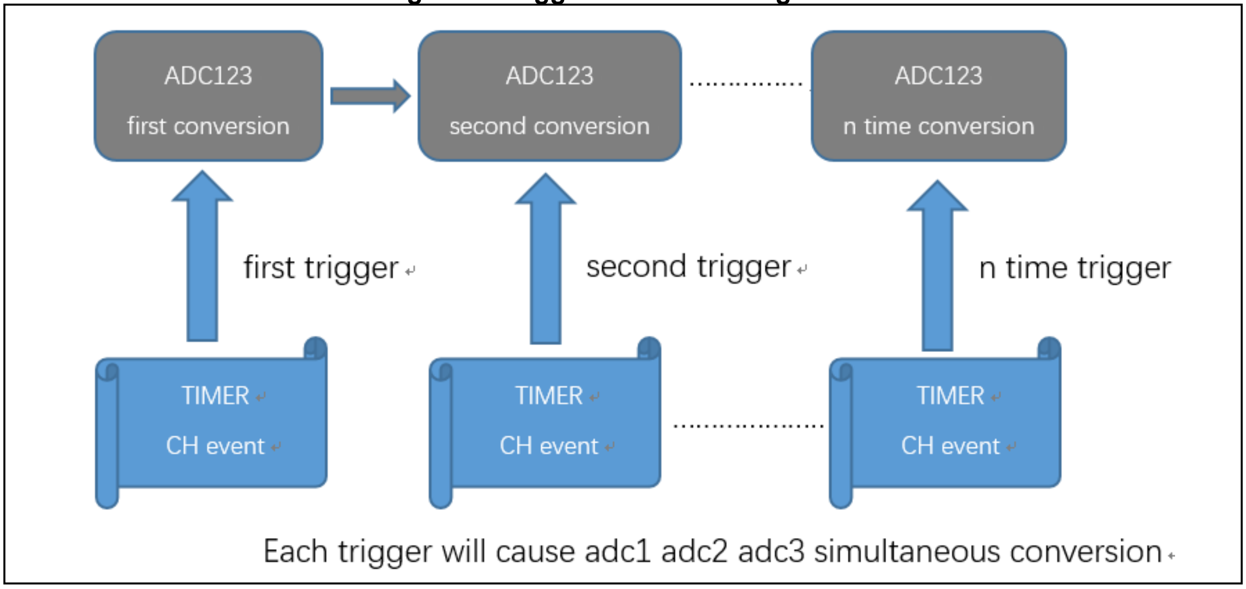
In application, initialize two channels for ADC1, ADC2 and ADC3 respectively. ADC1 uses channels 4~5, ADC2 uses channels 7~8 and ADC3 uses channels 10~11, to realize the following simultaneous trigger and conversion.

Table 1. Trigger situation

Trigger situation						
Trigger	First time		Second time		Third time	
ADC1	Channel 4	Channel 5	Channel 4	Channel 5	Channel 4	Channel 5
ADC2	Channel 7	Channel 8	Channel 7	Channel 8	Channel 7	Channel 8
ADC3	Channel 10	Channel 11	Channel 10	Channel 11	Channel 10	Channel 11

The trigger schematic diagram is shown below:

Figure 2. Trigger schematic diagram



3 ADC configuration

- **Trigger source configuration**

Configuration code of this application case is as follows:

```
static void tmr1_config(void)
{
    gpio_init_type gpio_initstructure;
    tmr_output_config_type tmr_oc_init_structure;
    crm_clocks_freq_type crm_clocks_freq_struct = {0};
    crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);

    gpio_default_para_init(&gpio_initstructure);
    gpio_initstructure.gpio_mode = GPIO_MODE_MUX;
    gpio_initstructure.gpio_pins = GPIO_PINS_8;
    gpio_initstructure.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
    gpio_initstructure.gpio_pull = GPIO_PULL_NONE;
    gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init(GPIOA, &gpio_initstructure);

    /* get system clock */
    crm_clocks_freq_get(&crm_clocks_freq_struct);

    crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);

    /* (systemclock/(systemclock/10000))/10000 = 1Hz(1s) */
    tmr_base_init(TMR1, 9999, (crm_clocks_freq_struct.sclk_freq/10000 - 1));
    tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
    tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV1);

    tmr_output_default_para_init(&tmr_oc_init_structure);
    tmr_oc_init_structure.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
    tmr_oc_init_structure.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;
    tmr_oc_init_structure.oc_output_state = TRUE;
    tmr_oc_init_structure.oc_idle_state = FALSE;
    tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_oc_init_structure);
    tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 5000);
    tmr_channel_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
    tmr_output_enable(TMR1, TRUE);
}
```

The case uses timer ch1 event to trigger ADC. On the basis of regular timer ch event configuration, the following should be noted.

Application design:

1) For the purpose of demonstration, the timer is set to 1 Hz, and the frequency can be adjusted as required on the premise that the trigger interval is not less than ADC sequence conversion time.
2) In order to realize ADC trigger, the tmr_output_enable(TMR1, TRUE) command cannot be omitted.

3) In order to avoid false trigger, the timer can be enabled only after DMA and ADC are configured.
4) In addition to TMR1_CH1, the ADC1 and ADC3 ordinary channel trigger sources also support:
   - TMR1_CH3 event
   - TMR1_TRGOUT event
   - TMR8_CH1 event
   - TMR8_TRGOUT event

DMA configuration

Configuration code of this application case is as follows:

```
static void dma_config(void)
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_DMA2_PERIPH_CLOCK, TRUE);
  nvic_irq_enable(DMA1_Channel1_IRQn, 0, 0);
  nvic_irq_enable(DMA2_Channel4_5_IRQn, 0, 0);
  dma_reset(DMA1_CHANNEL1);
  dma_reset(DMA2_CHANNEL5);
  dma_default_para_init(&dma_init_struct);
  dma_init_struct.buffer_size = 2;
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;
  dma_init_struct.memory_base_addr = (uint32_t)adc1_ordinary_valuetab;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_WORD;
  dma_init_struct.memory_inc_enable = TRUE;
  dma_init_struct.peripheral_base_addr = (uint32_t)&(ADC1->odt);
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_WORD;
  dma_init_struct.peripheral_inc_enable = FALSE;
  dma_init_struct.priority = DMA_PRIORITY_HIGH;
  dma_init_struct.loop_mode_enable = TRUE;
  dma_init(DMA1_CHANNEL1, &dma_init_struct);

  dma_init_struct.memory_base_addr = (uint32_t)adc3_ordinary_valuetab;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
  dma_init_struct.peripheral_base_addr = (uint32_t)&(ADC3->odt);
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
  dma_init(DMA2_CHANNEL5, &dma_init_struct);

  dma_interrupt_enable(DMA1_CHANNEL1, DMA_FDT_INT, TRUE);
  dma_interrupt_enable(DMA2_CHANNEL5, DMA_FDT_INT, TRUE);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);
  dma_channel_enable(DMA2_CHANNEL5, TRUE);
}
```

In this case, the converted data of ADC1&ADC2 is transferred through DMA1_CHANNEL1, and the converted data of ADC3 is transferred through DMA2_CHANNEL5. On the basis of regular DMA configuration, the following should be noted.

1) Perform flexible mapping configuration for channels if the application needs to replace DMA_CHANNEL.
2) The converted data of ADC1&ADC2 will be combined into 32-bit data for transfer; therefore, the peripheral and memory data width of DMA1_CHANNEL1 must be set to 32-bit.
3) The converted data of ADC3 remains 16-bit width; therefore, the peripheral and memory data width of DMA2_CHANNEL5 must be set to 16-bit.
4) The data is transferred from ADC peripheral to memory; therefore, the peripheral is set as the source of DAM data transfer direction.
5) In order to ensure stable transfer of data, the number of data transferred through DMA channels is set according to the number of ADC ordinary channel groups.
6) The priority level of DMA channels is set according to actual application. When the ADC conversion is fast, the priority of DMA channel should be increased appropriately to avoid data loss during transmission.

■ **ADC configuration**

Configuration code of this application case is as follows.

```
static void adc_config(void)
{
    adc_base_config_type adc_base_struct;
    crm_periph_clock_enable(CRM_ADC1_PERIPH_CLOCK, TRUE);
    crm_periph_clock_enable(CRM_ADC2_PERIPH_CLOCK, TRUE);
    crm_periph_clock_enable(CRM_ADC3_PERIPH_CLOCK, TRUE);
    crm_adc_clock_div_set(CRM_ADC_DIV_6);

    /* select combine mode */
    adc_combine_mode_select(ADC_ORDINARY_SMLT_ONLY_MODE);
    adc_base_default_para_init(&adc_base_struct);
    adc_base_struct.sequence_mode = TRUE;
    adc_base_struct.repeat_mode = FALSE;
    adc_base_struct.data_align = ADC_RIGHT_ALIGNMENT;
    adc_base_struct.ordinary_channel_length = 2;
    adc_base_config(ADC1, &adc_base_struct);
    adc_ordinary_channel_set(ADC1, ADC_CHANNEL_4, 1, ADC_SAMPLETIME_239_5);
    adc_ordinary_channel_set(ADC1, ADC_CHANNEL_5, 2, ADC_SAMPLETIME_239_5);
    adc_ordinary_conversion_trigger_set(ADC1, ADC12_ORDINARY_TRIG_TMR1CH1, TRUE);
    adc_dma_mode_enable(ADC1, TRUE);

    adc_base_config(ADC2, &adc_base_struct);
    adc_ordinary_channel_set(ADC2, ADC_CHANNEL_7, 1, ADC_SAMPLETIME_239_5);
    adc_ordinary_channel_set(ADC2, ADC_CHANNEL_8, 2, ADC_SAMPLETIME_239_5);
    adc_ordinary_conversion_trigger_set(ADC2, ADC12_ORDINARY_TRIG_SOFTWARE, TRUE);

    adc_base_config(ADC3, &adc_base_struct);
    adc_ordinary_channel_set(ADC3, ADC_CHANNEL_10, 1, ADC_SAMPLETIME_239_5);
    adc_ordinary_channel_set(ADC3, ADC_CHANNEL_11, 2, ADC_SAMPLETIME_239_5);
    adc_ordinary_conversion_trigger_set(ADC3, ADC3_ORDINARY_TRIG_TMR1CH1, TRUE);
    adc_dma_mode_enable(ADC3, TRUE);
```

```
    adc_enable(ADC1, TRUE);
    adc_enable(ADC2, TRUE);
    adc_calibration_init(ADC1);
    while(adc_calibration_init_status_get(ADC1));
    adc_calibration_start(ADC1);
    while(adc_calibration_status_get(ADC1));
    adc_calibration_init(ADC2);
    while(adc_calibration_init_status_get(ADC2));
    adc_calibration_start(ADC2);
    while(adc_calibration_status_get(ADC2));

    adc_enable(ADC3, TRUE);
    adc_calibration_init(ADC3);
    while(adc_calibration_init_status_get(ADC3));
    adc_calibration_start(ADC3);
    while(adc_calibration_status_get(ADC3));
}
```

In this case, ADC1, ADC2 and ADC3 are used to realize conversion of channels. On the basis of regular ADC configuration, the following should be noted.

1) The converted data of ADC2 does not have an independent DMA transfer; therefore, to realize simultaneous conversion, ADC1 and ADC2 must be combined into the master/slave mode (regular simultaneous mode). ADC3 has an independent DMA request, so it is configured to independent mode.
2) For the purpose of simultaneous conversion, ADC1 and ADC3 should select the same trigger source for ordinary channels. In this case, TMR1_CH1 is used, and one of the following should also be selected:
   - TMR1_CH3 event
   - TMR1_TRGOUT event
   - TMR8_CH1 event
   - TMR8_TRGOUT event
3) Trigger by software must be selected as the trigger source of ADC2 to avoid loss of synchronization of ADC2 in slave mode.
4) The trigger interval cannot be less than the ADC sequence conversion time to ensure that the trigger can be responded effectively.
5) Calibration can be performed only after ADC1 and ADC2 in master/slave mode are enabled.
6) Ensure that the same channel cannot be sampled and converted by multiple ADCs simultaneously.
7) In order to ensure stable transfer of data, the number of data transferred through DMA channels is set according to the number of ADC ordinary channel groups.
8) This case is only applicable to ADC ordinary channels; the converted data of preempted channels does not have DMA transfer capability.

■ **Recommended overall initialization sequence**

The recommended initialization sequence is as follows.

```
gpio_config();
tmr1_config();
dma_config();
adc_config();
```

```
tmr_counter_enable(TMR1, TRUE);
```

This initialization sequence is recommended based on the following considerations:

1) Configure timer before ADC initialization, and enable timer after ADC initialization is enabled

   The timer starts to run after being enabled, and generates a trigger event as soon as it meet requirements. At this point, if ADC initialization is not completed, the trigger event will be lost; if ADC is being calibrated or waiting for power-on, a corresponding trigger event may occur.

2) Configure and enable DAM before ADC initialization is enabled

   After ADC is enabled, it will start conversion accordingly as long as there is a trigger condition, and each channel will generate a DMA transfer request immediately after the completion of conversion. If there is hysteresis in DMA enabling, the data transfer request will not be responded in time, causing data loss and eventually data misalignment.

■ Interrupt service function design

Code of this application case is designed as follows:

```
void DMA1_Channel1_IRQHandler(void)
{
  if(dma_flag_get(DMA1_FDT1_FLAG) != RESET)
  {
    dma_flag_clear(DMA1_FDT1_FLAG);
    dma1_trans_complete_flag = 1;
  }
}
void DMA2_Channel4_5_IRQHandler(void)
{
  if(dma_flag_get(DMA2_FDT5_FLAG) != RESET)
  {
    dma_flag_clear(DMA2_FDT5_FLAG);
    dma2_trans_complete_flag = 1;
  }
}
```

This case only uses two DMA transfer complete interrupts, and the interrupt service function only records whether a transfer complete event occurred.

The interrupt service function is designed as simple and concise as possible. Since the response to interrupt function follows the priority principle, in order to avoid delaying the execution of other important application codes due to complex interrupt response, it is recommended not to heap too much application logic in the interrupt function.

■ main function design

Code of this application case is designed as follows:

```
int main(void)
{
  __IO uint32_t index = 0;
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  system_clock_config();
  at32_board_init();
  at32_led_off(LED2);
  at32_led_off(LED3);
  at32_led_off(LED4);
```

```
    usart1_config(115200);
    gpio_config();
    tmr1_config();
    dma_config();
    adc_config();

    tmr_counter_enable(TMR1, TRUE);
    while(1)
    {
      if(dma1_trans_complete_flag != 0)
      {
        index++;
        dma1_trans_complete_flag = 0;
        printf("adc1_channel4_data[0] = 0x%x\r\n", adc1_ordinary_valuetab[0] & 0xFFFF);
        printf("adc1_channel5_data[1] = 0x%x\r\n", adc1_ordinary_valuetab[1] & 0xFFFF);
        printf("adc2_channel7_data[0] = 0x%x\r\n", (adc1_ordinary_valuetab[0] >> 16) & 0xFFFF);
        printf("adc2_channel8_data[1] = 0x%x\r\n", (adc1_ordinary_valuetab[1] >> 16) & 0xFFFF);
        printf("\r\n");
        at32_led_toggle(LED2);
      }
      if(dma2_trans_complete_flag != 0)
      {
        dma2_trans_complete_flag = 0;
        printf("adc3_channel10_data[0] = 0x%x\r\n", adc3_ordinary_valuetab[0]);
        printf("adc3_channel11_data[1] = 0x%x\r\n", adc3_ordinary_valuetab[1]);
        printf("\r\n");
        at32_led_toggle(LED3);
      }
    }
}
```

In the main function, except for peripheral initialization, only the printout of converted data is performed by querying the DMA transfer complete event flag. The following should be noted when designing the application.

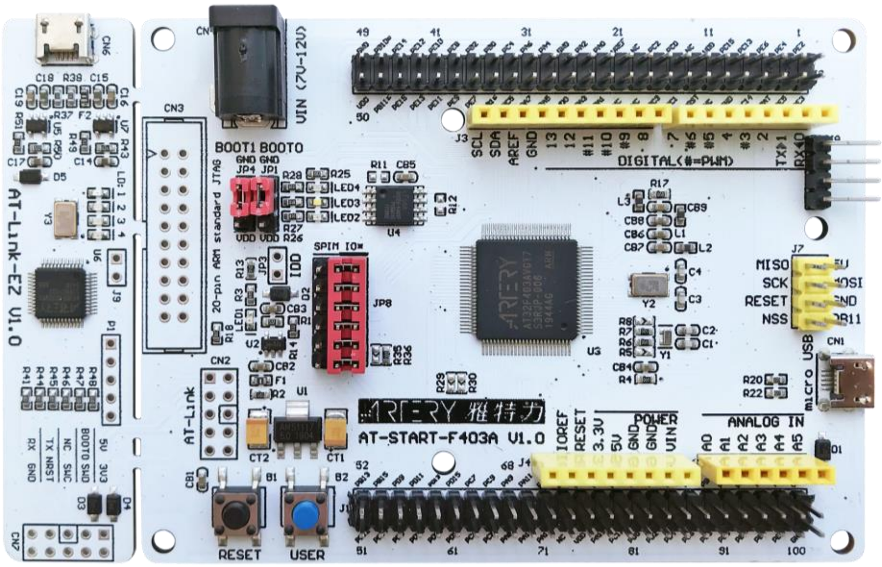
The data of ADC1 and ADC2 in master/slave mode (regular simultaneous mode) is encapsulated into 32-bit data by hardware, and the 32-bit data obtained through DMA can be used only after being parsed. The upper 16 bits are the converted data of ADC2, and the lower 16 bits are the converted data of ADC1.

4 How to use Demo Code

■ **Hardware resources**

AT-START-F403A V1.0 demo board

Figure 3. AT-START-F403A



ADC channels and corresponding GPIO ports are listed below.

Table 2. ADC channels and corresponding GPIOs

ADC1	Channel 4→ PA4	Channel 5→ PA5
ADC2	Channel 7→ PA7	Channel 8→ PB0
ADC3	Channel 10→ PC0	Channel 11→ PC1

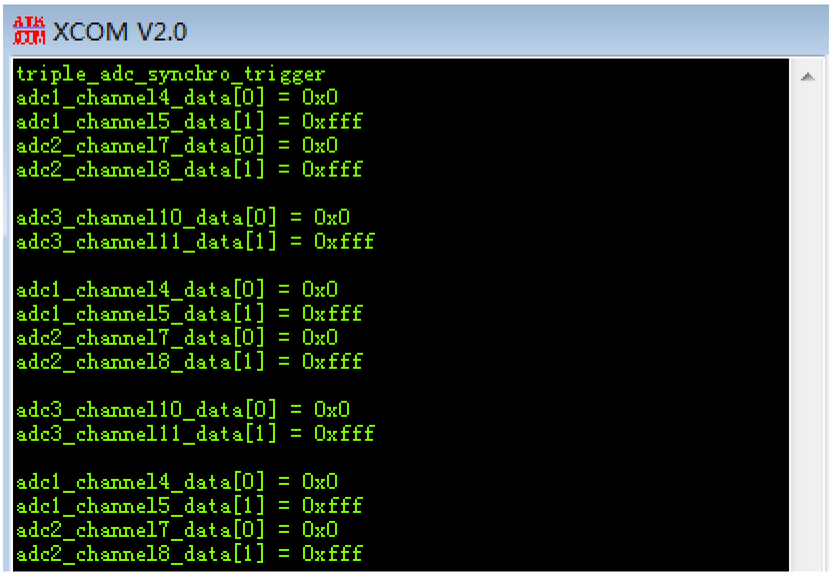
When doing the test, apply the voltage values to these six GPIOs respectively.

■ **Test method**

1. Open the project, compile and download to the target board;
2. Apply voltage values to the corresponding ADC1/2/3 pins. Print through the serial port or enter debug mode to check whether the conversion result is as expected.

The test result is shown below:

Figure 4. Test result



The corresponding channel voltage value has been converted and transferred to the specified array through DMA.

Note: All projects are built around AT32F403A. If users want to use them in other models, please refer to sample projects of each model in AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates for a simple change.

Note: All projects are built around keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

5 Revision history

Table 3. Document revision history

Date	Version	Revision note
2021.12.14	2.0.0	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.