

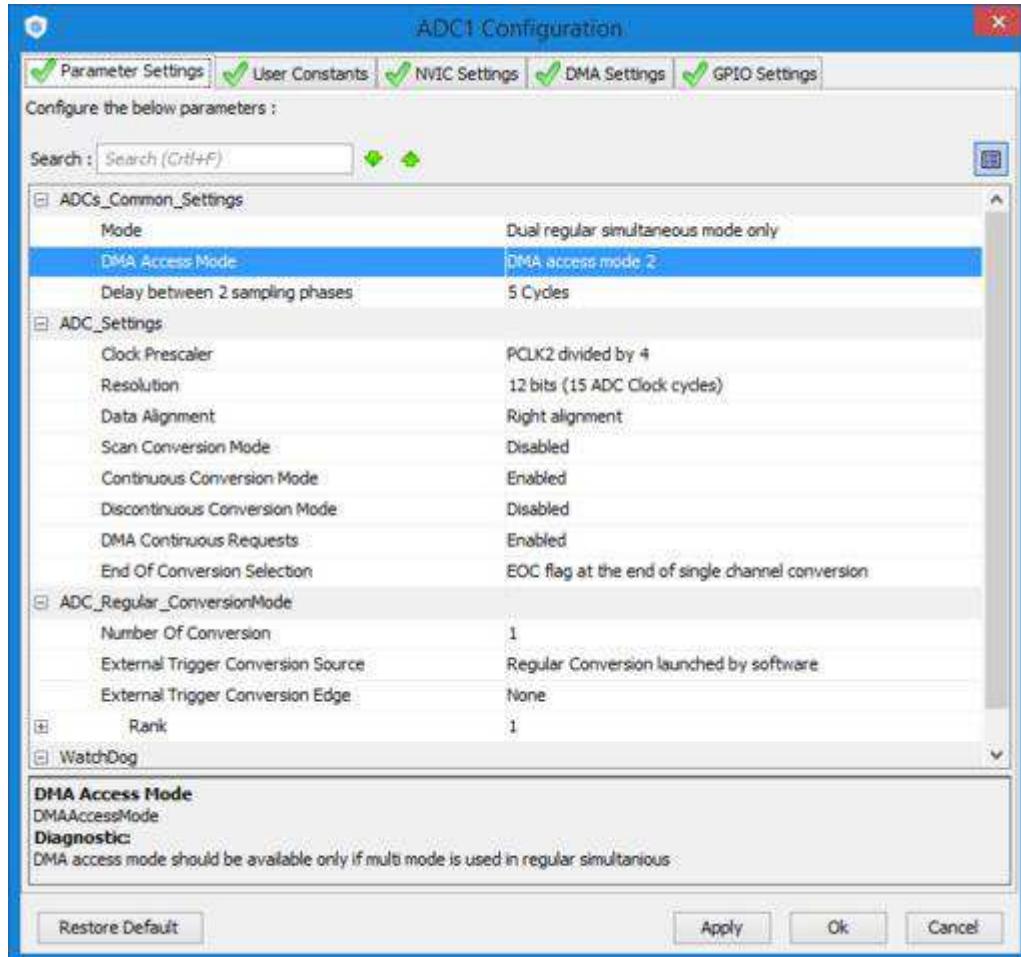
Hello everybody. I'm new with programming STM32F4 DISCO1, and now I have a problem with Dual regular simultaneous mode ADC + DMA. I would like to take a series of samples from ADC1 and ADC2 in Dual regular simultaneous mode. I generated code using CubeMX 4.19.0. I configured ADC1 and ADC2 in those pictures.

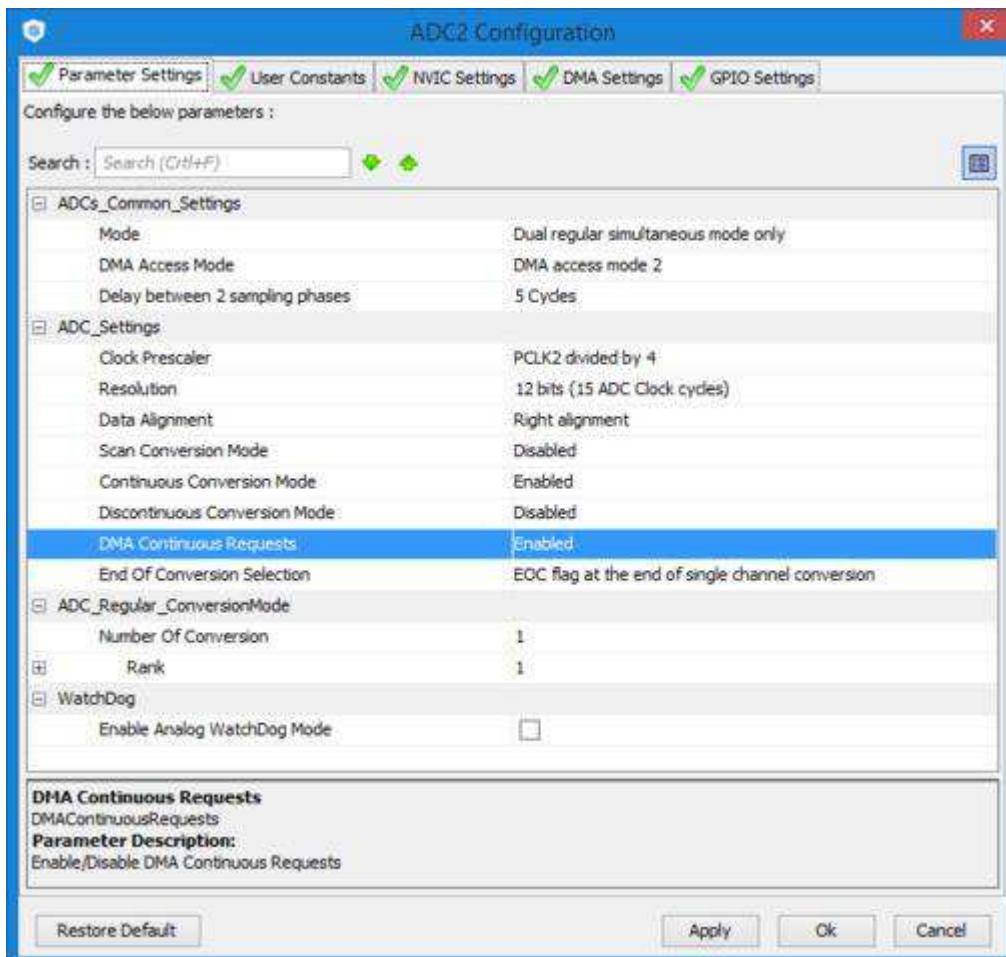
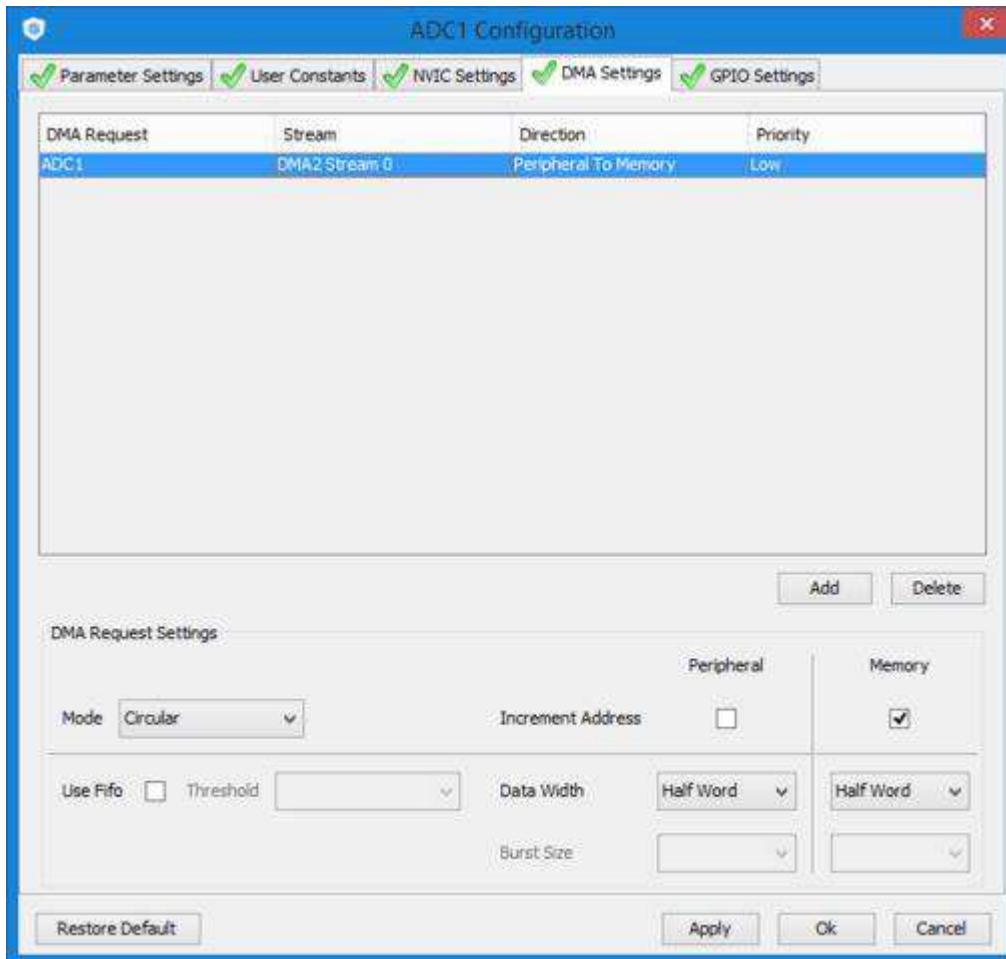
In KeilC, I just add a code

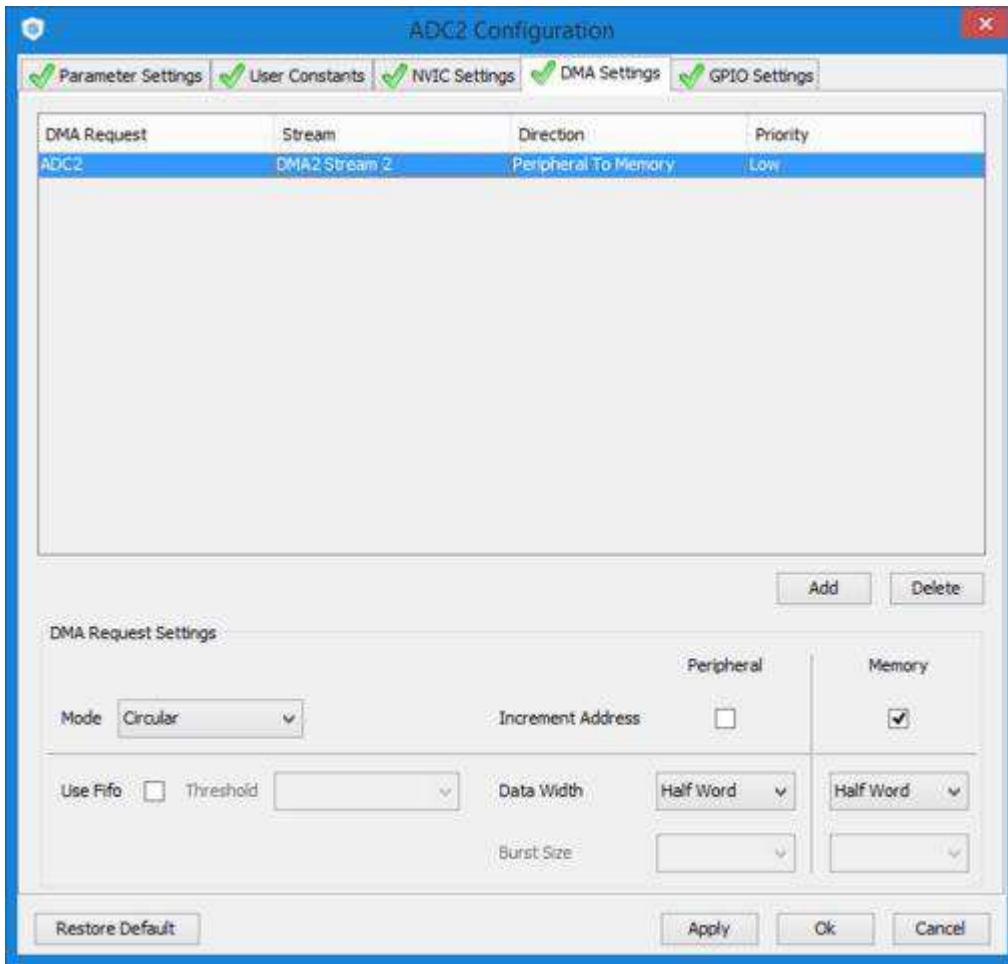
line HAL\_ADCEx\_MultiModeStart\_DMA(&hadc1,adc\_value,sizeof(adc\_value)); in User Code begin 2.  
The full project was added in the end of this post.

The problem is, when I debug it, I don't get the value that DMA transferred. The adc\_value = 0 all the time. I don't know why.

Thank you so much for your help.







Here are my code

```
/* Includes ----- */
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ----- */
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
DMA_HandleTypeDef hdma_adc1;
DMA_HandleTypeDef hdma_adc2;

uint32_t adc_value[4];
/* USER CODE BEGIN PV */
/* Private variables ----- */

/* USER CODE END PV */

/* Private function prototypes ----- */
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes ----- */
```

```
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration-----*/
/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* Configure the system clock */
SystemClock_Config();

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();
MX_ADC2_Init();

/* USER CODE BEGIN 2 */
HAL_ADCEx_MultiModeStart_DMA(&hadc1,adc_value,sizeof(adc_value));
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

}
/* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct;
RCC_ClkInitTypeDef RCC_ClkInitStruct;

/**Configure the main internal regulator output voltage
*/
__HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

/**Initializes the CPU, AHB and APB busses clocks
*/
}
```

```
RCC_OscInitStruct.OscillatorType = RCC OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}

/**Initializes the CPU, AHB and APB busses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{
    Error_Handler();
}

/**Configure the Systick interrupt time
*/
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

/**Configure the Systick
*/
HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

/* SysTick_IRQn interrupt configuration */
HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

ADC_MultiModeTypeDef multimode;
ADC_ChannelConfTypeDef sConfig;

/**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = DISABLE;
hadc1.Init.ContinuousConvMode = ENABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 1;
hadc1.Init.DMAContinuousRequests = ENABLE;
```

```
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
Error_Handler();
}

/**Configure the ADC multi-mode
*/
multimode.Mode = ADC_DUALMODE_REGSIMULT;
multimode.DMAAccessMode = ADC_DMAACCESSMODE_2;
multimode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_5CYCLES;
if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
{
Error_Handler();
}

/**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample
time.
*/
sConfig.Channel = ADC_CHANNEL_0;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}

/*
* ADC2 init function */
static void MX_ADC2_Init(void)
{
ADC_ChannelConfTypeDef sConfig;

/**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
*/
hadc2.Instance = ADC2;
hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc2.Init.Resolution = ADC_RESOLUTION_12B;
hadc2.Init.ScanConvMode = DISABLE;
hadc2.Init.ContinuousConvMode = ENABLE;
hadc2.Init.DiscontinuousConvMode = DISABLE;
hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc2.Init.NbrOfConversion = 1;
hadc2.Init.DMAContinuousRequests = ENABLE;
hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc2) != HAL_OK)
{
Error_Handler();
}

/**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample
time.
*/
sConfig.Channel = ADC_CHANNEL_1;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
```

```
if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
{
Error_Handler();
}

/***
* Enable DMA controller clock
*/
static void MX_DMA_Init(void)
{
/* DMA controller clock enable */
__HAL_RCC_DMA2_CLK_ENABLE();

/* DMA interrupt init */
/* DMA2_Stream0_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
/* DMA2_Stream2_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);

}

/** Configure pins as
* Analog
* Input
* Output
* EVENT_OUT
* EXTI
*/
static void MX_GPIO_Init(void)
{

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/***
* @brief This function is executed in case of error occurrence.
* @param None
* @retval None
*/
void Error_Handler(void)
{
/* USER CODE BEGIN Error_Handler */
/* User can add his own implementation to report the HAL error return state */
while(1)
{
}
/* USER CODE END Error_Handler */
}
```

```
#ifdef USE_FULL_ASSERT
```

```
/**  
 * @brief Reports the name of the source file and the source line number  
 * where the assert_param error has occurred.  
 * @param file: pointer to the source file name  
 * @param line: assert_param error line source number  
 * @retval None  
 */  
void assert_failed(uint8_t* file, uint32_t line)  
{  
/* USER CODE BEGIN 6 */  
/* User can add his own implementation to report the file name and line number,  
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */  
/* USER CODE END 6 */  
}  
  
#endif  
  
/**  
 * @}  
 */  
  
/**  
 * @}  
 */  
  
***** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```