

stm32f429-DMA Tutorial

제작자: 유영재

목차

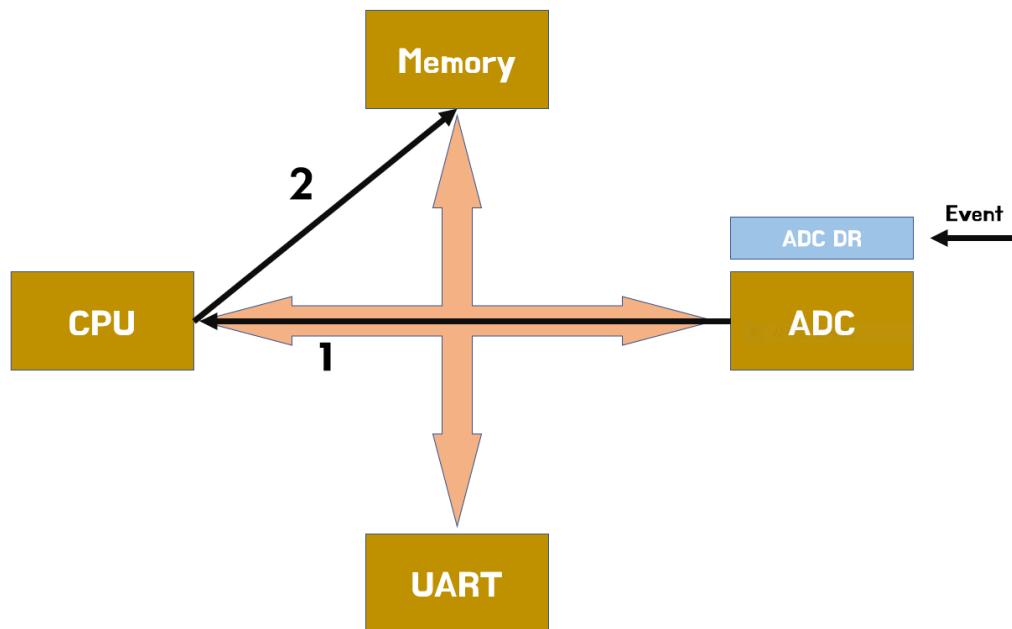
1.	DMA 소개	3
1.1	Master-slave system.....	3
1.2	DMA 사용 예	4
1.3	MCU Block Diagram	5
1.4	Bus matrix.....	10
1.5	Interrupt와 DMA 비교.....	16
2.	DMA 예제.....	21
2.1	LED Toggle.....	21
2.2	LED Toggle with Interrupt.....	28
2.3	DMA를 이용한 SRAM 간 데이터 공유	31
2.4	DMA1으로 했을 때의 오류 발생.....	34
2.5	DMA를 이용한 UART-SRAM 데이터 공유	35
3.	DMA 고급 개념.....	41
3.1	DMA Slave Port	41
3.2	DMA peripheral and memory port.....	42
3.3	DMA Stream	43

1. DMA 소개

- DMA는 Direct Memory Access의 약자로 최근 MCU에는 기본적으로 들어가 있는 Peripheral 중 하나다. 현재 우리가 다룰 arm-cortex M4 기반의 stm32f429 도 총 2개의 dma 컨트롤러가 내장되어 있다. **dma를 이해하기 위해선 arm-cortex의 버스 시스템을 이해하고 있어야 한다**

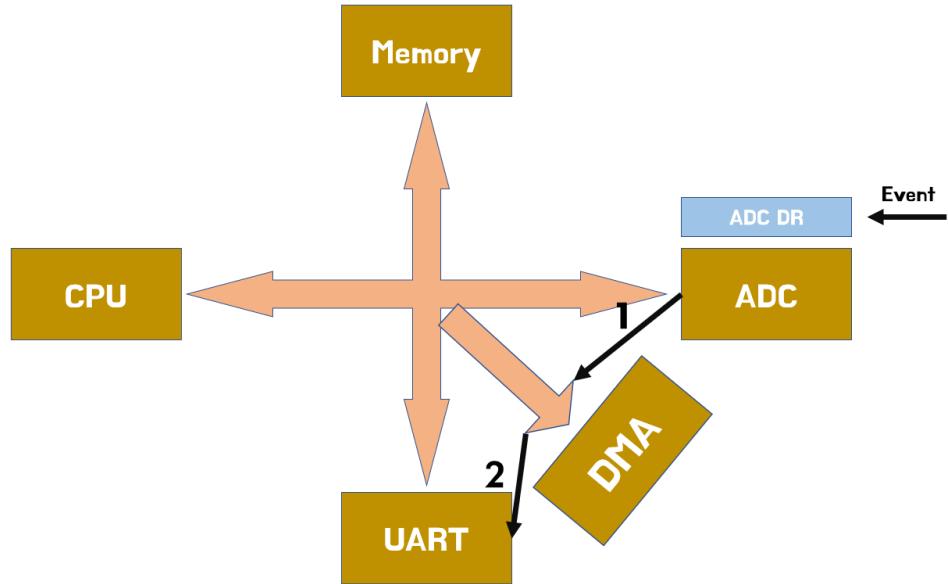
1.1 Master-slave system

아래 그림은 ARM의 버스 시스템을 나타낸 것이다. ARM 버스 시스템은 기본적으로 CPU와 주변 Peripheral들은 버스로 연결되어 있는 것을 볼 수 있다. 예를 들어 전역 변수에 ADC 데이터 레지스터 값을 저장하려고 한다



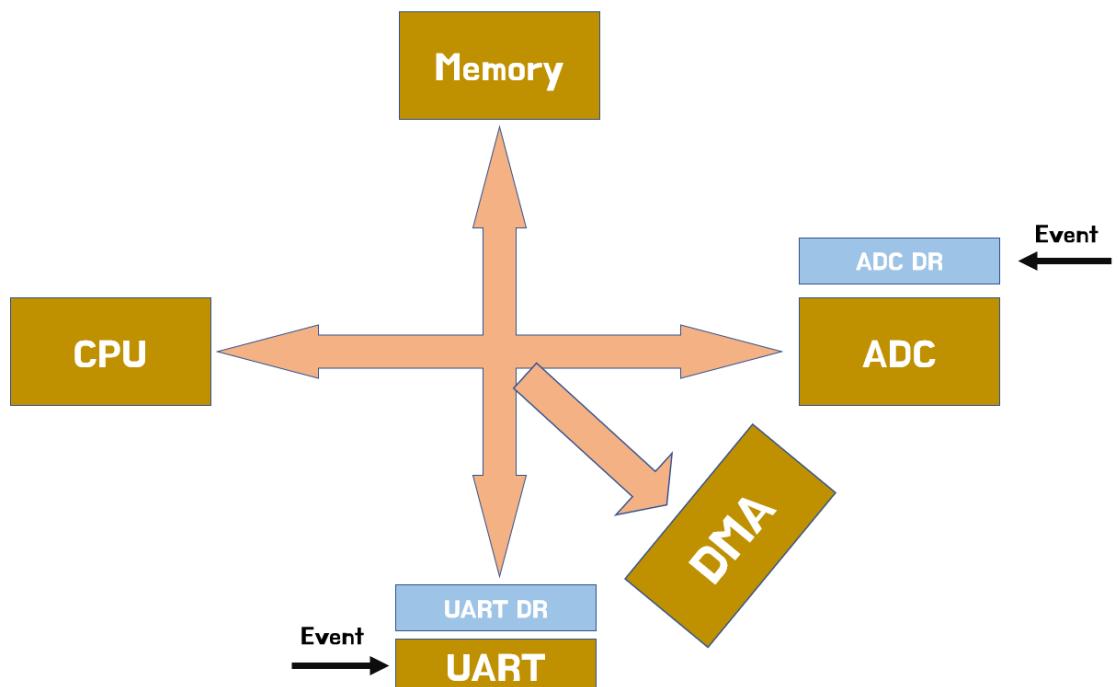
- A. 하나의 상황을 예를 들어서 설명을 해보려고 한다. ADC에 이벤트가 발생해서 digitize 된 값이 ADC 데이터 레지스터로 써지게 된다
- B. 그러면 ADC는 CPU에게 Event를 발생시켜서 CPU는 해당 데이터를 프로세서 레지스터에 저장한 이후 SRAM 메모리에 쓸 수 있게 된다
- C. 위에서 설명했듯이 총 2개의 경로로 데이터는 메모리에 써지게 된다. **그러면 바로 ADC DR 값이 메모리로 쓸 수는 없을까?**
- D. 답은 그렇게 할 수 없다. 왜냐하면 명령어 중 하나인 load는 CPU만 가지고 있기 때문이다. 즉 각 주변장치들은 두뇌가 없기 때문에 해당 명령을 실행할 수 없다. 그래서 매번 CPU의 도움을 받기 위해서 Event를 발생시키는 것이다
- E. 그래서 **버스 시스템에서 버스를 컨트롤 할 수 있는 기능을 가지고 있는 것은 master라고 부르며, 아닌 것은 slave라고 지칭한다**. 그래서 slave끼리는 버스를 끼고 데이터 공유가 불가능한 것이다
- F. 그러면 CPU 말고도 master 역할을 대신 할 수 있는 것은 없나? 그 역할을 하는 것이 바로 DMA controller다.

- G. DMA 컨트롤러가 master라고 지칭할 수 있어도, 명령어(instruction)를 제공하는 장치는 CPU밖에 존재하지 않는다. 다만 DMA가 하드웨어 로직으로 CPU의 일을 offload할 수 있게 설계되어 있을 뿐이다. 그래서 도움을 받아 버스를 control 할 수 있게 된다



1.2 DMA 사용 예

- DMA가 CPU load를 덜어주는 역할을 한다고 하지만, 언제 사용하게 되는가? 단순하게 CPU가 하나의 일만 수행을 한다면 굳이 사용할 필요는 없다. 다음과 같은 예를 들어보기로 한다



- A. 두 인터럽트가 동시에 config되어 있고, ADC, UART 모두 동시에 들어오는 상황이라고 가정을 한다. 인터럽트로 받은 데이터를 모두 메모리에 저장해야 하는 상황이다
- B. ADC 우선순위가 더 높은 상황이고, 데이터는 두 주변장치에서 지속적으로 들어오고 있다. 그래서 ADC는 데이터가 지속적으로 SRAM에 저장되지만, UART는 그렇지 않다
- C. 이럴 때 UART 데이터 손실을 막기 위해서 UART 데이터는 DMA 컨트롤러로 처리를 하게 하는 것이다. 따라서 DMA를 거쳐 SRAM으로 저장되게 된다

- 그리고 두번째 이유로 뒤에 실험을 하겠지만, ARM ON일 때와 ARM OFF/DMA ON일 때의 전류를 비교하면 차이가 생기게 된다. 전력을 아끼는 쪽이 DMA이기 때문에 좋은 Application에는 DMA 사용이 반드시 따라오게 된다

1.3 MCU Block Diagram

MCU Block Diagram은 DMA를 이해하는 데 반드시 숙지해야 하는 부분이다. RM 문서 2장 Memory and bus architecture을 보면 위에서 언급한 master/slave 개념이 드러나고 있다. stm32f429에는 2개의 DMA를 포함해서 10개의 master 버스가 존재한다

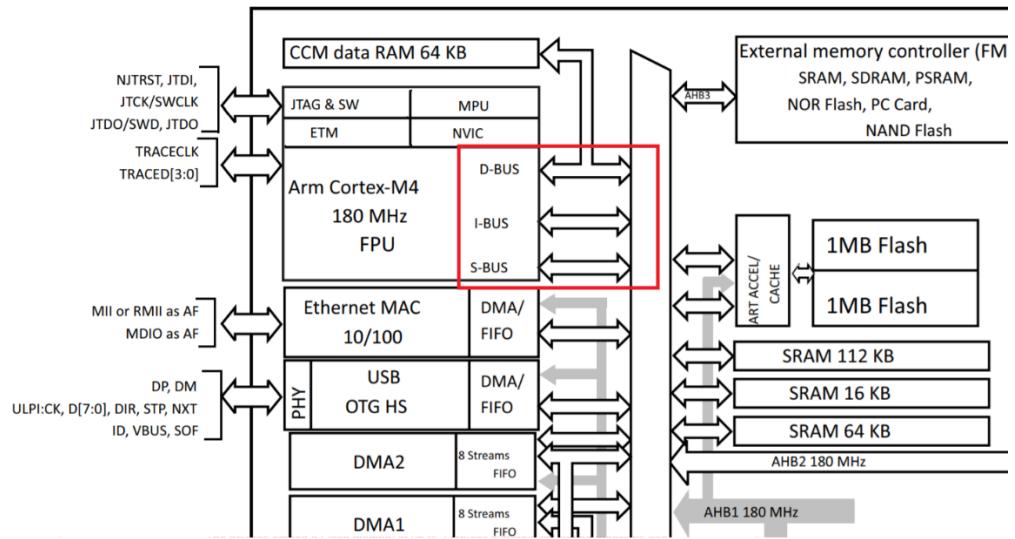
In the STM32F42xx and STM32F43xx devices, the main system consists of 32-bit multilayer AHB bus matrix that interconnects:

- Ten masters:
 - Cortex[®]-M4 with FPU core I-bus, D-bus and S-bus
 - DMA1 memory bus
 - DMA2 memory bus
 - DMA2 peripheral bus
 - Ethernet DMA bus
 - USB OTG HS DMA bus
 - LCD Controller DMA-bus
 - DMA2D (Chrom-Art Accelerator™) memory bus
- Eight slaves:
 - Internal Flash memory ICode bus
 - Internal Flash memory DCode bus
 - Main internal SRAM1 (112 KB)
 - Auxiliary internal SRAM2 (16 KB)
 - Auxiliary internal SRAM3 (64 KB)
 - AHB1peripherals including AHB to APB bridges and APB peripherals
 - AHB2 peripherals
 - FMC

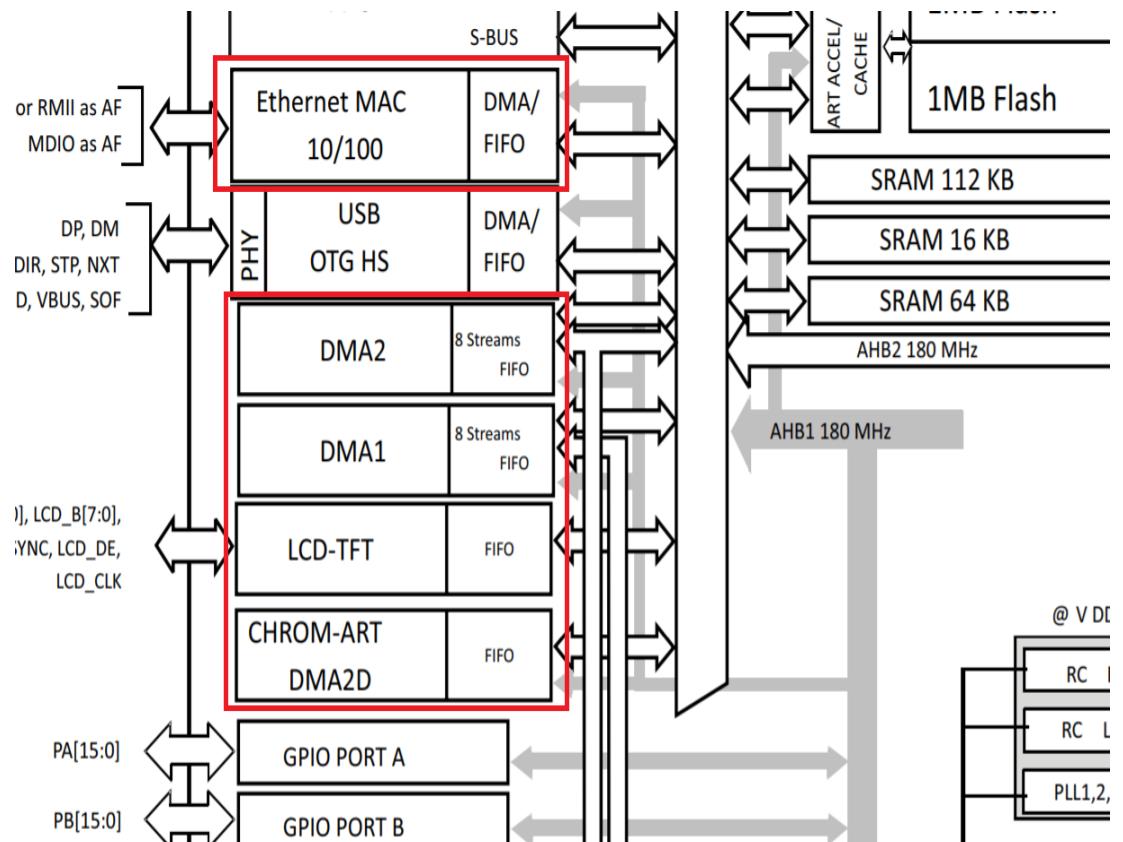
The bus matrix provides access from a master to a slave, enabling concurrent access and efficient operation even when several high-speed peripherals work simultaneously. The 64-Kbyte CCM (core coupled memory) data RAM is not part of the bus matrix and can be accessed only through the CPU. This architecture is shown in [Figure 2](#).

A. Master

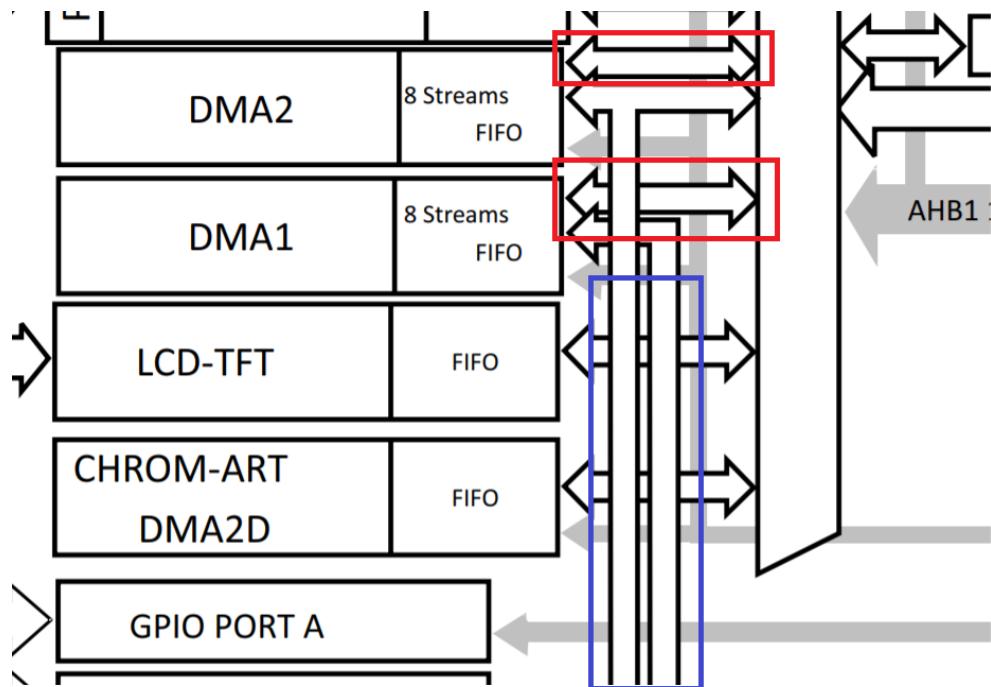
a. Core I-bus, S-bus, D-bus



b. 다른 Master



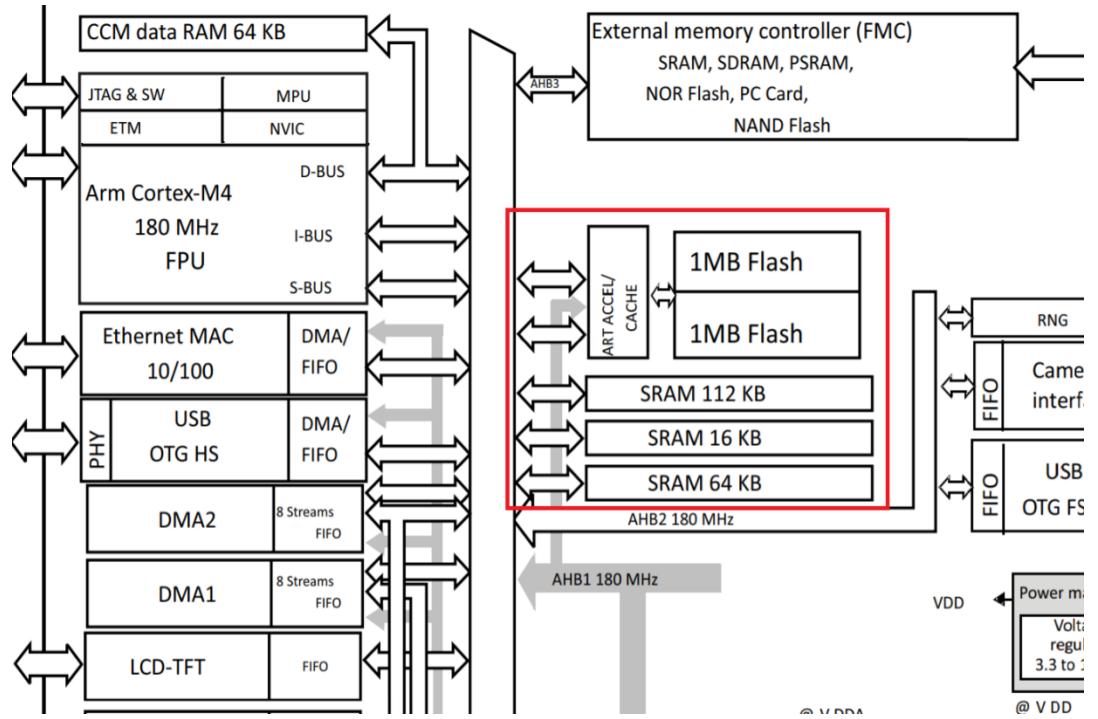
참고로 DMA controller에서 두 갈래의 길로 나눠진다. 하나는 memory쪽 버스와 나머지는 peripheral쪽 버스가 된다. 아래 버스에서 빨간색이 memory, 파란색이 peripheral로 해당된다



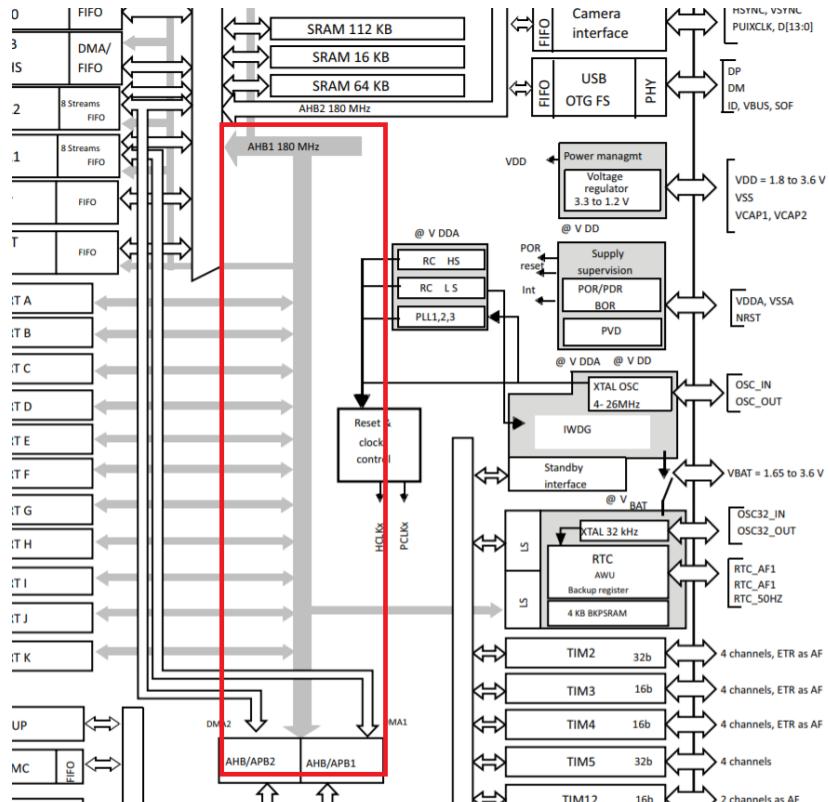
따라서 DMA 컨트롤러를 통해서 M2M, P2M, M2P가 가능하게 된다

B. Slave

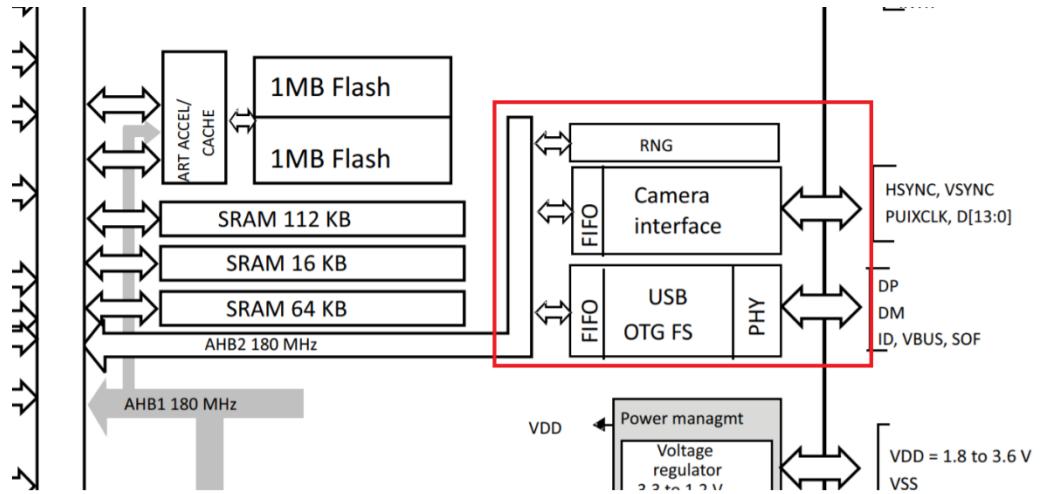
a. Flash I-code, D-code, SRAM



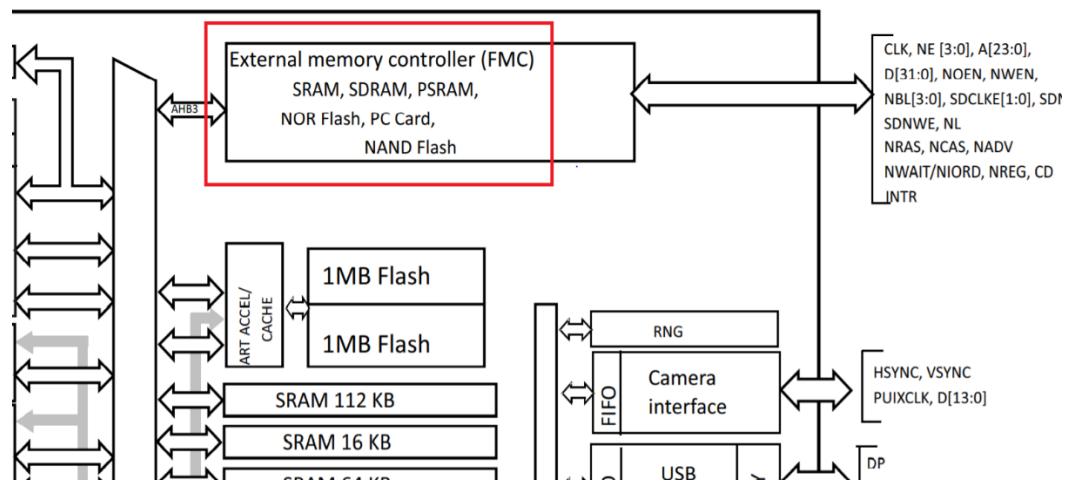
b. AHB1은 APB1, 2 버스를 포함하고 있다



c. AHB2



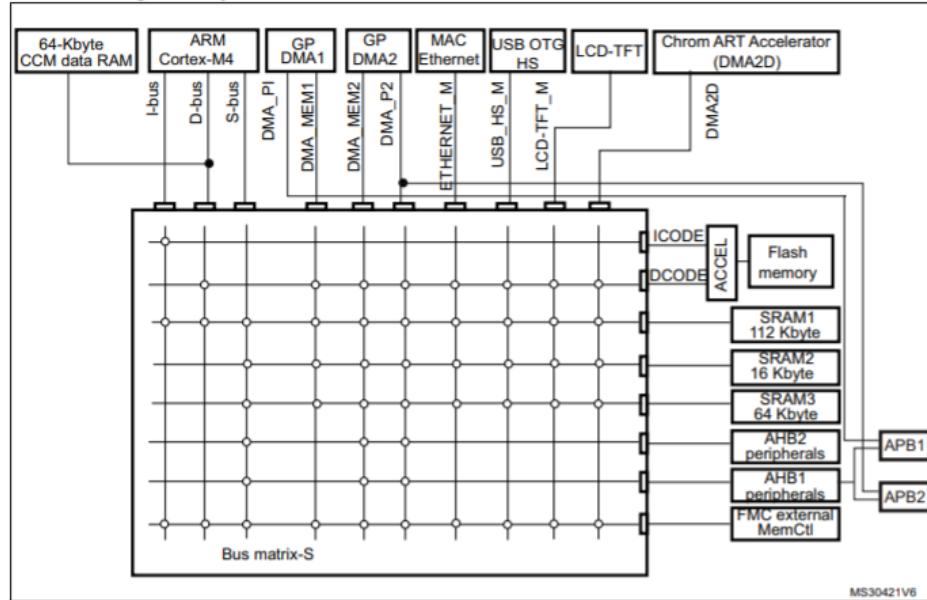
d. FMC



1.4 Bus matrix

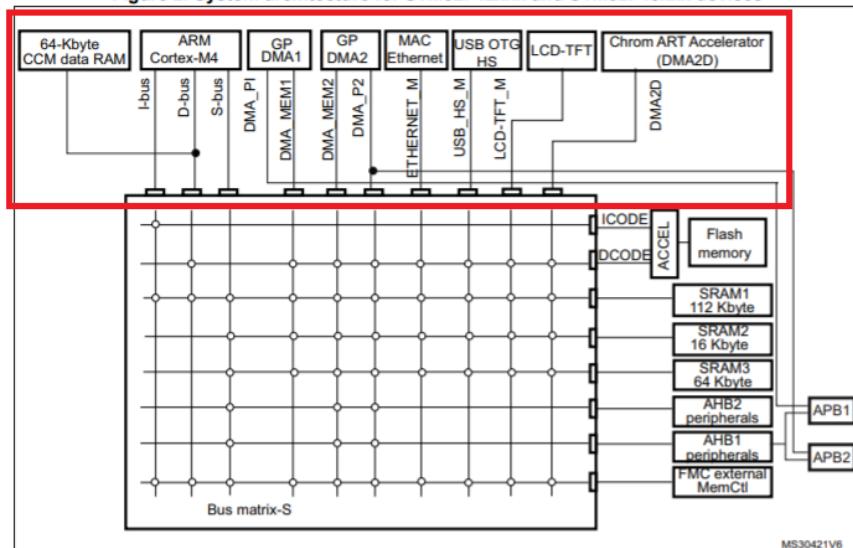
- RM 데이터시트에는 바로 밑 페이지에 위 내용을 matrix 형식으로 정리한 diagram을 제공하고 있다. 아래 matrix로 DMA를 좀 더 쉽게 이해해 보고자 한다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



A. Master

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices

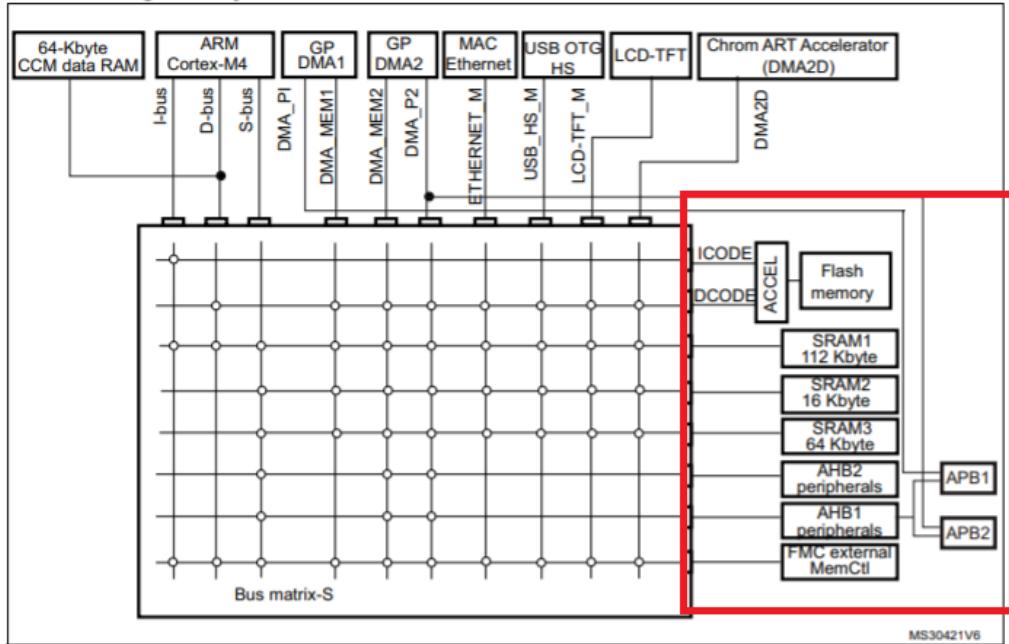


Q. ARM과 DMA를 제외한 나머지들이 master 역할을 할 수 있는 이유는?

- in this MCU, we have advance peripherals that uses an advanced bus matrix interconnect on which the USB, TFT, Ethernet acts as a master which means it can do its own memory transfer based on the software configuration.

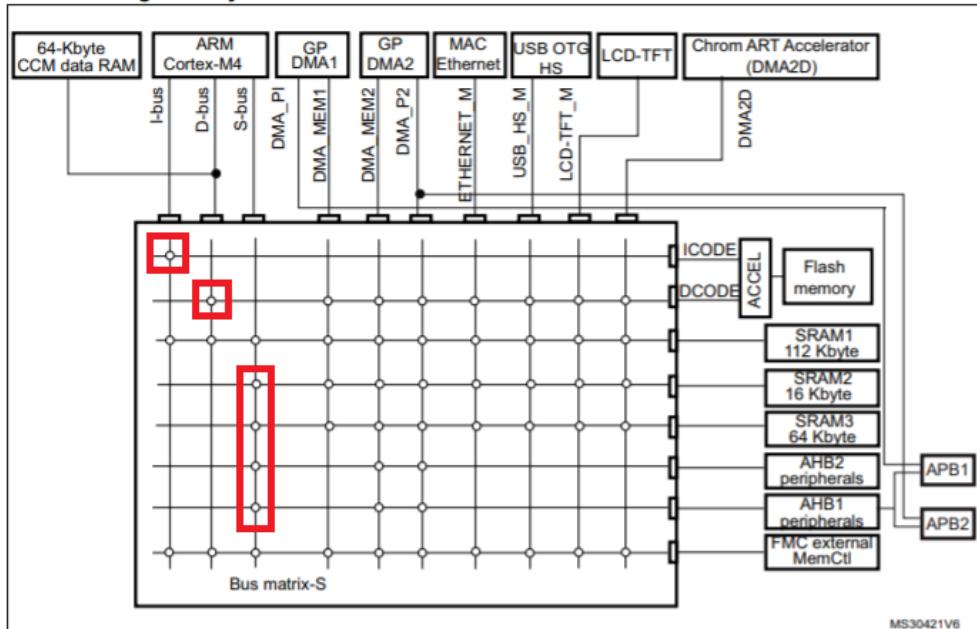
B. Slave

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



C. Matrix 보는 법

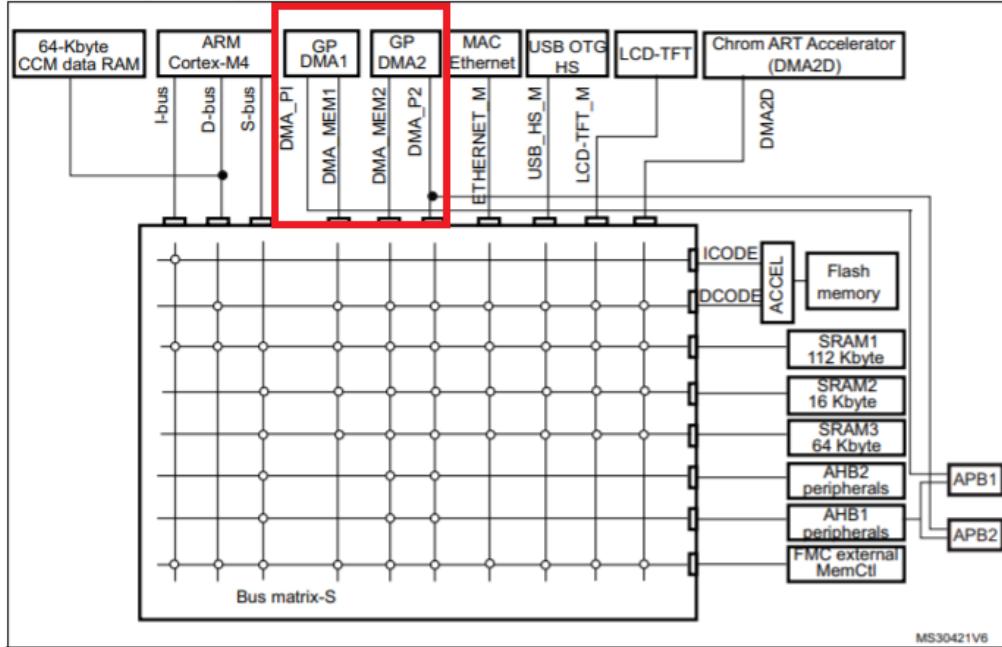
Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



빨간색 박스가 쳐진 부분은 연결된 부분을 뜻한다. 따라서 ARM 코어를 기반으로 보면 Flash 메모리와는 I-bus와 D-bus로 내용을 주고받는 것을 볼 수 있다. 또한 SRAM2, SRAM3, rr 다른 peripheral bus에는 S-bus만 접근 가능한 것을 확인할 수 있다

D. DMA Matrix

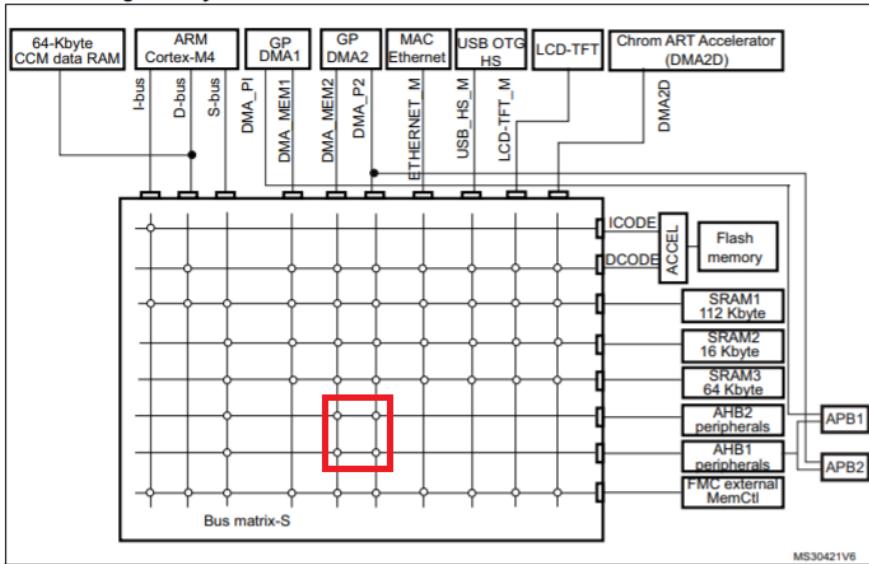
Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



stm32f429에는 총 2개의 DMA 컨트롤러가 존재하고 있다. 각 컨트롤러에는 DMA_P1, DMA_MEM1 2가지 줄기가 뻗어져 있다. 선을 잘 따라가보면 Px 라인에는 APB1, 2와 연결되어 있다. 추측해보면 peripheral과 memory 사이의 처리는 dma 컨트롤러를 통해서 처리될 수 있는 것으로 보인다.

또한 DMA2는 AHB 사이의 처리도 담당해줄 수 있는 것으로 보인다. 다만 DMA1은 P2M(Peripheral to Memory)만 가능한 것을 볼 수 있다

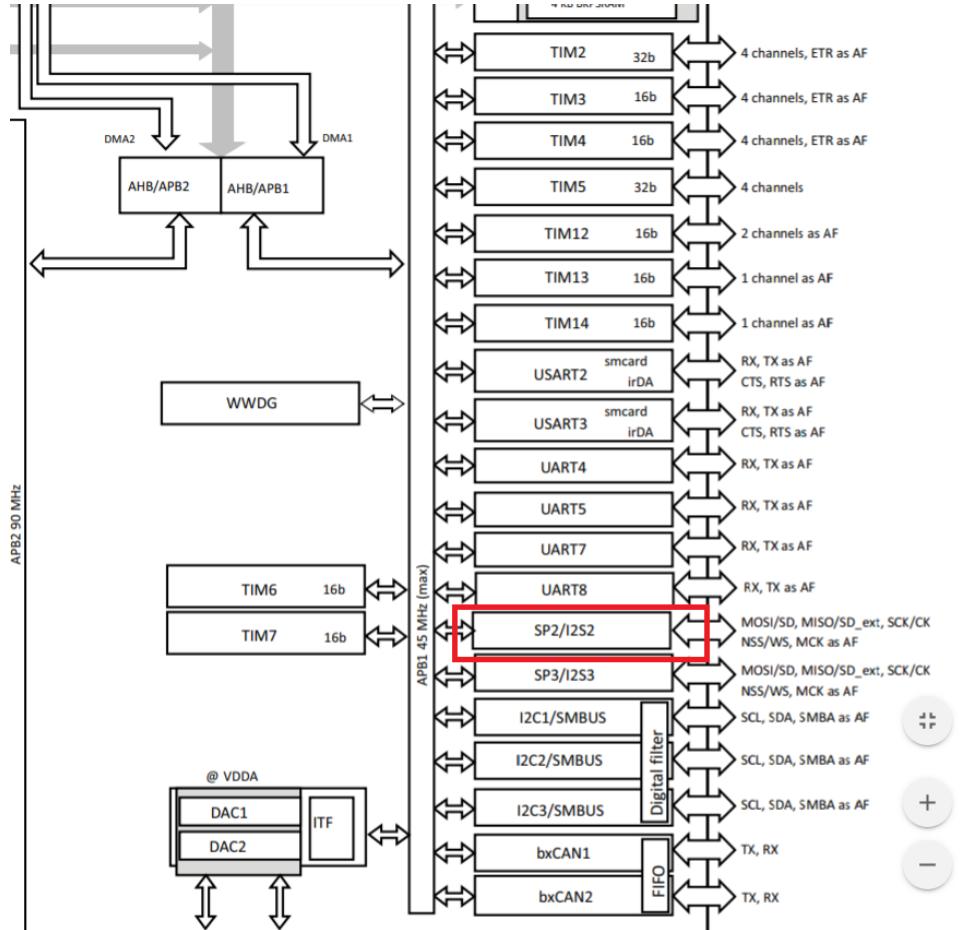
Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



E. 상황 예제

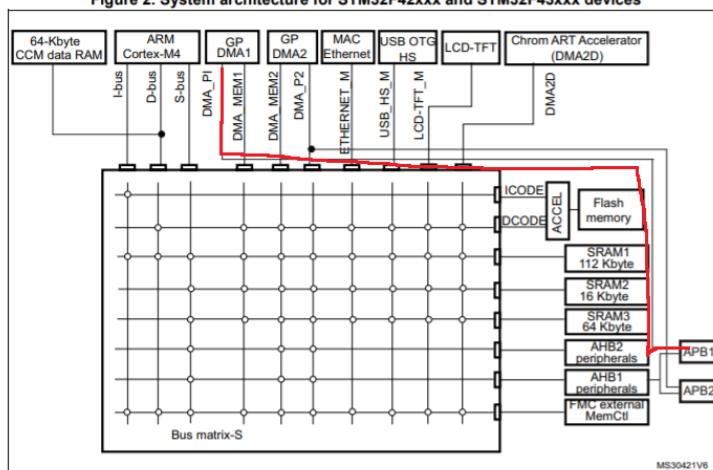
예를 들어 SPI2의 데이터를 SRAM1으로 쓰고 싶을 때 어떻게 DMA Path는 구성될 수 있을까?

먼저 SPI2는 APB1 버스에 연결된 것을 확인할 수 있다



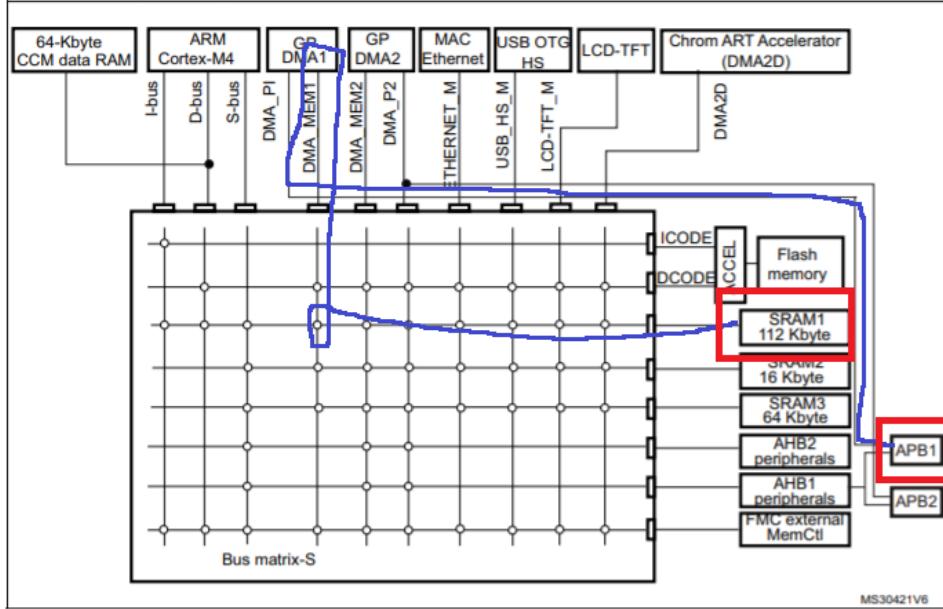
Bus matrix로 봤을 때 APB1은 DMA1_P1과 연결되어 있는 것을 볼 수 있다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



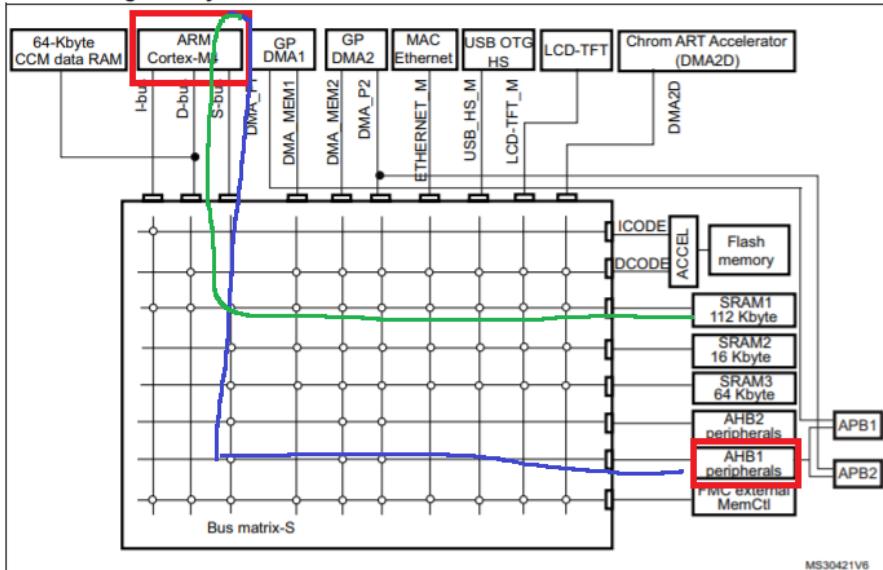
그러면 다음과 같은 path로 DMA1을 통해서 SRAM1 데이터 저장이 가능하다. 다음 path를 확인하면 ARM 코어를 전혀 거치지 않는다는 점을 확인할 수 있다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



F. ARM 코어를 통한 Path (SPI2, AHB1 peripheral)

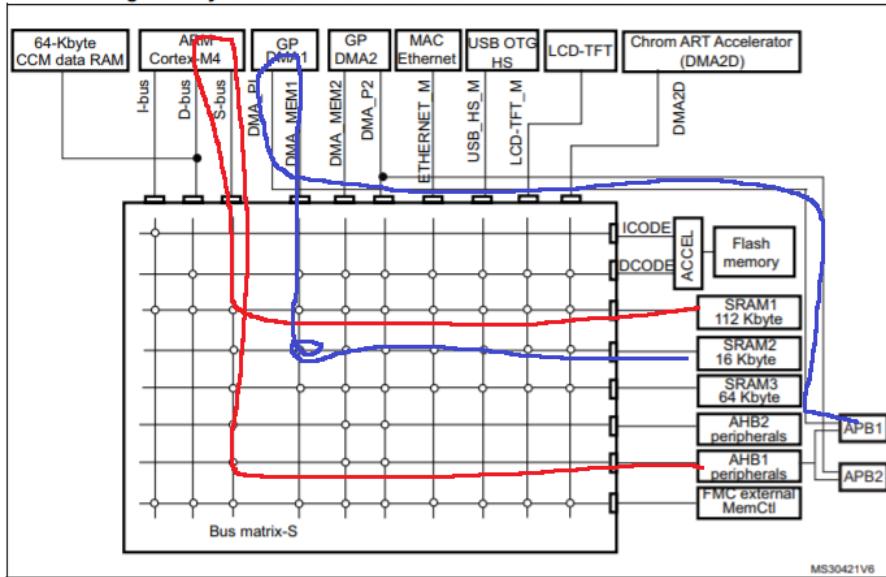
Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



G. DMA와 ARM 코어를 이용한 동시 데이터 전달

위의 내용을 종합해보면 DMA는 ARM과 별개로 데이터 처리를 할 수 있도록 되어 있다. 예를 들어, 1) **UART 데이터를 SRAM1에 저장** 2) **ADC 데이터를 SRAM2에 저장**이 아래 Path와 같이 동시에 별도로 동작이 가능하다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



H. 만일 RAM 메모리가 SRAM1 하나만 존재한다면, 동시에 SRAM1에 저장하는 길을 동시에 이용하게 된다. 순간적인 동시 이용은 불가능하기 때문에 priority가 높은 master가 먼저 bus를 선점하게 된다

이렇게 되면 데이터를 잃을 수도 있지만, DMA에는 따로 FIFO가 존재하기 때문에 손실없이 bus의 기회가 올때까지 기다리게 된다

1.5 Interrupt와 DMA 비교

- 두 프로그램을 구성해서 비교해보려고 한다. 하나는 ARM 코어가 지속적으로 while 무한루프를 수행하고 있는데, 인터럽트로 코어가 다른 일을 처리하게끔 하는 것이다. 나머지는 DMA로 코어가 일을 멈추는 것 없이 DMA 컨트롤러에 일을 맡기는 것이다
- User Application은 SRAM1에 지속적으로 데이터를 쓰는 작업을 하는 것이다

```
/* USER CODE BEGIN 0 */
uint8_t src_data[50];
#define OFF_SET 0X500
#define DEST_ADDRESS (volatile uint8_t*) (SRAM1_BASE + OFF_SET)

for( uint32_t i = 0 ; i < 50 ; i++)
{
    src_data[i] = 0xAA;
}

// receive 250 bytes
HAL_UART_Receive_IT(&huart1,(uint8_t*)SRAM2_BASE, 250);
/* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
        HAL_GPIO_WritePin(GPIOG,GPIO_PIN_13,GPIO_PIN_SET);

        for( i = 0 ; i < 50 ; i++)
        {
            *(DEST_ADDRESS+i) = src_data[i];

        }
        HAL_GPIO_WritePin(GPIOG,GPIO_PIN_13,GPIO_PIN_RESET);
    }
}
}
```

- A. Logic 분석기로 총 3가지 신호를 파악하려고 한다. While 루프에서 GPIOG_13에 대한 set, unset을 보려고 한다
- B. 그리고 UART 인터럽트를 1바이트 받을 때마다, 250바이트를 모두 받을 때까지의 신호를 측정하려고 한다

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */
    // UNUSED(huart);
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);

    HAL_UART_Receive_IT(&huart1,(uint8_t*)SRAM2_BASE, 250);
    /* NOTE: This function Should not be modified, when the callback is
needed,
        the HAL_UART_TxCpltCallback could be implemented in the user file
    */
}
```

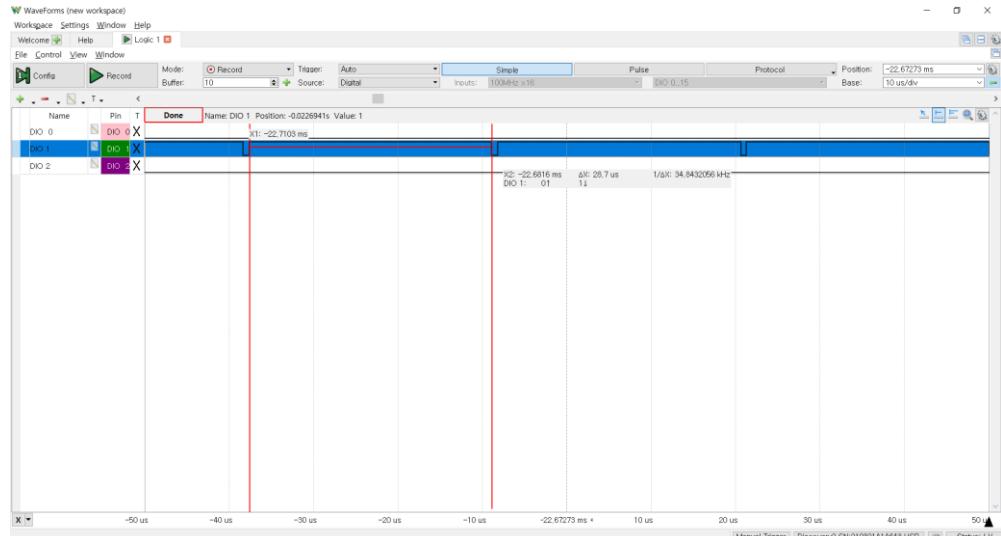
```

void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQHandler_0 */
    HAL_GPIO_WritePin(GPIOG,GPIO_PIN_12,GPIO_PIN_SET);
    /* USER CODE END USART1_IRQHandler_0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQHandler_1 */
    HAL_GPIO_WritePin(GPIOG,GPIO_PIN_12,GPIO_PIN_SET);
    /* USER CODE END USART1_IRQHandler_1 */
}

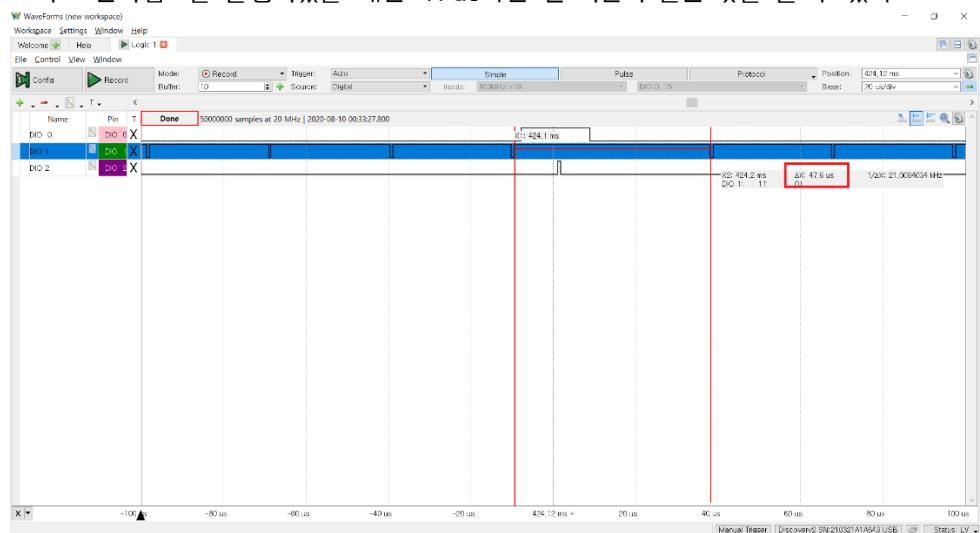
```

- C. 코드를 다운로드 한 후 ARM 코어가 SRAM2에 Write를 하는 시간을 측정해보려고 한다. Sample Rate는 측정하고자 하는 주파수의 5배로 지정한다. 현재 16MHz의 클럭으로 동작하고 있기 때문에, 여러 실험을 통해 약 5MHz 이상의 Sample Rate에서 정확한 측정을 할 수 있었다. 그렇지 않으면 신호를 놓치게 되었다

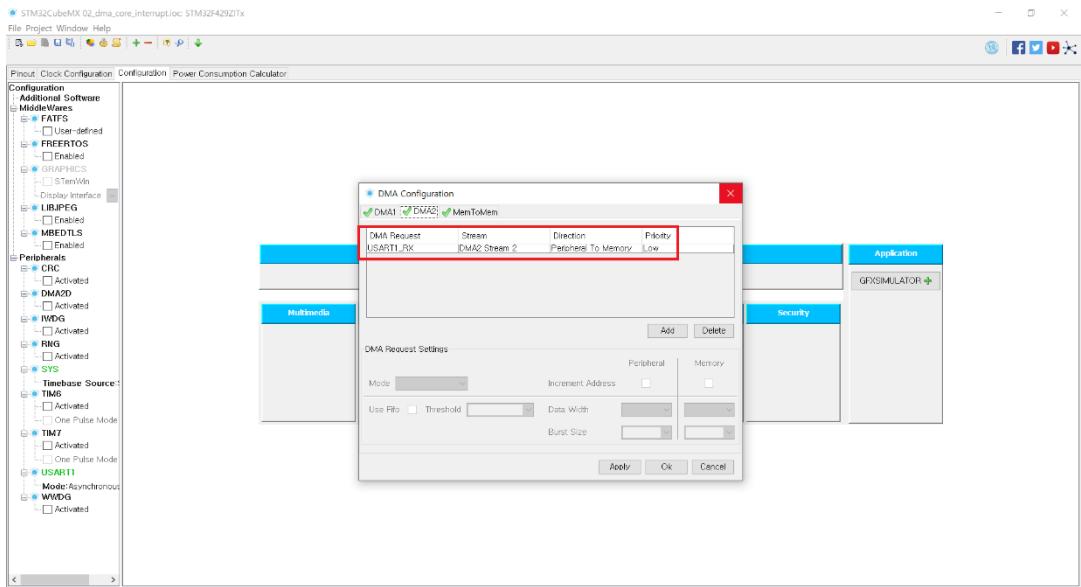
따라서 아래 측정 결과를 보면 인터럽트가 없을 때는 약 29us에 SRAM1에 데이터를 모두 쓰게 된다



- D. 그리고 인터럽트를 발생시켰을 때는 47us라는 긴 시간이 걸린 것을 볼 수 있다



- E. 이제는 DMA 코드를 넣어서 실행해보려고 한다. 먼저 CubeMX로 설정을 하도록 한다. 자세한 내용은 이후에 다루려고 한다



- F. 그리고 소스코드를 인터럽트가 아닌 DMA로 받도록 한다

```

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */
    // UNUSED(huart);
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);

    HAL_UART_Receive_DMA(&huart1, (uint8_t*)SRAM2_BASE, 250);
    /* NOTE: This function Should not be modified, when the callback
     * is needed,
     * the HAL_UART_TxCpltCallback could be implemented in the
     * user file
    */

    // HAL_UART_Receive_IT(&huart1, (uint8_t*)SRAM2_BASE, 250);
    HAL_UART_Receive_DMA(&huart1, (uint8_t*)SRAM2_BASE, 250);
    /* USER CODE END 2 */

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

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_SET);

        for( i = 0 ; i < 50 ; i++)
        {
            *(DEST_ADDRESS+i) = src_data[i];
        }
    }
}

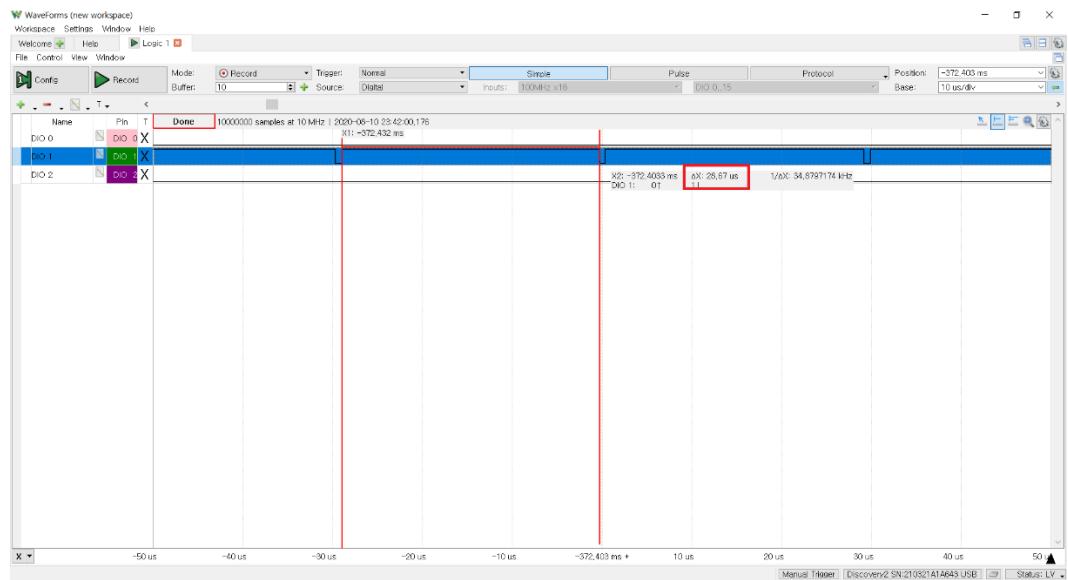
```

```

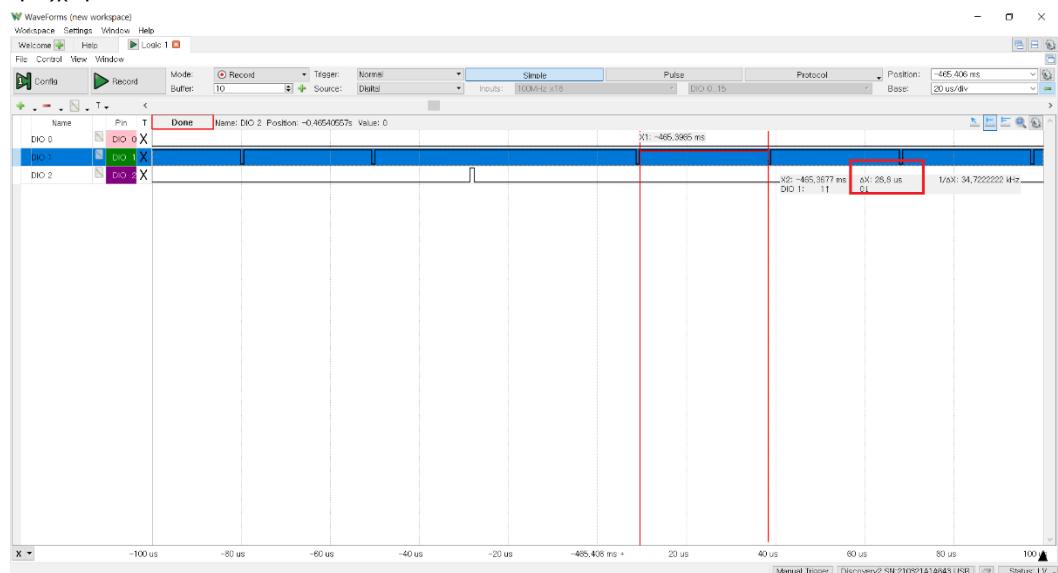
    }
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);
}

```

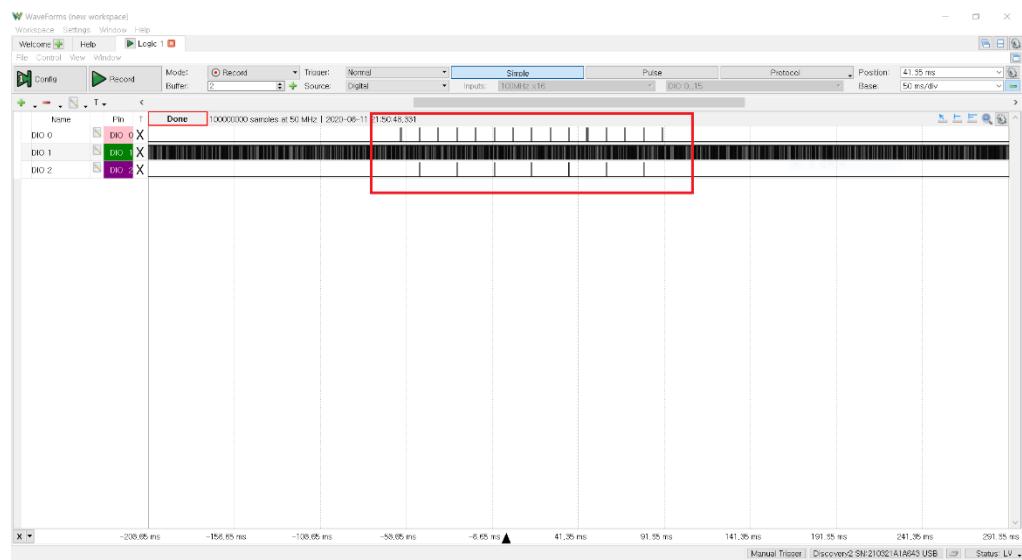
프로그램을 다운로드 한 후 측정을 하면 UART 데이터를 넣지 않았을 때는 28us로 기존 인터럽트와 비슷했다



G. DMA를 통해 지속적으로 250바이트를 송신해도 while문 안에는 28us를 유지하는 것을 볼 수 있다



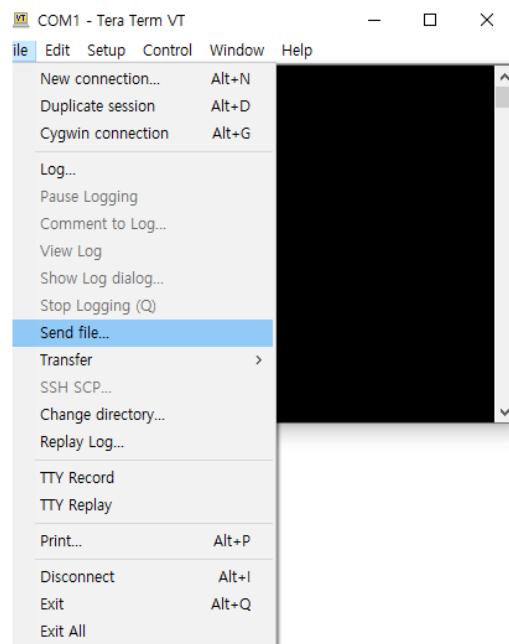
그리고 인터럽트 때만 ARM 코어가 DMA handler를 처리하는 것을 볼 수 있다. 코드 흐름 자체는 DMA 핸들러가 모든 바이트를 받은 후 UART로 전달한다는 것을 디버깅을 통해서 확인 할 수 있었다. 그리고 DMA 핸들러는 UART 수신보다 2배로 반응하는 것을 알 수 있었다. 이는 ARM 코어에서 처리되는 부분이기 때문이다



정확한 내용은 추후에 알아보려고 한다 (DMA IRQ Handler는 half 때마다 한 번씩 들어온다. 그리고 안에서 콜백 함수가 등록될 때 half complete 때라면 콜백 함수를 호출하는 구조다. 따라서 실제로는 2배로 신호가 뛰는 것이다)

● 참고

시리얼을 통한 파일 전송을 tera term의 send file을 이용하도록 한다

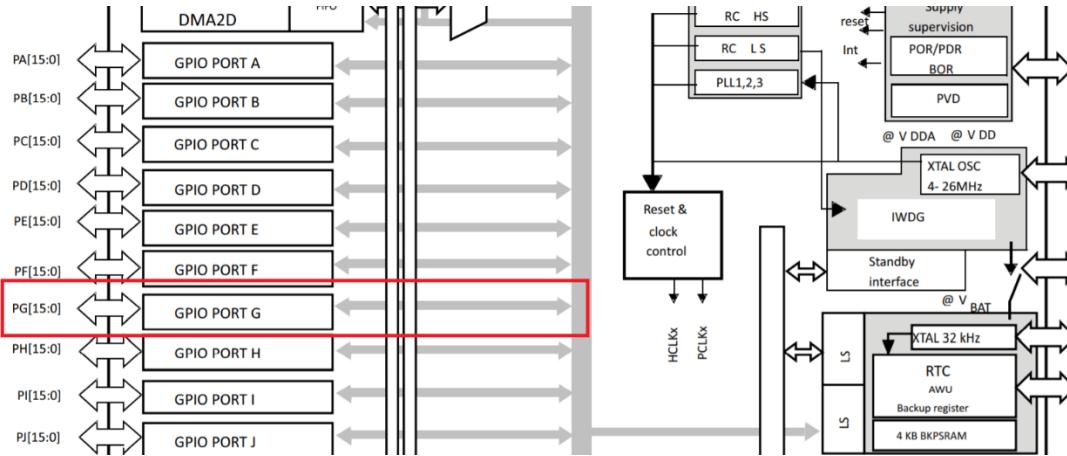


2. DMA 예제

- 간단히 DMA 예제를 작성해서 DMA API와 Bus Matrix를 이해해보려고 한다

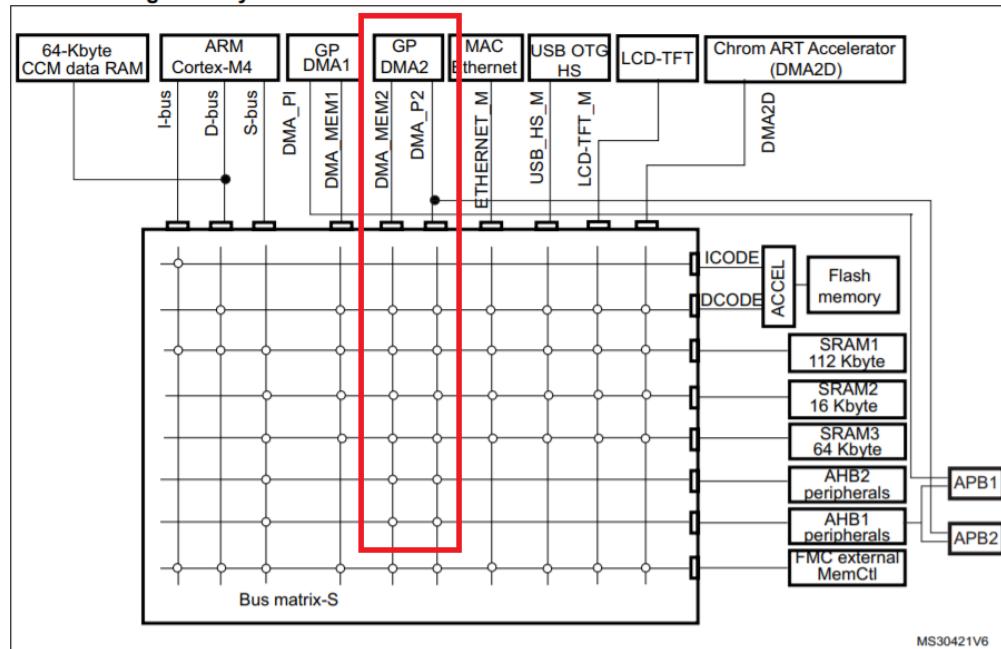
2.1 LED Toggle

- SRAM 데이터를 LED와 연결되어 있는 GPIO 데이터 레지스터를 쓰려고 한다. stm32f429의 GPIO는 AHB1 버스에 연결되어 있다



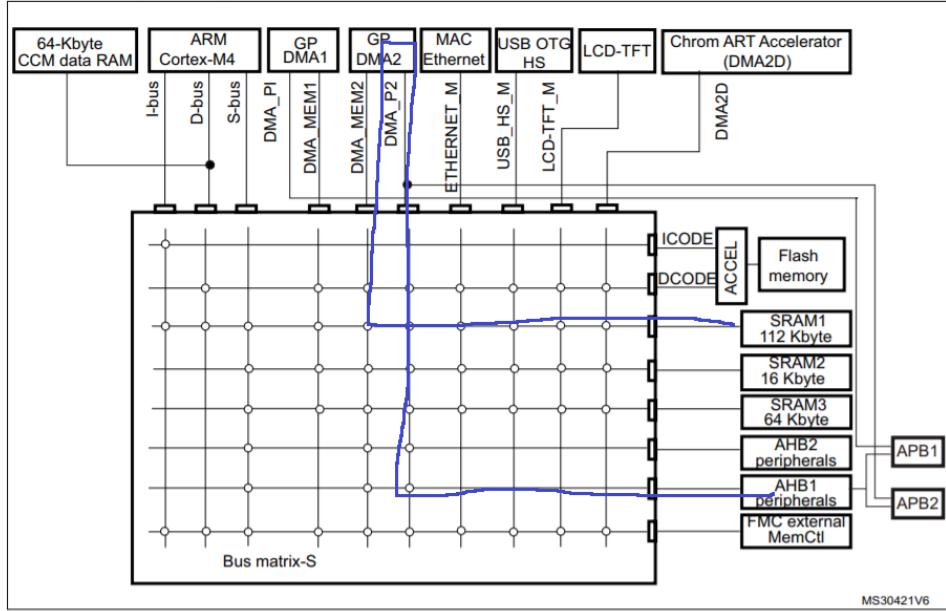
- 그러면 SRAM과 AHB1 버스와 통신할 수 있는 DMA Controller를 알아봐야 한다

Figure 2. System architecture for STM32F429xx and STM32F439xx devices

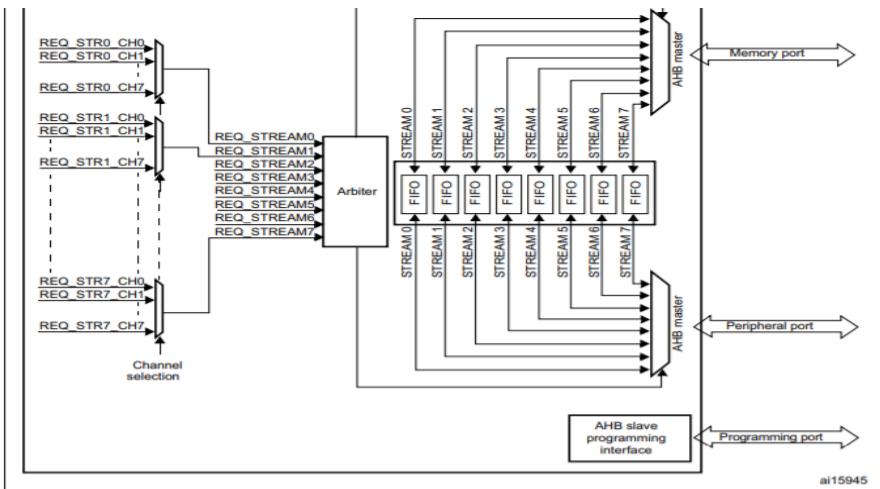


그래서 matrix 길을 따라가보면 다음과 같이 표현할 수 있다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



- Bus matrix에서 DMA는 2가지 포트가 나와있는 것을 볼 수 있다. 하나는 Memory, 나머지는 Peripheral 포트다. 이는 MCU 벤더에 따라 이름 짓는 것이 자유인데, 확장해보면 주변장치는 곧 메모리 mapped로 바꿔 말할 수 있다. 따라서 의미상 나눈 것으로 확인된다



The DMA controller performs direct memory transfer: as an AHB master, it can take the control of the AHB bus matrix to initiate AHB transactions.

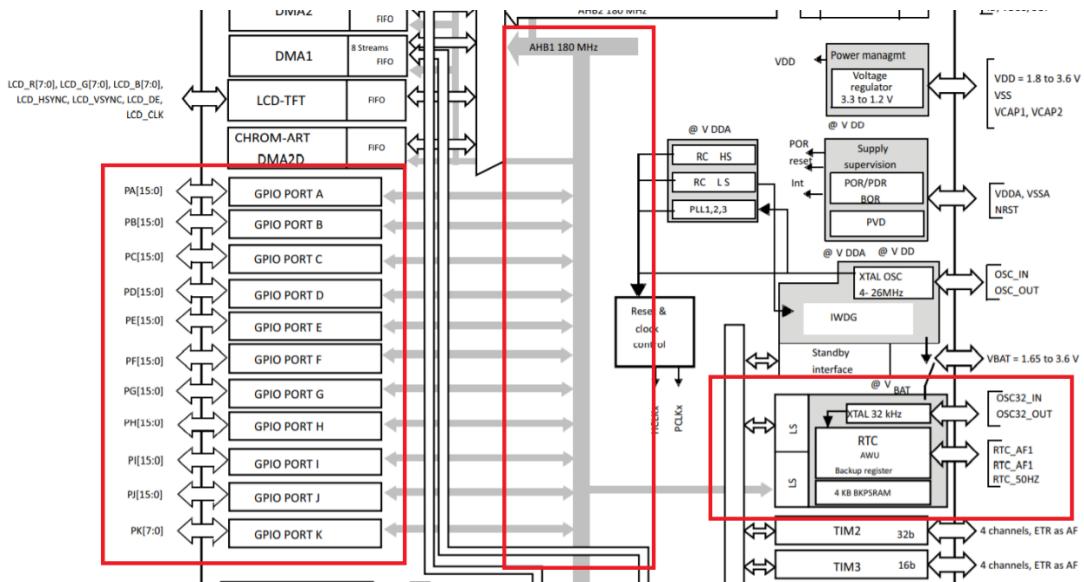
It can carry out the following transactions:

- peripheral-to-memory
- memory-to-peripheral
- memory-to-memory

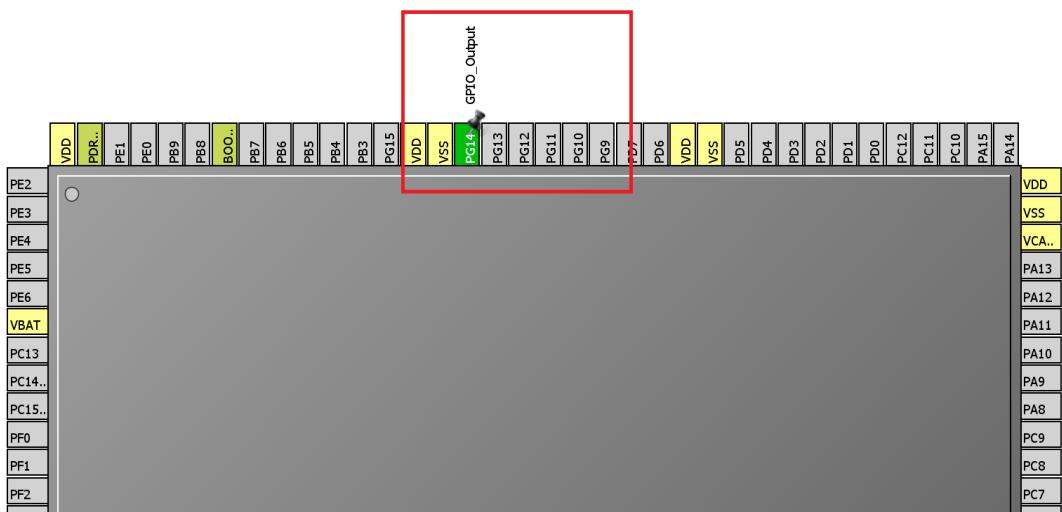
The DMA controller provides two AHB master ports: the *AHB memory port*, intended to be connected to memories and the *AHB peripheral port*, intended to be connected to peripherals. However, to allow memory-to-memory transfers, the *AHB peripheral port* must also have access to the memories.

The AHB slave port is used to program the DMA controller (it supports only 32-bit accesses).

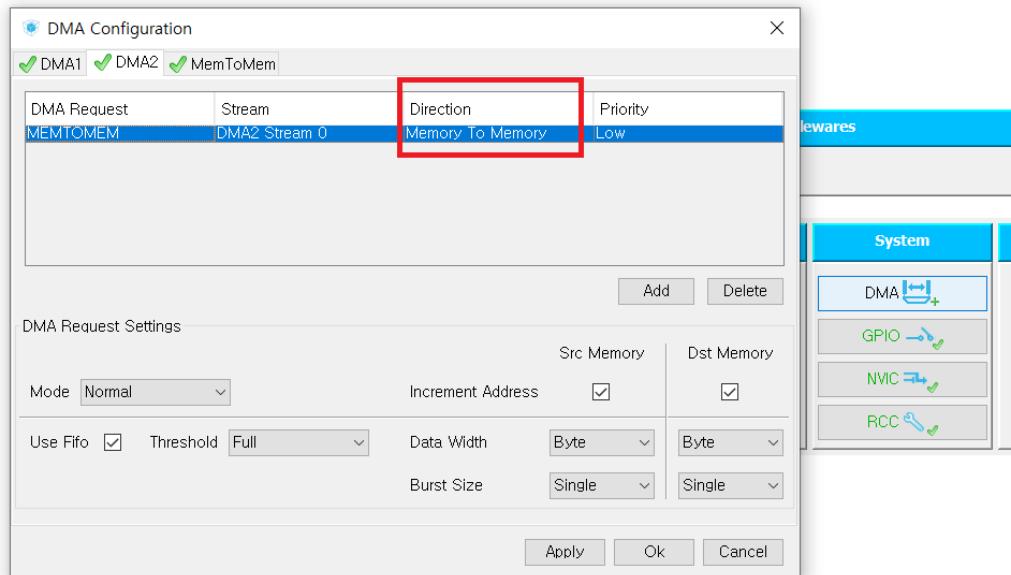
- 데이터시트에서는 추가적으로 Peripheral 포트는 곧 memory에도 접근할 수 있다고 나와있기 때문에 어느정도 위의 말과 일맥상통한다. 따라서 DMA2로는 APB2와 연결된 메모리를 제외하고 접근이 가능하다
- 헷갈렸던 부분이 AHB1이 APB1, 2와 합쳐 있는 것으로 알았는데, 그것이 아니라 AHB1에 단독으로 연결되어 있는 주변장치가 있다. 아래 표시해둔 것이 그러한 주변장치들이다



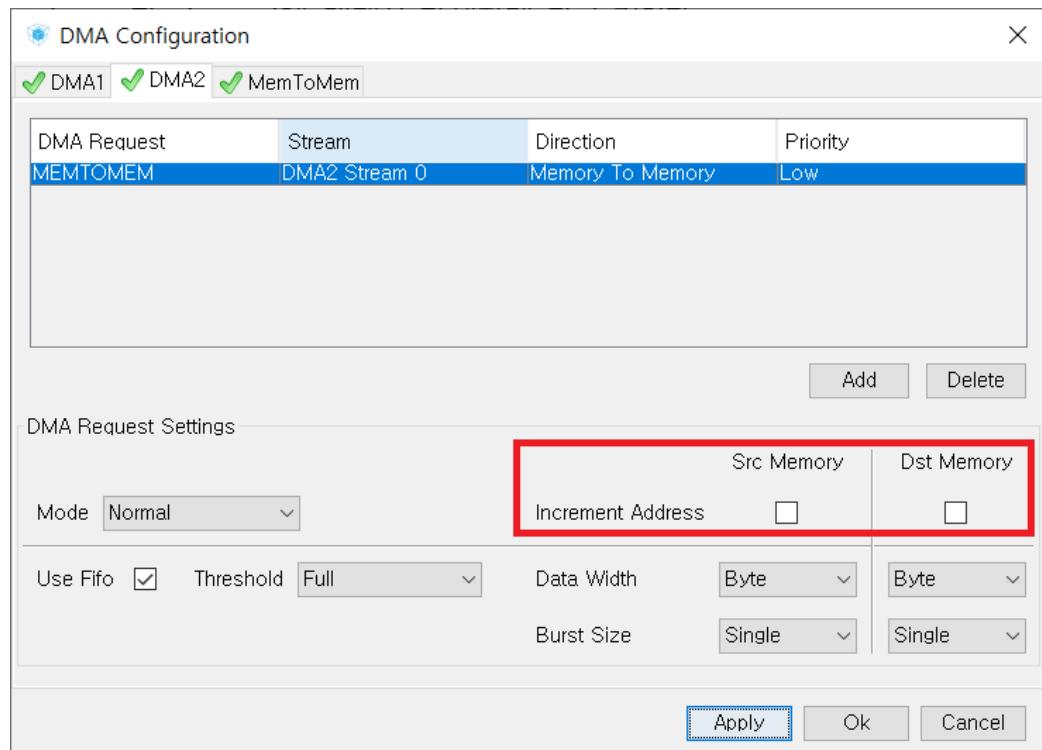
A. 이제 코드를 작성해보려고 한다



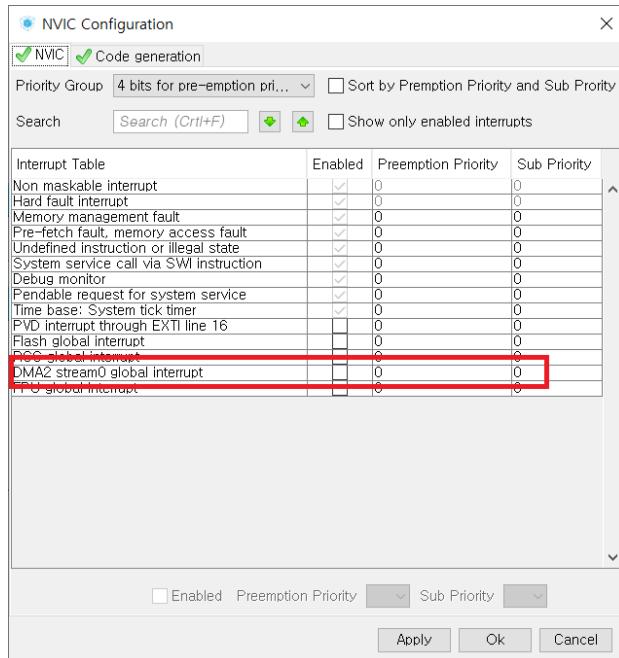
- B. 그래서 AHB1(GPIO 포트)은 DMA2에 연결되어 있기 때문에 아무 stream으로 지정하도록 한다. 뒤에서 stream과 channel에 대해서 자세하게 다룰 것이다



그리고 자동으로 주소를 증가시키는 것이 아닌, 코드로 직접적으로 주소를 증가시킬 것이다



C. 현재는 interrupt가 아닌 polling으로 DMA를 실행할 것이다



D. 코드를 생성하면 DMA 변수가 생성된 것을 확인할 수 있다, 그리고 cube에서 지정한 설정들도 확인이 가능하다

```

/* USER CODE END Includes */

/* Private variables -----
DMA_HandleTypeDef hdma_memtomem_dma2_stream0;

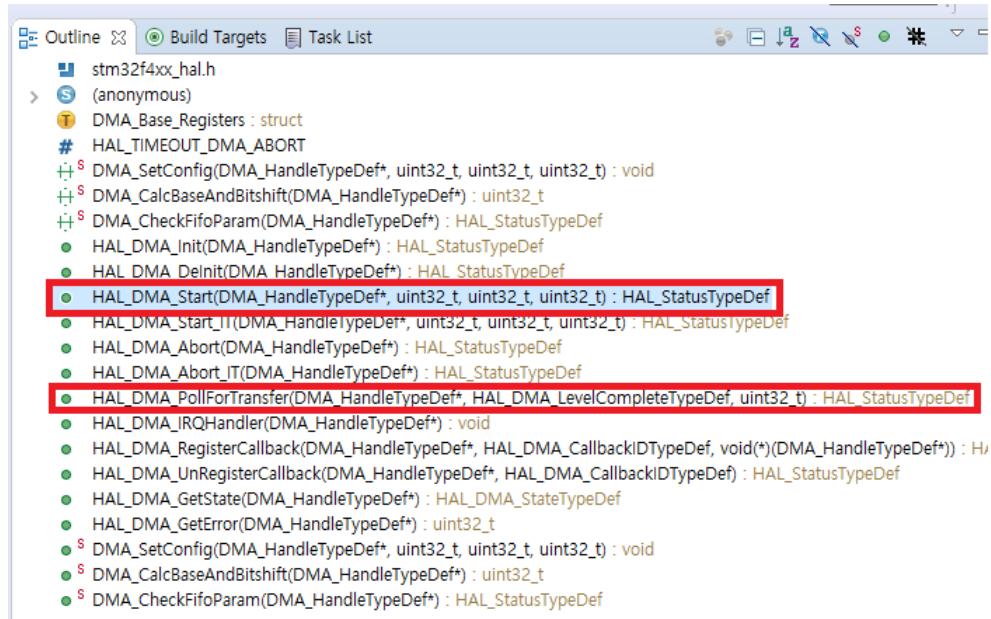
/* USER CODE BEGIN PV */
/* Private variables ----- */

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

    /* Configure DMA request hdma_memtomem_dma2_stream0 on DMA2_Stream0 */
    hdma_memtomem_dma2_stream0.Instance = DMA2_Stream0;
    hdma_memtomem_dma2_stream0.Init.Channel = DMA_CHANNEL_0;
    hdma_memtomem_dma2_stream0.Init.Direction = DMA_MEMORY_TO_MEMORY;
    hdma_memtomem_dma2_stream0.InitPeriphInc = DMA_PINC_DISABLE;
    hdma_memtomem_dma2_stream0.InitMemInc = DMA_MINC_DISABLE;
    hdma_memtomem_dma2_stream0.InitPeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_memtomem_dma2_stream0.InitMemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_memtomem_dma2_stream0.Init.Mode = DMA_NORMAL;
    hdma_memtomem_dma2_stream0.Init.Priority = DMA_PRIORITY_LOW;
    hdma_memtomem_dma2_stream0.Init.FIFOMode = DMA_FIFOMODE_ENABLE;
    hdma_memtomem_dma2_stream0.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_FULL;
    hdma_memtomem_dma2_stream0.Init.MemBurst = DMA_MBURST_SINGLE;
    hdma_memtomem_dma2_stream0.Init.PeriphBurst = DMA_PBURST_SINGLE;
    if (HAL_DMA_Init(&hdma_memtomem_dma2_stream0) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }
}

```

- E. 그리고 아웃라인 확인을 통해서 DMA API들을 살펴보도록 한다. Polling에서 사용할 함수들은 2가지가 있다. 하나는 DMA 시작, 나머지 하나는 polling 시스템에서 DMA가 도착지에 도달할 때까지 기다리는 것이다



- F. 그리고 다음과 같이 코드를 작성할 수 있다. 429보드에 있는 PG13, 14에 대한 LED를 제어하려고 한다

```

/* USER CODE BEGIN 0 */
uint16_t led_data[2] = {0xffff, 0x0000};
/* USER CODE END 0 */

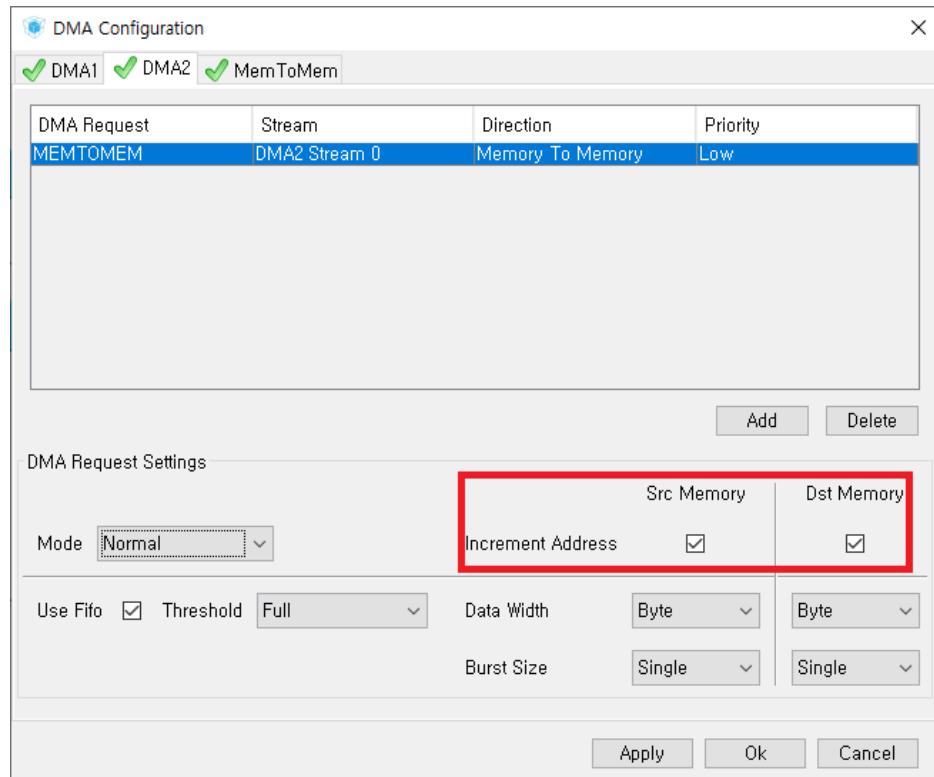
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_DMA_Start(&hdma_memtomem_dma2_stream0, (uint32_t)&led_data[0], (uint32_t)&GPIOG->ODR, 2);
    HAL_DMA_PollForTransfer(&hdma_memtomem_dma2_stream0, HAL_DMA_FULL_TRANSFER, HAL_MAX_DELAY);
    HAL_Delay(500);

    HAL_DMA_Start(&hdma_memtomem_dma2_stream0, (uint32_t)&led_data[1], (uint32_t)&GPIOG->ODR, 2);
    HAL_DMA_PollForTransfer(&hdma_memtomem_dma2_stream0, HAL_DMA_FULL_TRANSFER, HAL_MAX_DELAY);
    HAL_Delay(500);
}
/* USER CODE END 3 */

```

- G. 실수했던 점은 DMA는 한 바이트씩 전달을 한다. 따라서 DMA_Start 함수 마지막 매개변수인 length을 2라고 명시해야만 13, 14 포트에 데이터가 써질 수 있다. 그리고 DMA가 작성될 때, 데이터가 써질 때 Dst, Src 주소가 자동으로 증가해야만 정상적으로 써질 수 있었다. 따라서 위에서 놓친 자동 increment 설정을 해야만 한다



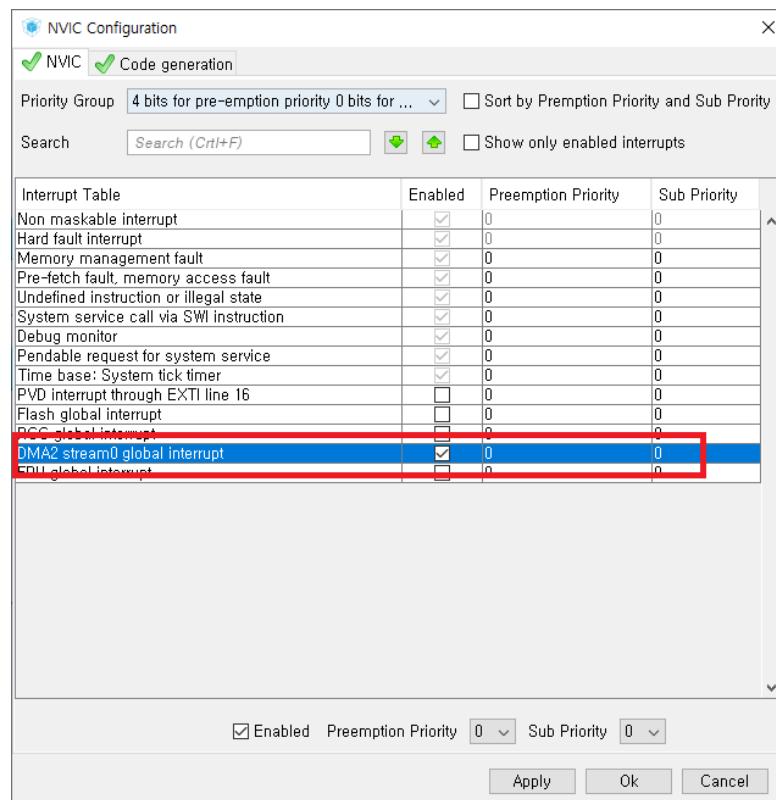
- H. 그러면 LED가 500ms 주기로 toggle 되는 것을 확인할 수 있다

[동영상]

2.2 LED Toggle with Interrupt

- 이전에 polling으로 DMA를 활용해봤다. 조금은 다르지만 거의 비슷한 프로그램 방식으로 인터럽트 방식을 활용하려고 한다

A. DMA2에 대한 NVIC을 활성화한다



B. 그리고 이번에는 IT API인 DMA_Start_IT를 실행해보도록 할 것이다

```
/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
/* USER CODE BEGIN 2 */

HAL_DMA_Start_IT(&hdma_memtomem_dma2_stream0, (uint32_t)&led_data[idx], (uint32_t)&GPIOG->ODR, 2);
/* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
}
```

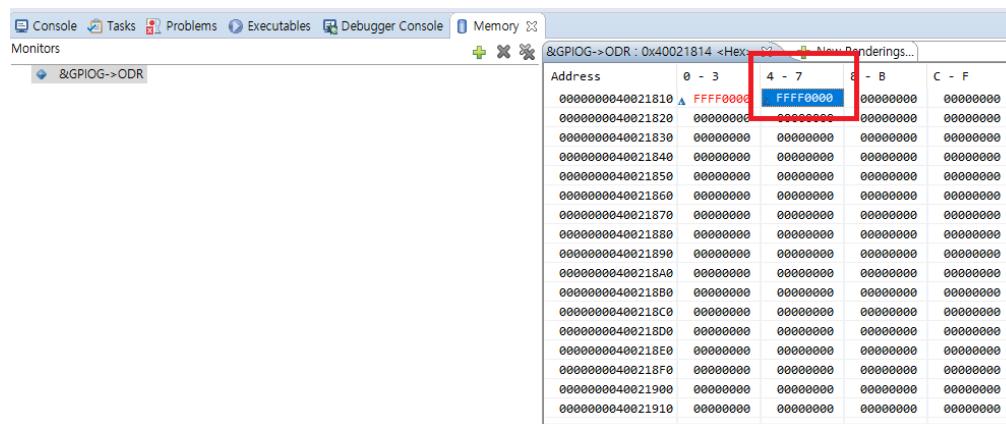
C. 디버깅을 실행하면 Handler로 진입하는 것을 알 수 있다

```

197 */
198 void DMA2_Stream0_IRQHandler(void)
199 {
200     /* USER CODE BEGIN DMA2_Stream0_IRQHandler 0 */
201
202     /* USER CODE END DMA2_Stream0_IRQHandler 0 */
203     HAL_DMA_IRQHandler(&hdma_memtomem_dma2_stream0);
204     /* USER CODE BEGIN DMA2_Stream0_IRQHandler 1 */
205
206     /* USER CODE END DMA2_Stream0_IRQHandler 1 */
207 }
208
209 /* USER CODE BEGIN 1 */
210
211 /* USER CODE END 1 */

```

그리고 메모리에 값이 제대로 쓰여지는 것도 확인이 가능하다



D. 하지만 Handler 안에서 정확한 callback이 정해져 있지 않아서 callback 함수가 실행이 되지 않는다. 아래 빨간 박스에 진입이 되지 않는다

```

912     if((hdma->Instance->CR & DMA_SxCR_CIRC) == RESET)
913     {
914         /* Disable the transfer complete interrupt */
915         hdma->Instance->CR  &= ~(DMA_IT_TC);
916
917         /* Process Unlocked */
918         __HAL_UNLOCK(hdma);
919
920         /* Change the DMA state */
921         hdma->State = HAL_DMA_STATE_READY;
922     }
923
924     if(hdma->XferCpltCallback != NULL)
925     {
926         /* Transfer complete callback */
927         hdma->XferCpltCallback(hdma);
928     }
929 }
930 }
931
932 /* manage error case */
933 if(hdma->ErrorCode != HAL_DMA_ERROR_NONE)
934 {
935     if((hdma->ErrorCode & HAL_DMA_ERROR_TE) != RESET)
936     {
937         hdma->State = HAL_DMA_STATE_ABORT;
938
939         /* Disable the stream */
940         __HAL_DMA_DISABLE(hdma);
941
942         do
943

```

E. 따라서 다음 API를 사용해서 콜백함수를 등록해야 한다

```

/**
 * @brief Register callbacks
 * @param hdma pointer to a DMA_HandleTypeDef structure that contains
 *              the configuration information for the specified DMA Stream.
 * @param CallbackID User Callback identifier
 * @param pCallback a DMA_HandleTypeDef structure as parameter.
 *                  pointer to private callback function which has pointer to
 *                  a DMA_HandleTypeDef structure as parameter.
 * @retval HAL status
 */
HAL_StatusTypeDef HAL_DMA_RegisterCallback(DMA_HandleTypeDef *_hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void (* pCallback)(DMA_HandleTypeDef *_hdma))
{
    HAL_StatusTypeDef status = HAL_OK;

    /* Process locked */
    _HAL_LOCK(hdma);

    if(HAL_DMA_STATE_READY == hdma->State)
    {
        switch (CallbackID)
        {
            case HAL_DMA_XFER_CPLT_CB_ID:
                hdma->xferCpltCallback = pCallback;
                break;

            case HAL_DMA_XFER_HALFCPLT_CB_ID:
                hdma->xferHalfCpltCallback = pCallback;
                break;
        }
    }
}

HAL_DMA_RegisterCallback(&hdma_memtomem_dma2_stream0,
                        HAL_DMA_XFER_CPLT_CB_ID, dma_full_callback);

```

다음과 같이 callback 함수를 만들어서 DMA에 등록을 해주도록 한다. 다음 callback 함수는 모든 전송이 완료되었을 때 이뤄진다

```

04
65 /* USER CODE BEGIN 0 */
66 uint8_t idx;
67 volatile uint8_t dma_flag;
68 uint16_t led_data[2] = {0xffff, 0x0000};
69
70 void dma_full_callback(DMA_HandleTypeDef *_hdma)
71 {
72     dma_flag = 1;
73 }
/* USER CODE END 0 */

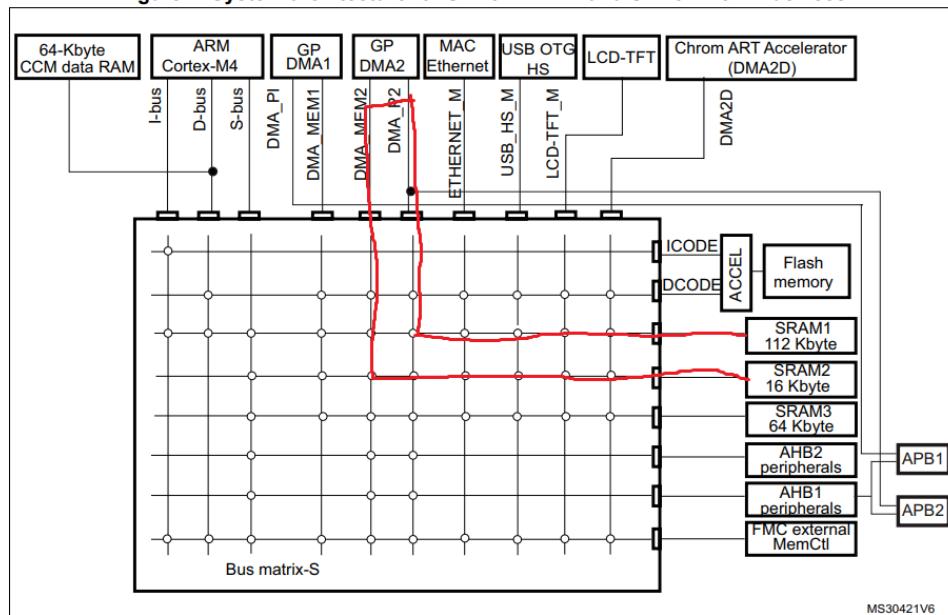
116 /* USER CODE BEGIN WHILE */
117 while (1)
118 {
119
120 /* USER CODE END WHILE */
121
122 /* USER CODE BEGIN 3 */
123     if(dma_flag) {
124         HAL_Delay(500);
125         idx = (idx + 1) % 2;
126         HAL_DMA_Start_IT(&hdma_memtomem_dma2_stream0, (uint32_t)&led_data[idx], (uint32_t)&GPIOG->ODR, 2);
127         dma_flag = 0;
128     }
129 }
/* USER CODE END 3 */
130
131 /* USER CODE END 3 */
132
133 }
134

```

2.3 DMA를 이용한 SRAM 간 데이터 공유

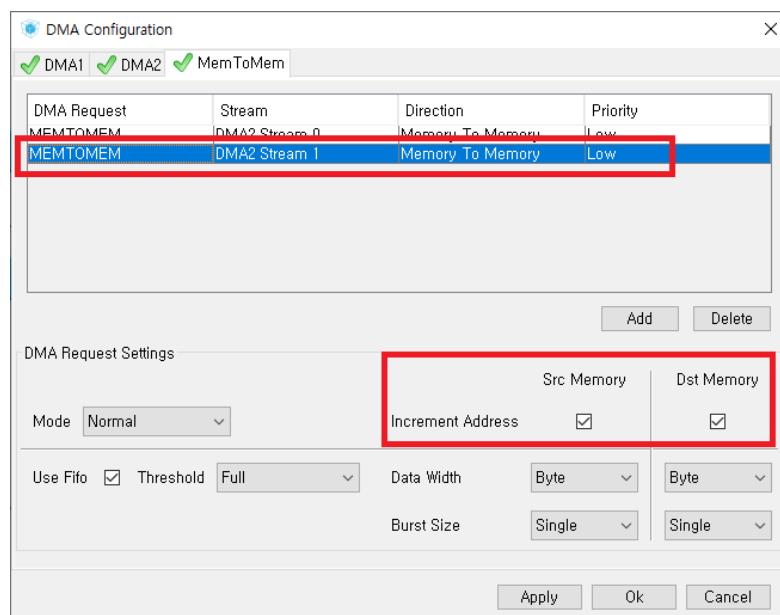
- DMA를 이용해서 SRAM1과 SRAM2 사이에 데이터를 공유하려고 한다. 먼저 버스 매트릭스를 통해서 어떤 루트로 데이터를 전달할 수 있을지 보려고 한다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



DMA1은 Peripheral간 메모리 공유만 가능한 것을 볼 수 있고, DMA2만 빨간색 경로로 SRAM간 데이터 공유가 가능하다. 따라서 DMA2를 사용하려고 한다. 기존의 스트림을 사용해도 되고, 하나의 스트림을 새로 생성해도 좋다

A. 새로 스트림을 만들도록 한다



B. 코드를 생성한 이후에 SRAM2의 주소를 확인하도록 한다

**Table 4. Memory mapping vs. Boot mode/physical remap
in STM32F42xxx and STM32F43xxx**

Addresses	Boot/Remap in main Flash memory	Boot/Remap in embedded SRAM	Boot/Remap in System memory	Remap in FMC
0x2002 0000 - 0x2002 FFFF	SRAM3 (64 KB)	SRAM3 (64 KB)	SRAM3 (64 KB)	SRAM3 (64 KB)
0x2001 C000 - 0x2001 FFFF	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)
0x2000 0000 - 0x2001 BFFF	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)
0x1FFF 0000 - 0x1FFF 77FF	System memory	System memory	System memory	System memory
0x0810 0000 - 0x0FFF FFFF	Reserved	Reserved	Reserved	Reserved
0x0800 0000 - 0x081F FFFF	Flash memory	Flash memory	Flash memory	Flash memory

C. 임의로 SRAM2의 주소를 지정한다

```
/* USER CODE BEGIN 0 */

uint16_t dma_data[2] = {0x5678, 0x0000};

/* USER CODE END 0 */
```

D. 데이터와 DMA API로 호출하도록 한다. 이상하게 DMA를 한 후에 코드가 없으면 데이터는 써지는데 while 무한루프로 돌아가지 않는 것을 확인할 수 있다. 그래서 임의의 코드를 작성한다

```
HAL_DMA_Start(&hdma_memtomem_dma2_stream1, (uint32_t)&dma_data[0], (uint32_t)SRAM2_BASE_ADDR, 2);

/* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
    HAL_Delay(500);

    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);
    HAL_Delay(500);
}
/* USER CODE END 3 */
```

E. 데이터가 써진 것을 확인할 수 있다. 리틀 엔디안 방식을 따르기 때문에 주소가 낮은 곳이 오른쪽 부분이다

```

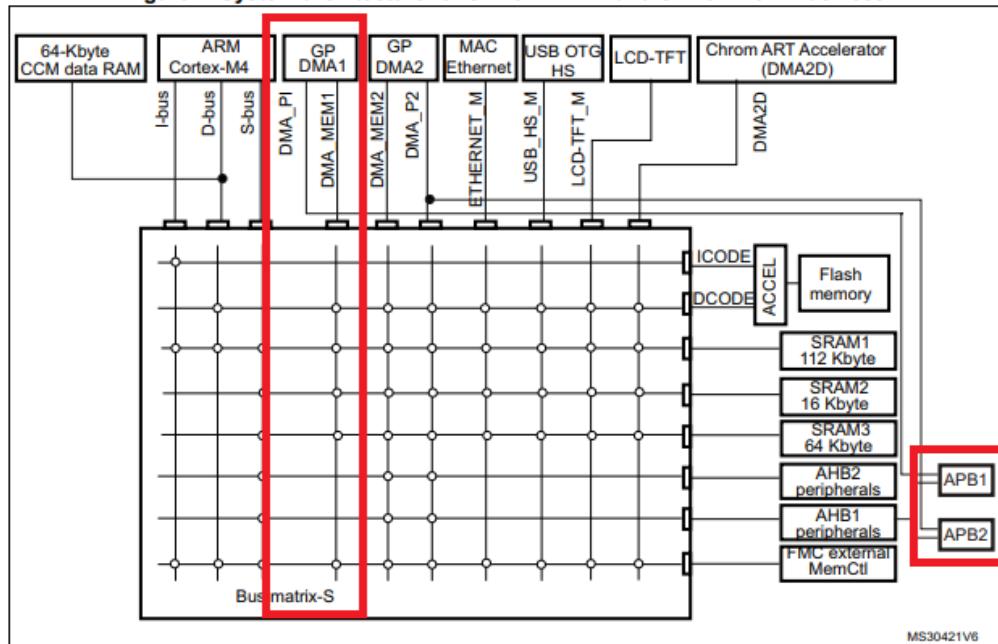
main.c   @ sm2l24x.h  @ sm2l24x.h  @ status_sm2...  @ STM32F429I...  @ sm2l24x.h  @ sm2l24x.h
97     /* USER CODE END Systick */
98
99     /* Initialize all configured peripherals */
100
101     MX_GPIO_Init();
102
103     /* USER CODE BEGIN 3 */
104
105     /* User can call背部callback(DMA_HandleTypeDef *hDMA, HAL_DMA_CallbackIDTypeDef CallbackID, void (*pCallback)(DMA_HandleTypeDef *hDMA));
106
107     /* USER CODE END 3 */
108
109     /* Infinite loop */
110     while (1)
111     {
112
113     /* USER CODE END WHILE */
114
115     /* USER CODE BEGIN 4 */
116
117     /* HAL_Delay(100); */
118
119     HAL_Delay(100);
120
121     /* USER CODE END 4 */
122
123     /* USER CODE BEGIN 5 */
124
125     /* HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); */
126
127     HAL_Delay(100);
128
129     /* USER CODE END 5 */
130
131     /* USER CODE BEGIN 6 */
132
133     /* USER CODE END 6 */
134
135     /* USER CODE BEGIN 7 */
136
137     /* USER CODE END 7 */
138
139     /* USER CODE BEGIN 8 */
140
141     /* USER CODE END 8 */
142
143     /* USER CODE BEGIN 9 */
144
145     /* USER CODE END 9 */
146
147     /* USER CODE BEGIN 10 */
148
149     /* USER CODE END 10 */
150
151     /* USER CODE BEGIN 11 */
152
153     /* USER CODE END 11 */
154
155     /* USER CODE BEGIN 12 */
156
157     /* USER CODE END 12 */
158
159     /* USER CODE BEGIN 13 */
160
161     /* USER CODE END 13 */
162
163     /* USER CODE BEGIN 14 */
164
165     /* USER CODE END 14 */
166
167     /* USER CODE BEGIN 15 */
168
169     /* USER CODE END 15 */
170
171     /* USER CODE BEGIN 16 */
172
173     /* USER CODE END 16 */
174
175     /* USER CODE BEGIN 17 */
176
177     /* USER CODE END 17 */
178
179     /* USER CODE BEGIN 18 */
180
181     /* USER CODE END 18 */
182
183     /* USER CODE BEGIN 19 */
184
185     /* USER CODE END 19 */
186
187     /* USER CODE BEGIN 20 */
188
189     /* USER CODE END 20 */
190
191     /* USER CODE BEGIN 21 */
192
193     /* USER CODE END 21 */
194
195     /* USER CODE BEGIN 22 */
196
197     /* USER CODE END 22 */
198
199     /* USER CODE BEGIN 23 */
200
201     /* USER CODE END 23 */
202
203     /* USER CODE BEGIN 24 */
204
205     /* USER CODE END 24 */
206
207     /* USER CODE BEGIN 25 */
208
209     /* USER CODE END 25 */
210
211     /* USER CODE BEGIN 26 */
212
213     /* USER CODE END 26 */
214
215     /* USER CODE BEGIN 27 */
216
217     /* USER CODE END 27 */
218
219     /* USER CODE BEGIN 28 */
220
221     /* USER CODE END 28 */
222
223     /* USER CODE BEGIN 29 */
224
225     /* USER CODE END 29 */
226
227     /* USER CODE BEGIN 30 */
228
229     /* USER CODE END 30 */
230
231     /* USER CODE BEGIN 31 */
232
233     /* USER CODE END 31 */
234
235     /* USER CODE BEGIN 32 */
236
237     /* USER CODE END 32 */
238
239     /* USER CODE BEGIN 33 */
240
241     /* USER CODE END 33 */
242
243     /* USER CODE BEGIN 34 */
244
245     /* USER CODE END 34 */
246
247     /* USER CODE BEGIN 35 */
248
249     /* USER CODE END 35 */
250
251     /* USER CODE BEGIN 36 */
252
253     /* USER CODE END 36 */
254
255     /* USER CODE BEGIN 37 */
256
257     /* USER CODE END 37 */
258
259     /* USER CODE BEGIN 38 */
260
261     /* USER CODE END 38 */
262
263     /* USER CODE BEGIN 39 */
264
265     /* USER CODE END 39 */
266
267     /* USER CODE BEGIN 40 */
268
269     /* USER CODE END 40 */
270
271     /* USER CODE BEGIN 41 */
272
273     /* USER CODE END 41 */
274
275     /* USER CODE BEGIN 42 */
276
277     /* USER CODE END 42 */
278
279     /* USER CODE BEGIN 43 */
280
281     /* USER CODE END 43 */
282
283     /* USER CODE BEGIN 44 */
284
285     /* USER CODE END 44 */
286
287     /* USER CODE BEGIN 45 */
288
289     /* USER CODE END 45 */
290
291     /* USER CODE BEGIN 46 */
292
293     /* USER CODE END 46 */
294
295     /* USER CODE BEGIN 47 */
296
297     /* USER CODE END 47 */
298
299     /* USER CODE BEGIN 50 */
300
301     /* USER CODE END 50 */
302
303     /* USER CODE BEGIN 51 */
304
305     /* USER CODE END 51 */
306
307     /* USER CODE BEGIN 52 */
308
309     /* USER CODE END 52 */
310
311     /* USER CODE BEGIN 53 */
312
313     /* USER CODE END 53 */
314
315     /* USER CODE BEGIN 54 */
316
317     /* USER CODE END 54 */
318
319     /* USER CODE BEGIN 55 */
320
321     /* USER CODE END 55 */
322
323     /* USER CODE BEGIN 56 */
324
325     /* USER CODE END 56 */
326
327     /* USER CODE BEGIN 57 */
328
329     /* USER CODE END 57 */
330
331     /* USER CODE BEGIN 58 */
332
333     /* USER CODE END 58 */
334
335     /* USER CODE BEGIN 59 */
336
337     /* USER CODE END 59 */
338
339     /* USER CODE BEGIN 60 */
340
341     /* USER CODE END 60 */
342
343     /* USER CODE BEGIN 61 */
344
345     /* USER CODE END 61 */
346
347     /* USER CODE BEGIN 62 */
348
349     /* USER CODE END 62 */
350
351     /* USER CODE BEGIN 63 */
352
353     /* USER CODE END 63 */
354
355     /* USER CODE BEGIN 64 */
356
357     /* USER CODE END 64 */
358
359     /* USER CODE BEGIN 65 */
360
361     /* USER CODE END 65 */
362
363     /* USER CODE BEGIN 66 */
364
365     /* USER CODE END 66 */
366
367     /* USER CODE BEGIN 67 */
368
369     /* USER CODE END 67 */
370
371     /* USER CODE BEGIN 68 */
372
373     /* USER CODE END 68 */
374
375     /* USER CODE BEGIN 69 */
376
377     /* USER CODE END 69 */
378
379     /* USER CODE BEGIN 70 */
380
381     /* USER CODE END 70 */
382
383     /* USER CODE BEGIN 71 */
384
385     /* USER CODE END 71 */
386
387     /* USER CODE BEGIN 72 */
388
389     /* USER CODE END 72 */
390
391     /* USER CODE BEGIN 73 */
392
393     /* USER CODE END 73 */
394
395     /* USER CODE BEGIN 74 */
396
397     /* USER CODE END 74 */
398
399     /* USER CODE BEGIN 75 */
400
401     /* USER CODE END 75 */
402
403     /* USER CODE BEGIN 76 */
404
405     /* USER CODE END 76 */
406
407     /* USER CODE BEGIN 77 */
408
409     /* USER CODE END 77 */
410
411     /* USER CODE BEGIN 78 */
412
413     /* USER CODE END 78 */
414
415     /* USER CODE BEGIN 79 */
416
417     /* USER CODE END 79 */
418
419     /* USER CODE BEGIN 80 */
420
421     /* USER CODE END 80 */
422
423     /* USER CODE BEGIN 81 */
424
425     /* USER CODE END 81 */
426
427     /* USER CODE BEGIN 82 */
428
429     /* USER CODE END 82 */
430
431     /* USER CODE BEGIN 83 */
432
433     /* USER CODE END 83 */
434
435     /* USER CODE BEGIN 84 */
436
437     /* USER CODE END 84 */
438
439     /* USER CODE BEGIN 85 */
440
441     /* USER CODE END 85 */
442
443     /* USER CODE BEGIN 86 */
444
445     /* USER CODE END 86 */
446
447     /* USER CODE BEGIN 87 */
448
449     /* USER CODE END 87 */
450
451     /* USER CODE BEGIN 88 */
452
453     /* USER CODE END 88 */
454
455     /* USER CODE BEGIN 89 */
456
457     /* USER CODE END 89 */
458
459     /* USER CODE BEGIN 90 */
460
461     /* USER CODE END 90 */
462
463     /* USER CODE BEGIN 91 */
464
465     /* USER CODE END 91 */
466
467     /* USER CODE BEGIN 92 */
468
469     /* USER CODE END 92 */
470
471     /* USER CODE BEGIN 93 */
472
473     /* USER CODE END 93 */
474
475     /* USER CODE BEGIN 94 */
476
477     /* USER CODE END 94 */
478
479     /* USER CODE BEGIN 95 */
480
481     /* USER CODE END 95 */
482
483     /* USER CODE BEGIN 96 */
484
485     /* USER CODE END 96 */
486
487     /* USER CODE BEGIN 97 */
488
489     /* USER CODE END 97 */
490
491     /* USER CODE BEGIN 98 */
492
493     /* USER CODE END 98 */
494
495     /* USER CODE BEGIN 99 */
496
497     /* USER CODE END 99 */
498
499     /* USER CODE BEGIN 100 */
500
501     /* USER CODE END 100 */
502
503     /* USER CODE BEGIN 101 */
504
505     /* USER CODE END 101 */
506
507     /* USER CODE BEGIN 102 */
508
509     /* USER CODE END 102 */
510
511     /* USER CODE BEGIN 103 */
512
513     /* USER CODE END 103 */
514
515     /* USER CODE BEGIN 104 */
516
517     /* USER CODE END 104 */
518
519     /* USER CODE BEGIN 105 */
520
521     /* USER CODE END 105 */
522
523     /* USER CODE BEGIN 106 */
524
525     /* USER CODE END 106 */
526
527     /* USER CODE BEGIN 107 */
528
529     /* USER CODE END 107 */
530
531     /* USER CODE BEGIN 108 */
532
533     /* USER CODE END 108 */
534
535     /* USER CODE BEGIN 109 */
536
537     /* USER CODE END 109 */
538
539     /* USER CODE BEGIN 110 */
540
541     /* USER CODE END 110 */
542
543     /* USER CODE BEGIN 111 */
544
545     /* USER CODE END 111 */
546
547     /* USER CODE BEGIN 112 */
548
549     /* USER CODE END 112 */
550
551     /* USER CODE BEGIN 113 */
552
553     /* USER CODE END 113 */
554
555     /* USER CODE BEGIN 114 */
556
557     /* USER CODE END 114 */
558
559     /* USER CODE BEGIN 115 */
560
561     /* USER CODE END 115 */
562
563     /* USER CODE BEGIN 116 */
564
565     /* USER CODE END 116 */
566
567     /* USER CODE BEGIN 117 */
568
569     /* USER CODE END 117 */
570
571     /* USER CODE BEGIN 118 */
572
573     /* USER CODE END 118 */
574
575     /* USER CODE BEGIN 119 */
576
577     /* USER CODE END 119 */
578
579     /* USER CODE BEGIN 120 */
580
581     /* USER CODE END 120 */
582
583     /* USER CODE BEGIN 121 */
584
585     /* USER CODE END 121 */
586
587     /* USER CODE BEGIN 122 */
588
589     /* USER CODE END 122 */
590
591     /* USER CODE BEGIN 123 */
592
593     /* USER CODE END 123 */
594
595     /* USER CODE BEGIN 124 */
596
597     /* USER CODE END 124 */
598
599     /* USER CODE BEGIN 125 */
599
600     /* USER CODE END 125 */
601
602     /* USER CODE BEGIN 126 */
603
604     /* USER CODE END 126 */
605
606     /* USER CODE BEGIN 127 */
607
608     /* USER CODE END 127 */
609
610     /* USER CODE BEGIN 128 */
611
612     /* USER CODE END 128 */
613
614     /* USER CODE BEGIN 129 */
615
616     /* USER CODE END 129 */
617
618     /* USER CODE BEGIN 130 */
619
620     /* USER CODE END 130 */
621
622     /* USER CODE BEGIN 131 */
623
624     /* USER CODE END 131 */
625
626     /* USER CODE BEGIN 132 */
627
628     /* USER CODE END 132 */
629
630     /* USER CODE BEGIN 133 */
631
632     /* USER CODE END 133 */
633
634     /* USER CODE BEGIN 134 */
635
636     /* USER CODE END 134 */
637
638     /* USER CODE BEGIN 135 */
639
640     /* USER CODE END 135 */
641
642     /* USER CODE BEGIN 136 */
643
644     /* USER CODE END 136 */
645
646     /* USER CODE BEGIN 137 */
647
648     /* USER CODE END 137 */
649
650     /* USER CODE BEGIN 138 */
651
652     /* USER CODE END 138 */
653
654     /* USER CODE BEGIN 139 */
655
656     /* USER CODE END 139 */
657
658     /* USER CODE BEGIN 140 */
659
660     /* USER CODE END 140 */
661
662     /* USER CODE BEGIN 141 */
663
664     /* USER CODE END 141 */
665
666     /* USER CODE BEGIN 142 */
667
668     /* USER CODE END 142 */
669
670     /* USER CODE BEGIN 143 */
671
672     /* USER CODE END 143 */
673
674     /* USER CODE BEGIN 144 */
675
676     /* USER CODE END 144 */
677
678     /* USER CODE BEGIN 145 */
679
680     /* USER CODE END 145 */
681
682     /* USER CODE BEGIN 146 */
683
684     /* USER CODE END 146 */
685
686     /* USER CODE BEGIN 147 */
687
688     /* USER CODE END 147 */
689
690     /* USER CODE BEGIN 148 */
691
692     /* USER CODE END 148 */
693
694     /* USER CODE BEGIN 149 */
695
696     /* USER CODE END 149 */
697
698     /* USER CODE BEGIN 150 */
699
700     /* USER CODE END 150 */
701
702     /* USER CODE BEGIN 151 */
703
704     /* USER CODE END 151 */
705
706     /* USER CODE BEGIN 152 */
707
708     /* USER CODE END 152 */
709
710     /* USER CODE BEGIN 153 */
711
712     /* USER CODE END 153 */
713
714     /* USER CODE BEGIN 154 */
715
716     /* USER CODE END 154 */
717
718     /* USER CODE BEGIN 155 */
719
720     /* USER CODE END 155 */
721
722     /* USER CODE BEGIN 156 */
723
724     /* USER CODE END 156 */
725
726     /* USER CODE BEGIN 157 */
727
728     /* USER CODE END 157 */
729
730     /* USER CODE BEGIN 158 */
731
732     /* USER CODE END 158 */
733
734     /* USER CODE BEGIN 159 */
735
736     /* USER CODE END 159 */
737
738     /* USER CODE BEGIN 160 */
739
740     /* USER CODE END 160 */
741
742     /* USER CODE BEGIN 161 */
743
744     /* USER CODE END 161 */
745
746     /* USER CODE BEGIN 162 */
747
748     /* USER CODE END 162 */
749
750     /* USER CODE BEGIN 163 */
751
752     /* USER CODE END 163 */
753
754     /* USER CODE BEGIN 164 */
755
756     /* USER CODE END 164 */
757
758     /* USER CODE BEGIN 165 */
759
760     /* USER CODE END 165 */
761
762     /* USER CODE BEGIN 166 */
763
764     /* USER CODE END 166 */
765
766     /* USER CODE BEGIN 167 */
767
768     /* USER CODE END 167 */
769
770     /* USER CODE BEGIN 168 */
771
772     /* USER CODE END 168 */
773
774     /* USER CODE BEGIN 169 */
775
776     /* USER CODE END 169 */
777
778     /* USER CODE BEGIN 170 */
779
780     /* USER CODE END 170 */
781
782     /* USER CODE BEGIN 171 */
783
784     /* USER CODE END 171 */
785
786     /* USER CODE BEGIN 172 */
787
788     /* USER CODE END 172 */
789
790     /* USER CODE BEGIN 173 */
791
792     /* USER CODE END 173 */
793
794     /* USER CODE BEGIN 174 */
795
796     /* USER CODE END 174 */
797
798     /* USER CODE BEGIN 175 */
799
800     /* USER CODE END 175 */
801
802     /* USER CODE BEGIN 176 */
803
804     /* USER CODE END 176 */
805
806     /* USER CODE BEGIN 177 */
807
808     /* USER CODE END 177 */
809
810     /* USER CODE BEGIN 178 */
811
812     /* USER CODE END 178 */
813
814     /* USER CODE BEGIN 179 */
815
816     /* USER CODE END 179 */
817
818     /* USER CODE BEGIN 180 */
819
820     /* USER CODE END 180 */
821
822     /* USER CODE BEGIN 181 */
823
824     /* USER CODE END 181 */
825
826     /* USER CODE BEGIN 182 */
827
828     /* USER CODE END 182 */
829
830     /* USER CODE BEGIN 183 */
831
832     /* USER CODE END 183 */
833
834     /* USER CODE BEGIN 184 */
835
836     /* USER CODE END 184 */
837
838     /* USER CODE BEGIN 185 */
839
840     /* USER CODE END 185 */
841
842     /* USER CODE BEGIN 186 */
843
844     /* USER CODE END 186 */
845
846     /* USER CODE BEGIN 187 */
847
848     /* USER CODE END 187 */
849
850     /* USER CODE BEGIN 188 */
851
852     /* USER CODE END 188 */
853
854     /* USER CODE BEGIN 189 */
855
856     /* USER CODE END 189 */
857
858     /* USER CODE BEGIN 190 */
859
860     /* USER CODE END 190 */
861
862     /* USER CODE BEGIN 191 */
863
864     /* USER CODE END 191 */
865
866     /* USER CODE BEGIN 192 */
867
868     /* USER CODE END 192 */
869
870     /* USER CODE BEGIN 193 */
871
872     /* USER CODE END 193 */
873
874     /* USER CODE BEGIN 194 */
875
876     /* USER CODE END 194 */
877
878     /* USER CODE BEGIN 195 */
879
880     /* USER CODE END 195 */
881
882     /* USER CODE BEGIN 196 */
883
884     /* USER CODE END 196 */
885
886     /* USER CODE BEGIN 197 */
887
888     /* USER CODE END 197 */
889
890     /* USER CODE BEGIN 198 */
891
892     /* USER CODE END 198 */
893
894     /* USER CODE BEGIN 199 */
895
896     /* USER CODE END 199 */
897
898     /* USER CODE BEGIN 200 */
899
900     /* USER CODE END 200 */
901
902     /* USER CODE BEGIN 201 */
903
904     /* USER CODE END 201 */
905
906     /* USER CODE BEGIN 202 */
907
908     /* USER CODE END 202 */
909
910     /* USER CODE BEGIN 203 */
911
912     /* USER CODE END 203 */
913
914     /* USER CODE BEGIN 204 */
915
916     /* USER CODE END 204 */
917
918     /* USER CODE BEGIN 205 */
919
920     /* USER CODE END 205 */
921
922     /* USER CODE BEGIN 206 */
923
924     /* USER CODE END 206 */
925
926     /* USER CODE BEGIN 207 */
927
928     /* USER CODE END 207 */
929
930     /* USER CODE BEGIN 208 */
931
932     /* USER CODE END 208 */
933
934     /* USER CODE BEGIN 209 */
935
936     /* USER CODE END 209 */
937
938     /* USER CODE BEGIN 210 */
939
940     /* USER CODE END 210 */
941
942     /* USER CODE BEGIN 211 */
943
944     /* USER CODE END 211 */
945
946     /* USER CODE BEGIN 212 */
947
948     /* USER CODE END 212 */
949
950     /* USER CODE BEGIN 213 */
951
952     /* USER CODE END 213 */
953
954     /* USER CODE BEGIN 214 */
955
956     /* USER CODE END 214 */
957
958     /* USER CODE BEGIN 215 */
959
960     /* USER CODE END 215 */
961
962     /* USER CODE BEGIN 216 */
963
964     /* USER CODE END 216 */
965
966     /* USER CODE BEGIN 217 */
967
968     /* USER CODE END 217 */
969
970     /* USER CODE BEGIN 218 */
971
972     /* USER CODE END 218 */
973
974     /* USER CODE BEGIN 219 */
975
976     /* USER CODE END 219 */
977
978     /* USER CODE BEGIN 220 */
979
980     /* USER CODE END 220 */
981
982     /* USER CODE BEGIN 221 */
983
984     /* USER CODE END 221 */
985
986     /* USER CODE BEGIN 222 */
987
988     /* USER CODE END 222 */
989
990     /* USER CODE BEGIN 223 */
991
992     /* USER CODE END 223 */
993
994     /* USER CODE BEGIN 224 */
995
996     /* USER CODE END 224 */
997
998     /* USER CODE BEGIN 225 */
999
1000    /* USER CODE END 225 */
1001
1002    /* USER CODE BEGIN 226 */
1003
1004    /* USER CODE END 226 */
1005
1006    /* USER CODE BEGIN 227 */
1007
1008    /* USER CODE END 227 */
1009
1010    /* USER CODE BEGIN 228 */
1011
1012    /* USER CODE END 228 */
1013
1014    /* USER CODE BEGIN 229 */
1015
1016    /* USER CODE END 229 */
1017
1018    /* USER CODE BEGIN 230 */
1019
1020    /* USER CODE END 230 */
1021
1022    /* USER CODE BEGIN 231 */
1023
1024    /* USER CODE END 231 */
1025
1026    /* USER CODE BEGIN 232 */
1027
1028    /* USER CODE END 232 */
1029
1030    /* USER CODE BEGIN 233 */
1031
1032    /* USER CODE END 233 */
1033
1034    /* USER CODE BEGIN 234 */
1035
1036    /* USER CODE END 234 */
1037
1038    /* USER CODE BEGIN 235 */
1039
1040    /* USER CODE END 235 */
1041
1042    /* USER CODE BEGIN 236 */
1043
1044    /* USER CODE END 236 */
1045
1046    /* USER CODE BEGIN 237 */
1047
1048    /* USER CODE END 237 */
1049
1050    /* USER CODE BEGIN 238 */
1051
1052    /* USER CODE END 238 */
1053
1054    /* USER CODE BEGIN 239 */
1055
1056    /* USER CODE END 239 */
1057
1058    /* USER CODE BEGIN 240 */
1059
1060    /* USER CODE END 240 */
1061
1062    /* USER CODE BEGIN 241 */
1063
1064    /* USER CODE END 241 */
1065
1066    /* USER CODE BEGIN 242 */
1067
1068    /* USER CODE END 242 */
1069
1070    /* USER CODE BEGIN 243 */
1071
1072    /* USER CODE END 243 */
1073
1074    /* USER CODE BEGIN 244 */
1075
1076    /* USER CODE END 244 */
1077
1078    /* USER CODE BEGIN 245 */
1079
1080    /* USER CODE END 245 */
1081
1082    /* USER CODE BEGIN 246 */
1083
1084    /* USER CODE END 246 */
1085
1086    /* USER CODE BEGIN 247 */
1087
1088    /* USER CODE END 247 */
1089
1090    /* USER CODE BEGIN 248 */
1091
1092    /* USER CODE END 248 */
1093
1094    /* USER CODE BEGIN 249 */
1095
1096    /* USER CODE END 249 */
1097
1098    /* USER CODE BEGIN 250 */
1099
1100    /* USER CODE END 250 */
1101
1102    /* USER CODE BEGIN 251 */
1103
1104    /* USER CODE END 251 */
1105
1106    /* USER CODE BEGIN 252 */
1107
1108    /* USER CODE END 252 */
1109
1110    /* USER CODE BEGIN 253 */
1111
1112    /* USER CODE END 253 */
1113
1114    /* USER CODE BEGIN 254 */
1115
1116    /* USER CODE END 254 */
1117
1118    /* USER CODE BEGIN 255 */
1119
1120    /* USER CODE END 255 */
1121
1122    /* USER CODE BEGIN 256 */
1123
1124    /* USER CODE END 256 */
1125
1126    /* USER CODE BEGIN 257 */
1127
1128    /* USER CODE END 257 */
1129
1130    /* USER CODE BEGIN 258 */
1131
1132    /* USER CODE END 258 */
1133
1134    /* USER CODE BEGIN 259 */
1135
1136    /* USER CODE END 259 */
1137
1138    /* USER CODE BEGIN 260 */
1139
1140    /* USER CODE END 260 */
1141
1142    /* USER CODE BEGIN 261 */
1143
1144    /* USER CODE END 261 */
1145
1146    /* USER CODE BEGIN 262 */
1147
1148    /* USER CODE END 262 */
1149
1150    /* USER CODE BEGIN 263 */
1151
1152    /* USER CODE END 263 */
1153
1154    /* USER CODE BEGIN 264 */
1155
1156    /* USER CODE END 264 */
1157
1158    /* USER CODE BEGIN 265 */
1159
1160    /* USER CODE END 265 */
1161
1162    /* USER CODE BEGIN 266 */
1163
1164    /* USER CODE END 266 */
1165
1166    /* USER CODE BEGIN 267 */
1167
1168    /* USER CODE END 267 */
1169
1170    /* USER CODE BEGIN 268 */
1171
1172    /* USER CODE END 268 */
1173
1174    /* USER CODE BEGIN 269 */
1175
1176    /* USER CODE END 269 */
1177
1178    /* USER CODE BEGIN 270 */
1179
1180    /* USER CODE END 270 */
1181
1182    /* USER CODE BEGIN 271 */
1183
1184    /* USER CODE END 271 */
1185
1186    /* USER CODE BEGIN 272 */
1187
1188    /* USER CODE END 272 */
1189
1190    /* USER CODE BEGIN 273 */
1191
1192    /* USER CODE END 273 */
1193
1194    /* USER CODE BEGIN 274 */
1195
1196    /* USER CODE END 274 */
1197
1198    /* USER CODE BEGIN 275 */
1199
1200    /* USER CODE END 275 */
1201
1202    /* USER CODE BEGIN 276 */
1203
1204    /* USER CODE END 276 */
1205
1206    /* USER CODE BEGIN 277 */
1207
1208    /* USER CODE END 277 */
1209
1210    /* USER CODE BEGIN 278 */
1211
1212    /* USER CODE END 278 */
1213
1214    /* USER CODE BEGIN 279 */
1215
1216    /* USER CODE END 279 */
1217
1218    /* USER CODE BEGIN 280 */
1219
1220    /* USER CODE END 280 */
1221
1222    /* USER CODE BEGIN 281 */
1223
1224    /* USER CODE END 281 */
1225
1226    /* USER CODE BEGIN 282 */
1227
1228    /* USER CODE END 282 */
1229
1230    /* USER CODE BEGIN 283 */
1231
1232    /* USER CODE END 283 */
1233
1234    /* USER CODE BEGIN 284 */
1235
1236    /* USER CODE END 284 */
1237
1238    /* USER CODE BEGIN 285 */
1239
1240    /* USER CODE END 285 */
1241
1242    /* USER CODE BEGIN 286 */
1243
1244    /* USER CODE END 286 */
1245
1246    /* USER CODE BEGIN 287 */
1247
1248    /* USER CODE END 287 */
1249
1250    /* USER CODE BEGIN 288 */
1251
1252    /* USER CODE END 288 */
1253
1254    /* USER CODE BEGIN 289 */
1255
1256    /* USER CODE END 289 */
1257
1258    /* USER CODE BEGIN 290 */
1259
1260    /* USER CODE END 290 */
1261
1262    /* USER CODE BEGIN 291 */
1263
1264    /* USER CODE END 291 */
1265
1266    /* USER CODE BEGIN 292 */
1267
1268    /* USER CODE END 292 */
1269
1270    /* USER CODE BEGIN 293 */
1271
1272    /* USER CODE END 293 */
1273
1274    /* USER CODE BEGIN 295 */
1275
1276    /* USER CODE END 295 */
1277
1278    /* USER CODE BEGIN 296 */
1279
1280    /* USER CODE END 296 */
1281
1282    /* USER CODE BEGIN 297 */
1283
1284    /* USER CODE END 297 */
1285
1286    /* USER CODE BEGIN 298 */
1287
1288    /* USER CODE END 298 */
1289
1290    /* USER CODE BEGIN 299 */
1291
1292    /* USER CODE END 299 */
1293
1294    /* USER CODE BEGIN 300 */
1295
1296    /* USER CODE END 300 */
1297
1298    /* USER CODE BEGIN 301 */
1299
1300    /* USER CODE END 301 */
1301
1302    /* USER CODE BEGIN 302 */
1303
1304    /* USER CODE END 302 */
1305
1306    /* USER CODE BEGIN 303 */
1307
1308    /* USER CODE END 303 */
1309
1310    /* USER CODE BEGIN 304 */
1311
1312    /* USER CODE END 304 */
1313
1314    /* USER CODE BEGIN 305 */
1315
1316    /* USER CODE END 305 */
1317
1318    /* USER CODE BEGIN 306 */
1319
1320    /* USER CODE END 306 */
1321
1322    /* USER CODE BEGIN 307 */
1323
1324    /* USER CODE END 307 */
1325
1326    /* USER CODE BEGIN 308 */
1327
1328    /* USER CODE END 308 */
1329
1330    /* USER CODE BEGIN 309 */
1331
1332    /* USER CODE END 309 */
1333
1334    /* USER CODE BEGIN 310 */
1335
1336    /* USER CODE END 310 */
1337
1338    /* USER CODE BEGIN 311 */
1339
1340    /* USER CODE END 311 */
1341
1342    /* USER CODE BEGIN 312 */
1343
1344    /* USER CODE END 312 */
1345
1346    /* USER CODE BEGIN 313 */
1347
1348    /* USER CODE END 313 */
1349
1350    /* USER CODE BEGIN 314 */
1351
1352    /* USER CODE END 314 */
1353
1354    /* USER CODE BEGIN 315 */
1355
1356    /* USER CODE END 315 */
1357
1358    /* USER CODE BEGIN 316 */
1359
1360    /* USER CODE END 316 */
1361
1362    /* USER CODE BEGIN 317 */
1363

```

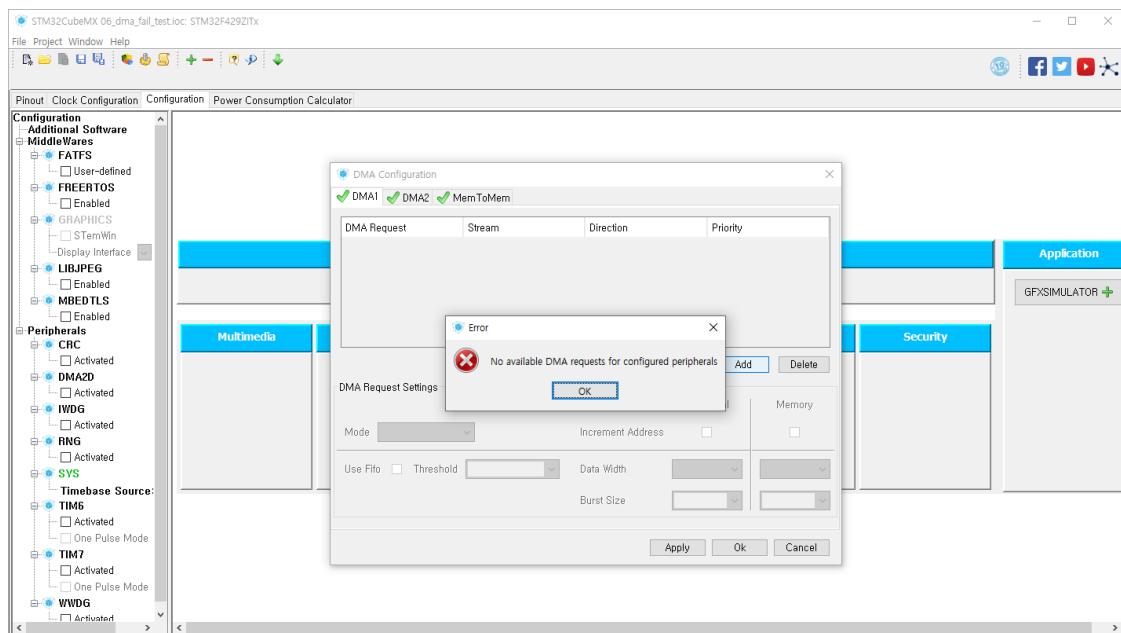
2.4 DMA1으로 했을 때의 오류 발생

- DMA1일 때는 버스 매트릭스로도 보았듯이, 페리페럴과 메모리간 DMA만 지원한다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices

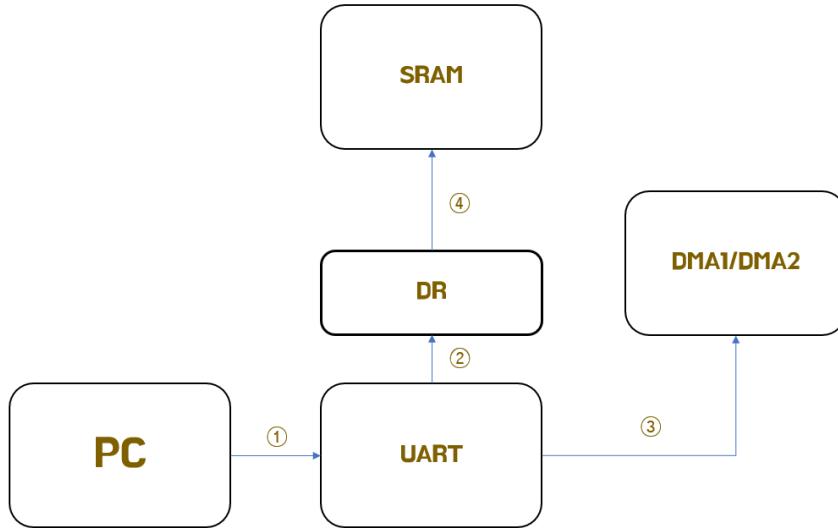


- 만일 cubemx에서 아무리 peripheral 없이 메모리간 DMA를 하기 위해서 Add를 하면 오류가 발생한다

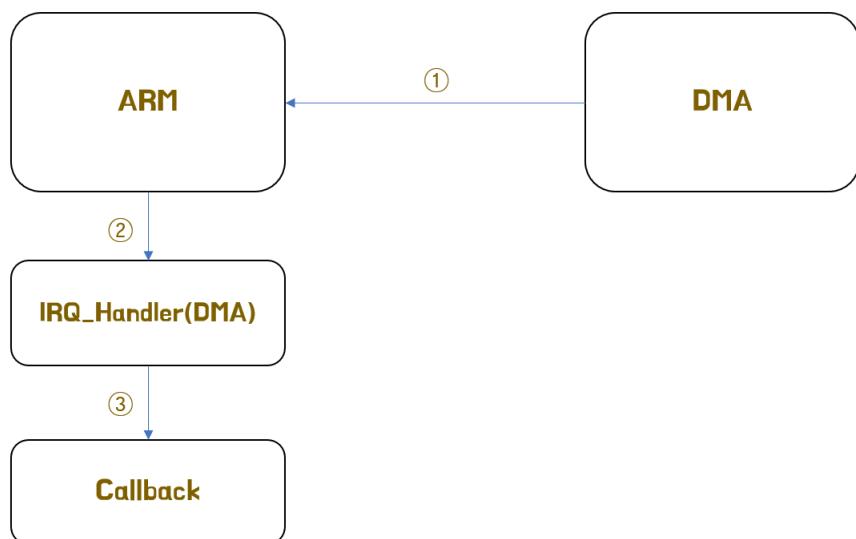


2.5 DMA를 이용한 UART-SRAM 데이터 공유

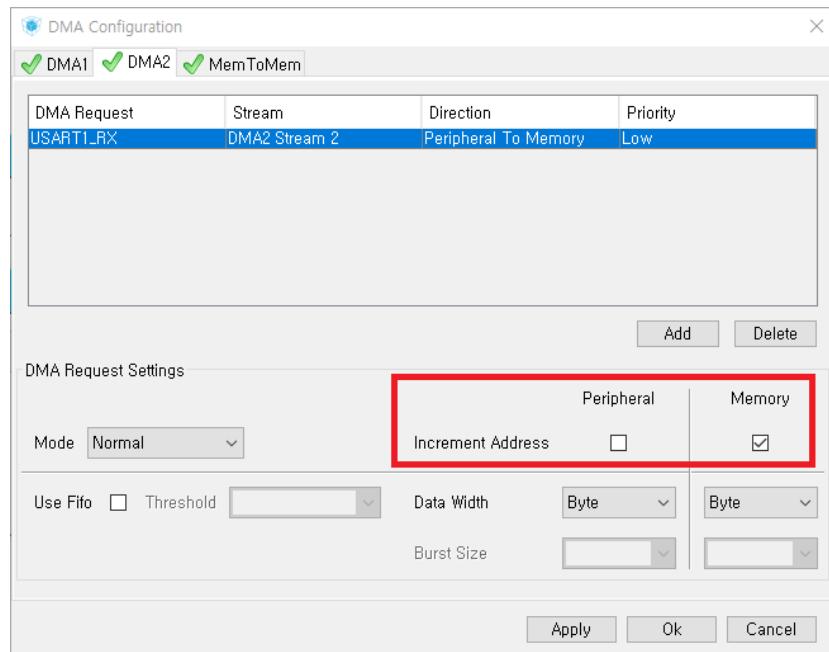
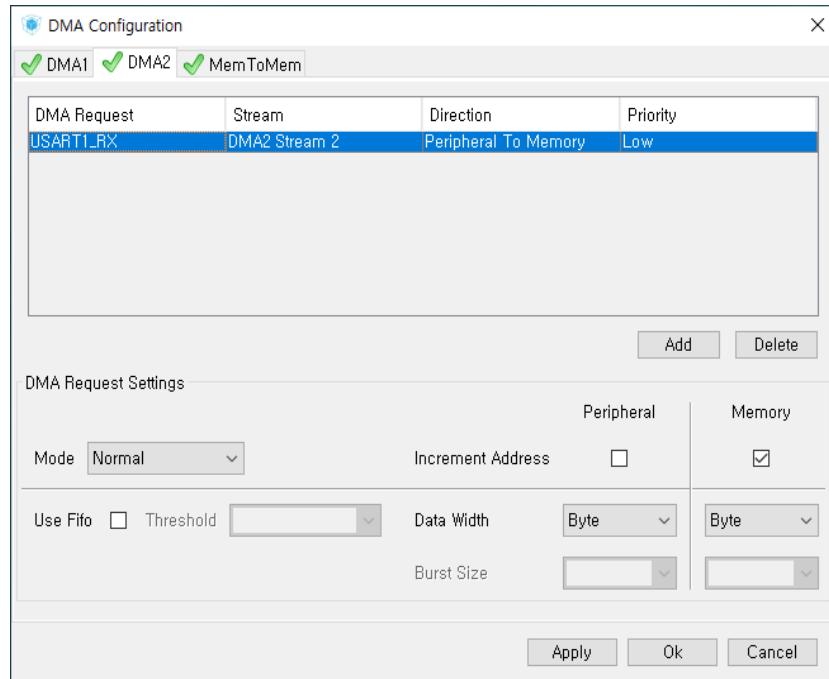
- 지금은 peripheral에서 memory로 데이터를 전달하려고 한다. Polling은 DMA를 쓰는 의미가 없다. 위에서 한 것은 DMA를 사용하기 위해서 실습을 해본 것이고, 지금부터는 인터럽트로 DMA를 사용하려고 한다
- DMA 요청은 다음과 같이 이뤄질 것이다



- 먼저 PC에서 UART로 데이터를 보내면 UART는 이를 데이터 레지스터에 저장한다
- 그리고 처음에 DMA1에 stream과 채널 개방을 요구한다. 열린다면 그 이후에 해당 stream으로 SRAM에 데이터를 저장할 수 있게 된다
- 그리고 config에서 설정한 데이터 바이트가 모두 전달되면 DMA는 ARM에게 handler 요청을하게 되어서, 이때 ARM이 처리를 하게 되는 것이다. 마지막으로 callback 함수를 호출하게 된다

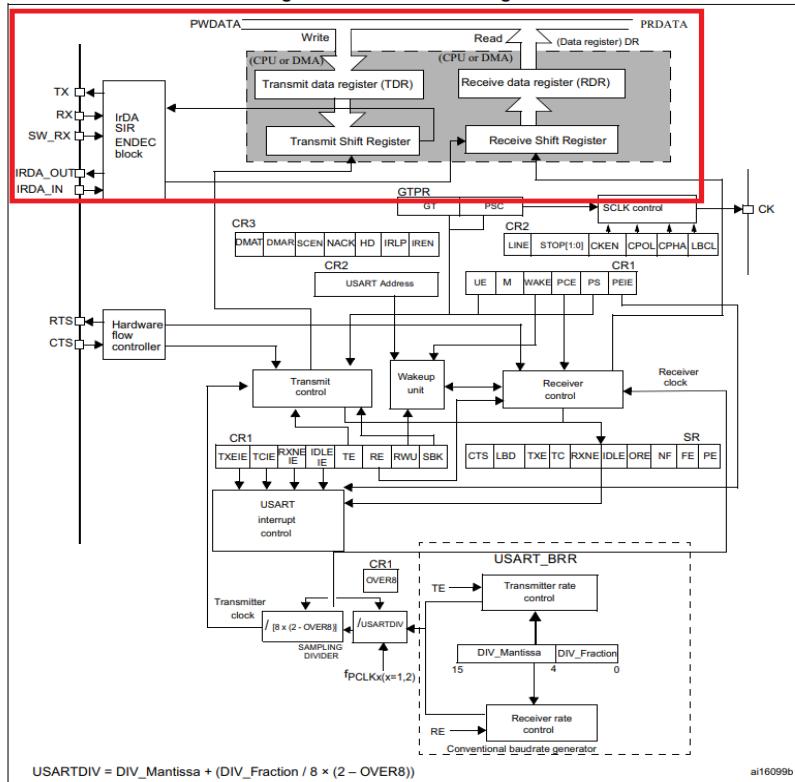


- A. Cubemx를 열어서 USART1을 비동기로 설정하고, DMA 셋팅을 하도록 한다. 페리페럴이기 때문에 채널과 스트림이 정해져 있다



빨간색 영역은 RX DR은 주소가 이미 정해져있는 1바이트 버퍼이고, 저장하려는 SRAM 영역은 연속적으로 증가해야 한다. 따라서 Memory 쪽만 Increment 하기로 한다

Figure 296. USART block diagram



B. 그리고 코드를 작성하도록 한다. UART를 DMA로 받으려고 하는 것이다. 다음 API를 사용하도록 한다

```
HAL_UART_Receive_DMA(&huart1, (uint8_t*)(SRAM_ADDR), 10);
/* USER CODE END 2 */
```

함수 안은 다음과 같이 작성되어 있다. Half, full 2가지 callback을 제공하고 있고, 디버깅 결과 모든 콜백이 호출된다는 것이 확인되었다

```
/* Set the UART DMA transfer complete callback */
huart->hdmarx->XferCpltCallback = UART_DMAReceiveCplt;

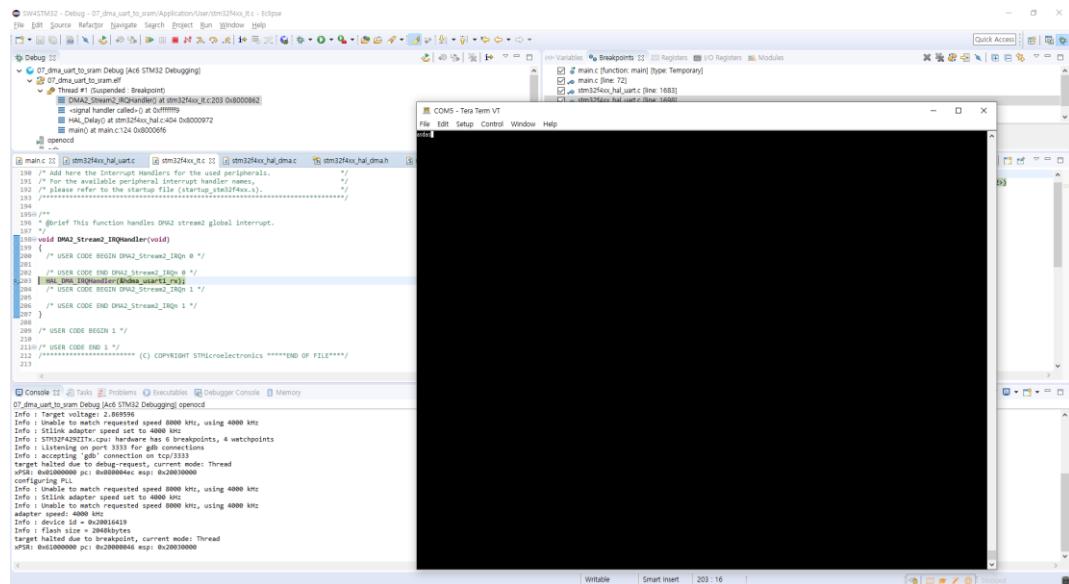
/* Set the UART DMA Half transfer complete callback */
huart->hdmarx->XferHalfCpltCallback = UART_DMARxHalfCplt;

/* Set the DMA error callback */
huart->hdmarx->XferErrorCallback = UART_DMAError;

/* Set the DMA abort callback */
huart->hdmarx->XferAbortCallback = NULL;

/* Enable the DMA Stream */
tmp = (uint32_t*)&pData;
HAL_DMA_Start_IT(huart->hdmarx, (uint32_t)&huart->Instance->DR, *(uint32_t*)tmp, Size);
```

C. Breakpoint를 몇 개 잡고나서 코드를 디버깅 해보도록 한다



먼저 총 5바이트를 입력하면 DMA가 ARM을 호출한다. 이유는 half 조건에서 걸렸기 때문이다. 그래서 다음으로 아래 2개의 함수를 지나가게 된다

```

059     * @brief This Function handles DMA2 stream global interrupt.
060
061     void DMA2_Stream0_IRQHandler(void)
062     {
063         /* USER CODE BEGIN DMA2_Stream0_IRQn 0 */
064
065         HAL_UART_RxHalfCpltCallback(huart);
066
067     /* USER CODE END DMA2_Stream0_IRQn 1 */
068
069     /* USER CODE BEGIN 1 */
070
071     /* USER CODE END 1 */
072
073     /* *****END OF FILE***** */
074
075 }

```

D. 다시 재개 후 나머지 5바이트를 입력하도록 한다

```

DMA2_Stream2_IRQHandler() at stm32f4xx_it.c:203 0x8000862
<signal handler called>() at 0xfffffff9
HAL_Delay() at stm32f4xx_hal.c:404 0x8000976
main() at main.c:124 0x80006f6

main.c stm32f4xx_hal_uart.c stm32f4xx_it.c stm32f4xx_hal_dma.c stm32f4xx_hal_dma.h
190 /* Add here the Interrupt Handlers for the used peripherals.
191 /* For the available peripheral interrupt handler names,
192 /* please refer to the startup file (startup_stm32f4xx.s).
193 *****/
194
195 /**
196 * @brief This function handles DMA2 stream2 global interrupt.
197 */
198 void DMA2_Stream2_IRQHandler(void)
199 {
200     /* USER CODE BEGIN DMA2_Stream2_IRQHandler 0 */
201
202     /* USER CODE END DMA2_Stream2_IRQHandler 0 */
203     HAL_DMA_IRQHandler(&hdma_usart1_rx);
204     /* USER CODE BEGIN DMA2_Stream2_IRQHandler 1 */
205
206     /* USER CODE END DMA2_Stream2_IRQHandler 1 */
207 }
208
209 /* USER CODE BEGIN 1 */
210
211 /* USER CODE END 1 */
212 ****/ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
213

```

Console Tasks Problems Executables Debugger Console Memory

7.dma_uart_to_sram Debug [Acc STM32 Debugging] openocd

Info : STM32F429ZITx.cpu: hardware has 6 breakpoints, 4 watchpoints

Info : Listening on port 3333 for gdb connections

Info : accepting 'gdb' connection on tcp/3333

그리고 최종적으로 rxcompletecallback 함수에 걸리게 된다

```

/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(huart);
    /* NOTE: This function Should not be modified, when the callback is needed,
       the HAL_UART_TxCpltCallback could be implemented in the user file
    */
}

```

Callback 함수에서 DMA를 재호출함으로써 계속 반복할 수 있게 된다

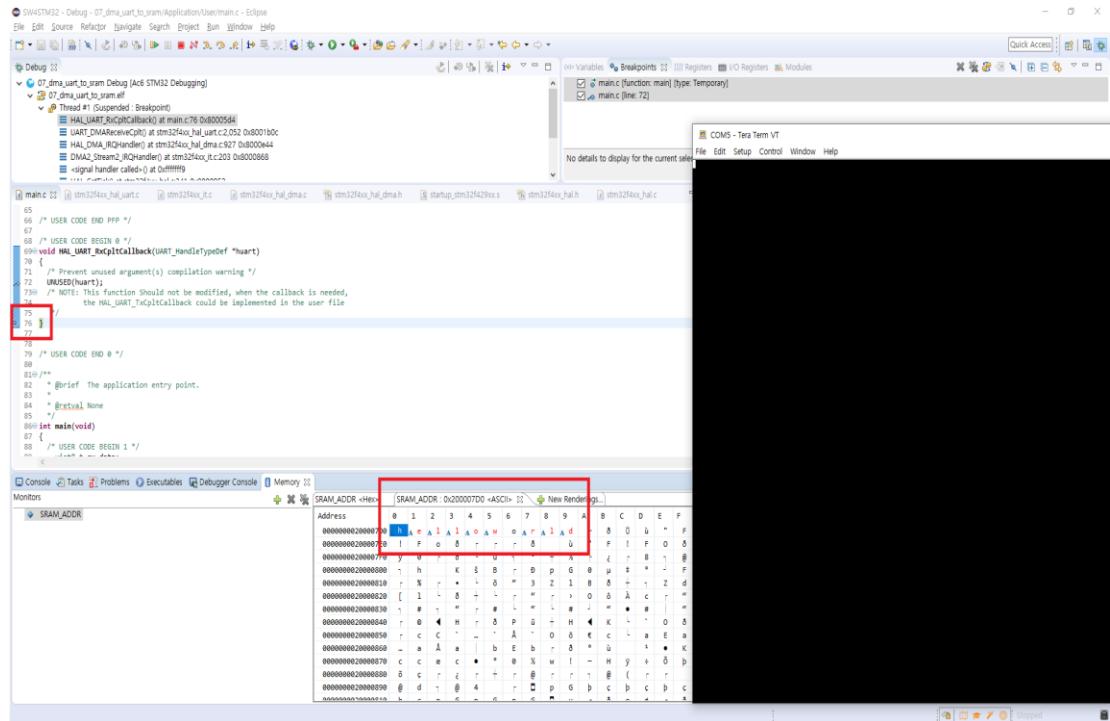
```

/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */

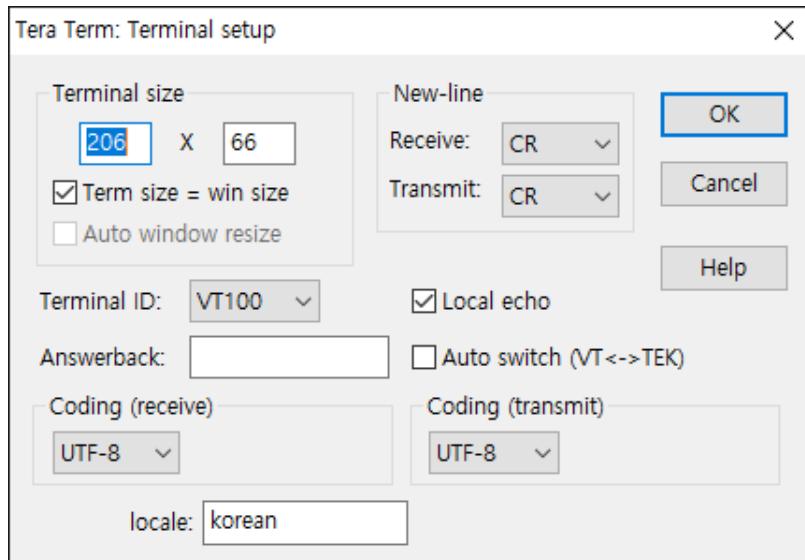
    /* NOTE: This function Should not be modified, when the callback is needed,
       the HAL_UART_TxCpltCallback could be implemented in the user file
    */
    HAL_UART_Receive_DMA(&huart1, (uint8_t*)(SRAM_ADDR), 10);
}

```

E. 다음은 쓰여진 helloworld라는 문자열이 SRAM에 저장된 것을 나타낸 사진이다



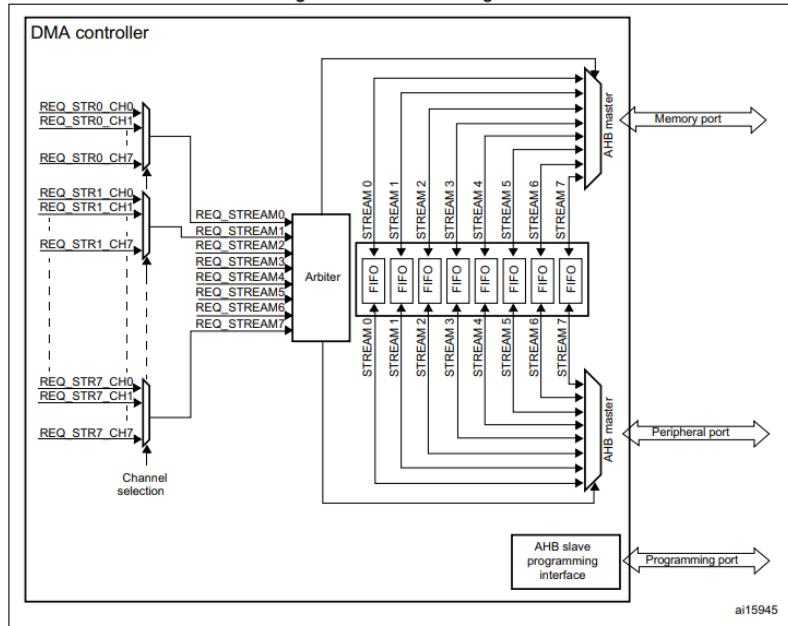
참고사항: teraterm echo 설정



3. DMA 고급 개념

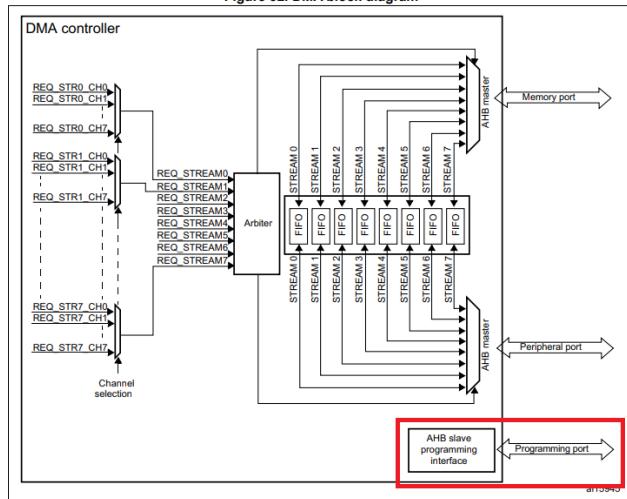
- 지금까지 DMA 기본 개념을 공부하고, M2M, P2M을 폴링, 인터럽트 방식으로 제어해봤다. 더 나아가서 DMA 설정에서 나타난 용어들에 대해서 서술해보려고 한다
- DMA는 AHB Master 모듈이면서 Stm32f429에는 2개가 존재하고 있다

Figure 32. DMA block diagram



3.1 DMA Slave Port

Figure 32. DMA block diagram

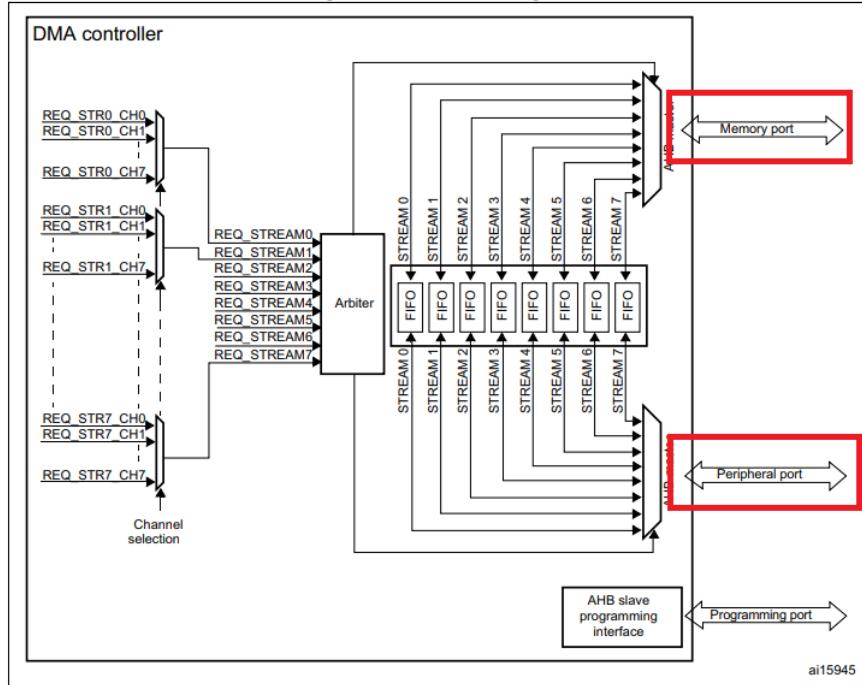


- DMA Slave Port는 source, destination 주소 설정 등 config를 위해서 있는 port다. 통신을 하진 않는다는 점을 유의해야 한다

The AHB slave port is used to program the DMA controller (it supports only 32-bit accesses).

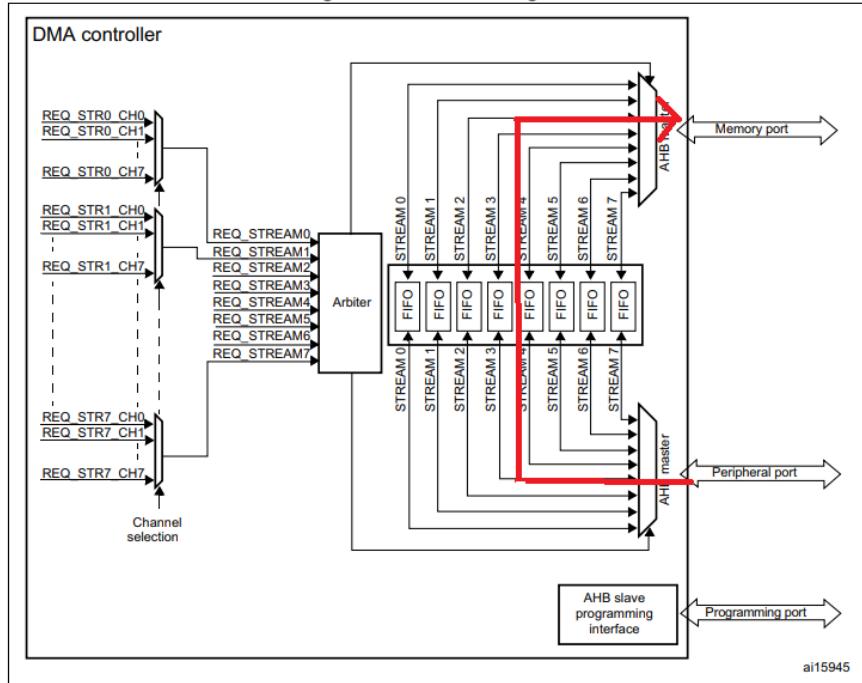
3.2 DMA peripheral and memory port

Figure 32. DMA block diagram



- 메모리, peripheral을 구분 지었는데, 메모리는 SRAM을 peripheral은 APB에 있는 여러 주변장치를 일컫는다
- 그래서 만일 UART Rx 데이터를 SRAM으로 DMA로 전달하려고 하면 다음과 같이 경로가 전달되게 된다

Figure 32. DMA block diagram



3.3 DMA Stream

- Block 디어그램에서도 확인할 수 있듯이, 각각의 포트들은 단일 방향을 가지고 있지 않고 양방향 DMA를 할 수 있다는 것이다. 즉
 - Memory to Peripheral
 - Peripheral to Memory
 - Memory to Memory

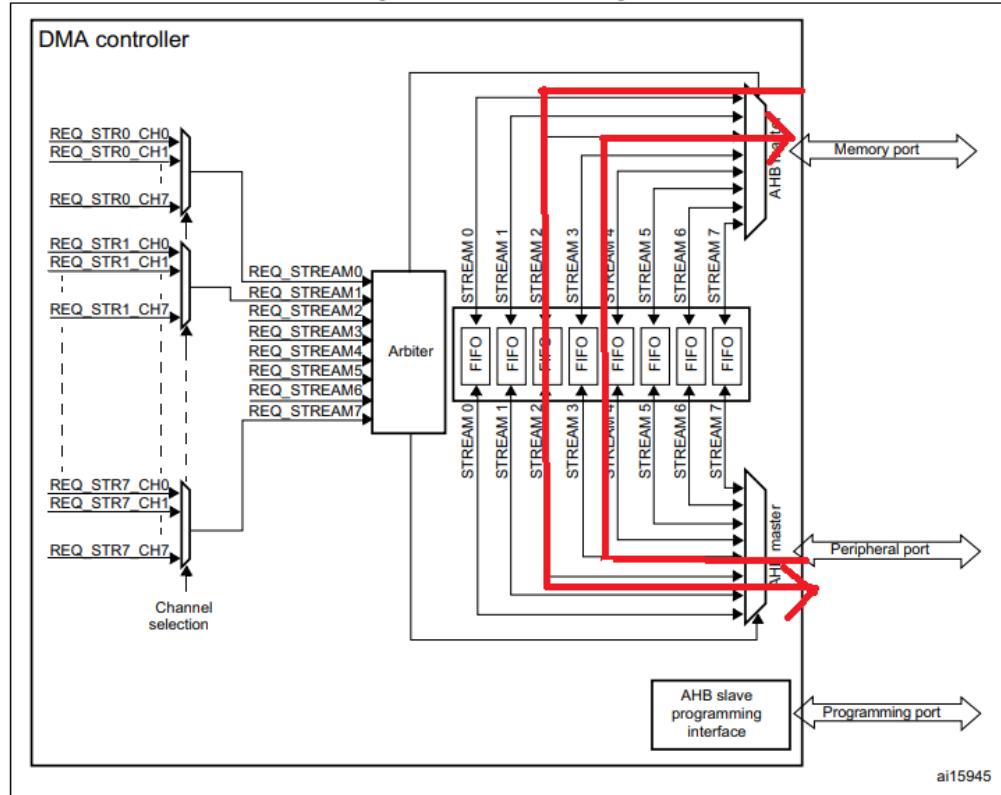
The DMA controller performs direct memory transfer: as an AHB master, it can take the control of the AHB bus matrix to initiate AHB transactions.

It can carry out the following transactions:

- peripheral-to-memory
- memory-to-peripheral
- memory-to-memory

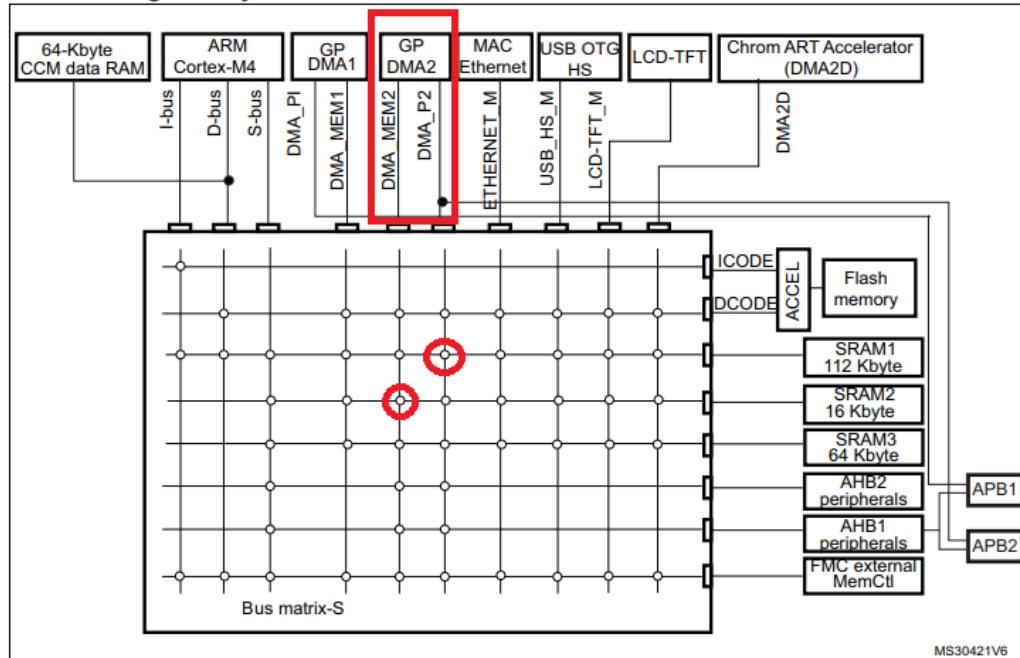
The DMA controller provides two AHB master ports: the *AHB memory port*, intended to be connected to memories and the *AHB peripheral port*, intended to be connected to peripherals. However, to allow memory-to-memory transfers, the *AHB peripheral port* must also have access to the memories.

Figure 32. DMA block diagram



- 그러면 M2M은 어떻게 하는 것일까? 결론적으로는 DMA1은 할 수 없지만, DMA2는 할 수 있다. Peripheral 포트를 Memory가 사용할 수 있게 길을 제공하고 있다

Figure 2. System architecture for STM32F42xxx and STM32F43xxx devices



Does the memory that starts from M2M have to start from the peripheral port Necessarily?

Figure 34. System implementation of the two DMA controllers (STM32F42xxx and STM32F43xxx)

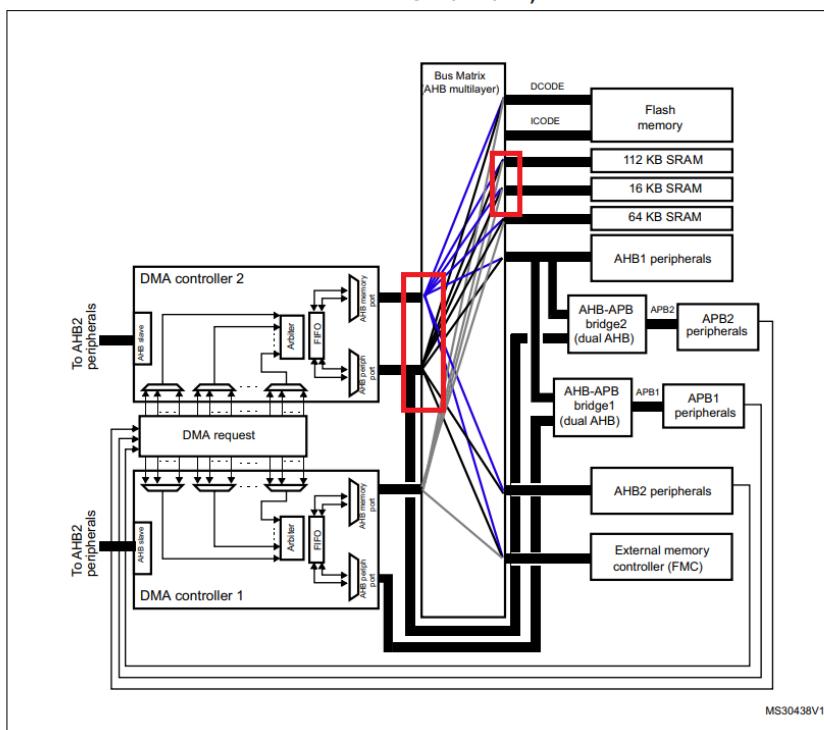
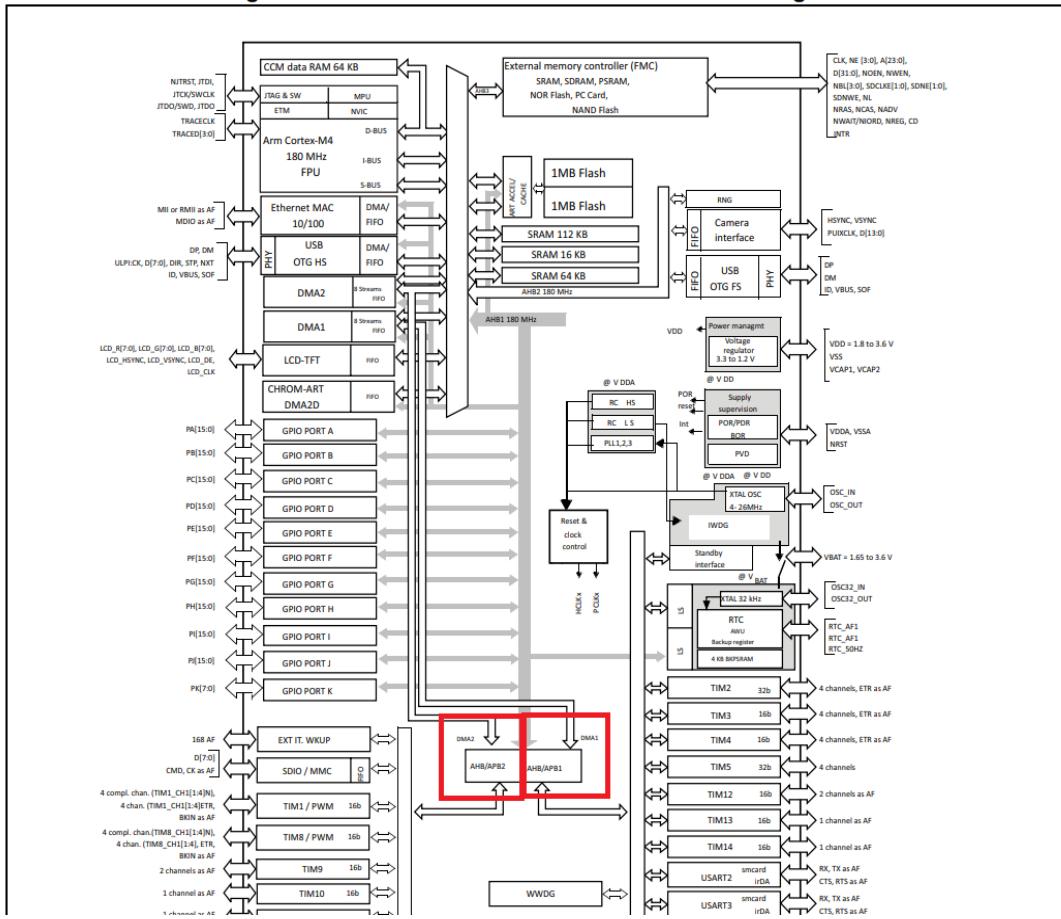


Figure 4. STM32F427xx and STM32F429xx block diagram

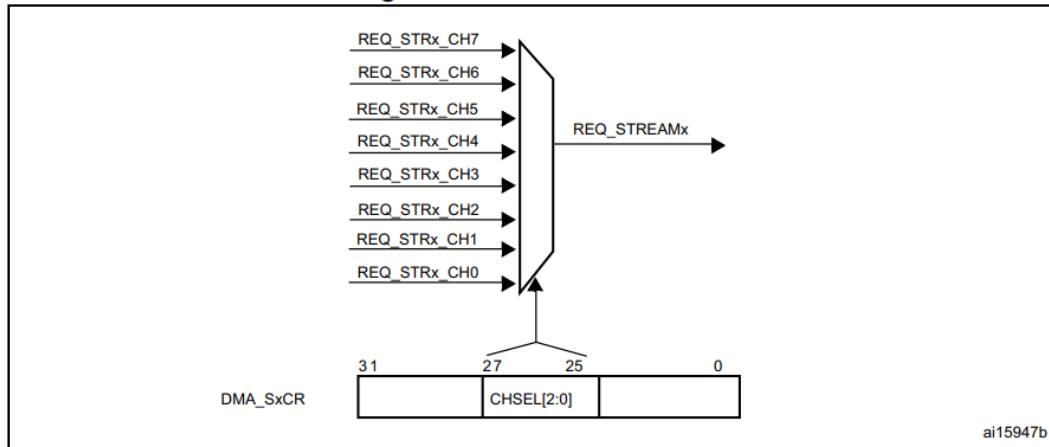


참고) AHB/APB2는 DMA2, 그리고 AHB/APB1은 DMA1에 연결되어 있다. 그리고 AHB1 Peripheral은 따로 있고 이는 DMA2에 연결되어 있다

3.4 DMA Channel

- 각각의 stream들은 8개의 DMA request와 연결되어 있다. 각 request들은 다양한 peripheral로 연결되어 있다

Figure 35. Channel selection



The 8 requests from the peripherals (TIM, ADC, SPI, I2C, etc.) are independently connected to each channel and their connection depends on the product implementation.

- 예를 들어 DMA2에 대한 request mapping이다. **SRAM, Flash와 같은 memory 디바이스는 정해진 stream 번호가 없다**

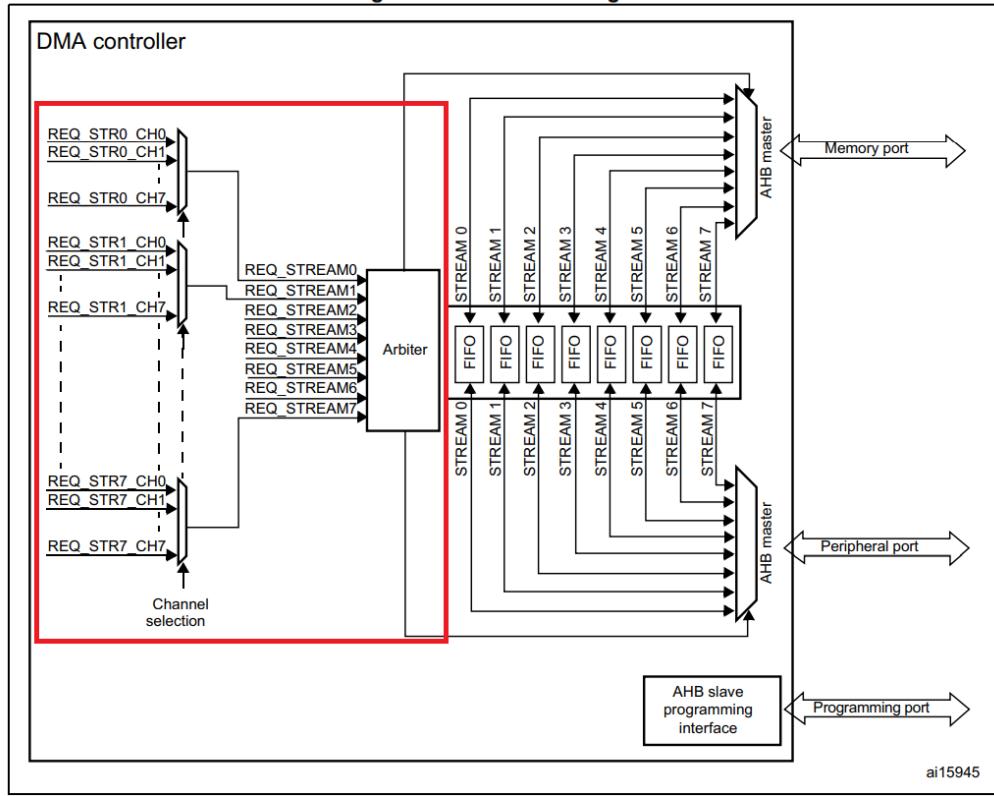
Table 43. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A ⁽¹⁾	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A ⁽¹⁾	ADC1	SAI1_B ⁽¹⁾	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B ⁽¹⁾	SPI6_TX ⁽¹⁾	SPI6_RX ⁽¹⁾	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX ⁽¹⁾	SPI5_TX ⁽¹⁾	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX ⁽¹⁾	SPI4_TX ⁽¹⁾	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX ⁽¹⁾	SPI4_TX ⁽¹⁾	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX ⁽¹⁾	SPI5_TX ⁽¹⁾	TIM8_CH4 TIM8_TRIG TIM8_COM

1. These requests are available on STM32F42xxx and STM32F43xxx.

- 전체 stream에 대한 request map이다

Figure 32. DMA block diagram



- 그리고 하나의 Peripheral은 하나의 stream으로만 연결되어 있는 것은 아니다

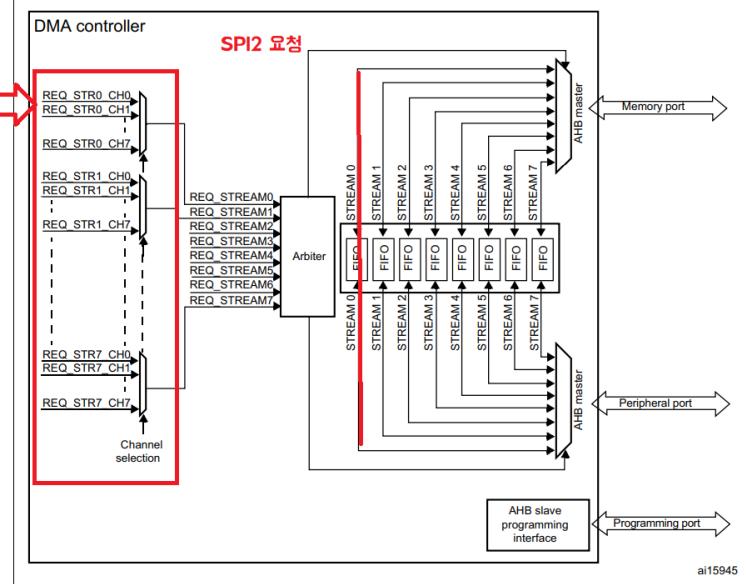
Table 42. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX
Channel 1	I2C1_RX	-	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX ⁽¹⁾	UART7_TX ⁽¹⁾	TIM3_CH4 TIM3_UP	UART7_RX ⁽¹⁾	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX ⁽¹⁾	TIM3_CH3

DMA Channel Mapping은 다음과 같이 한다

1) P2M

Figure 32. DMA block diagram

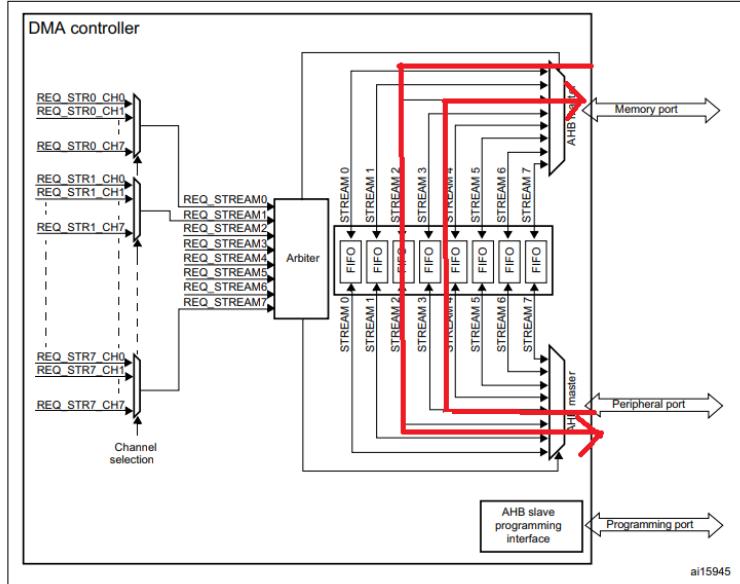


만약에 Peripheral이 DMA에게 STREAM request를 보내면 Arbiter는 해당 스트림을 열어 주게 된다

2) M2M

Arbiter쪽을 사용할 필요가 없고 M2M은 아무 Stream을 사용하면 된다

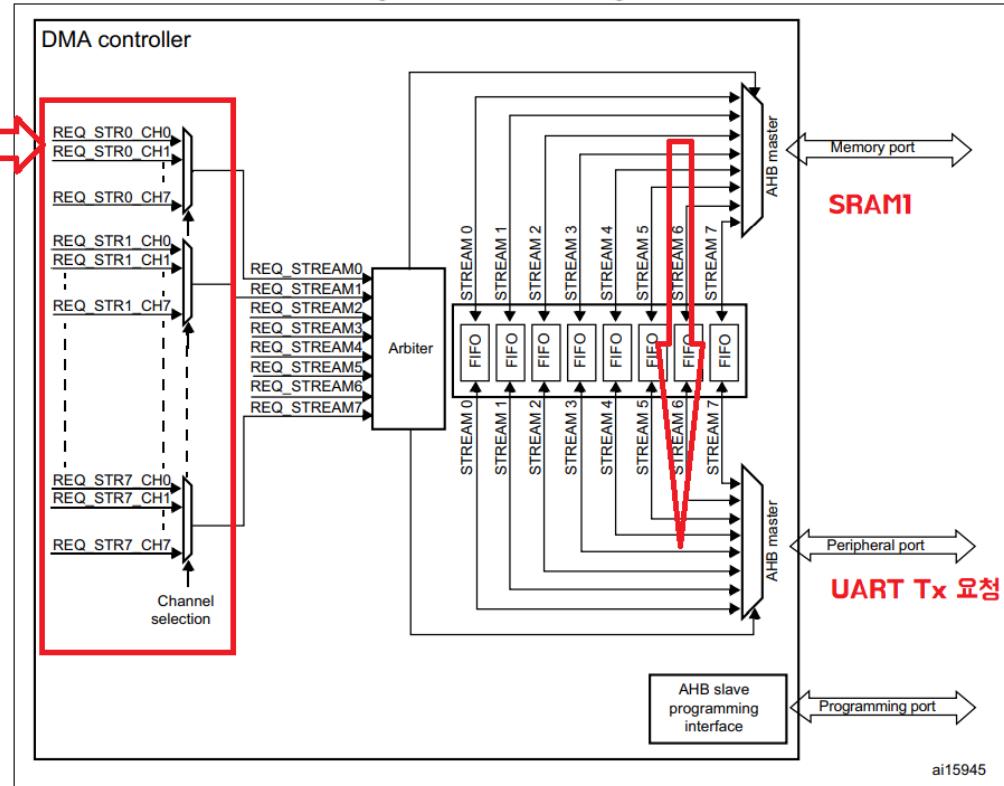
Figure 32. DMA block diagram



3) M2P

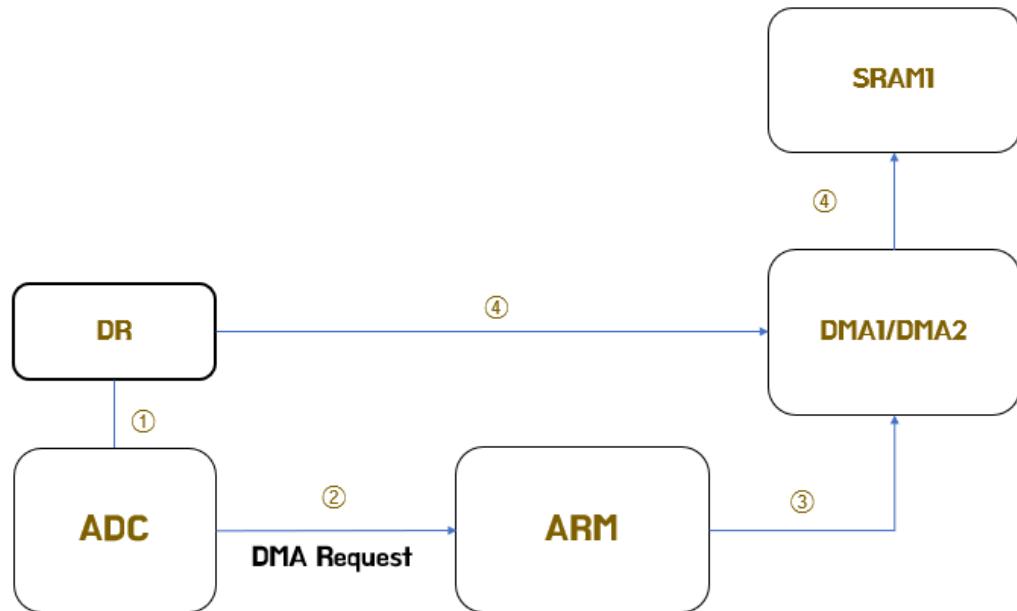
M2P 또한 결국엔 Peripheral로 전달하는 것이기 때문에 반드시 stream과 채널을 정해져서 내보내야 한다

Figure 32. DMA block diagram



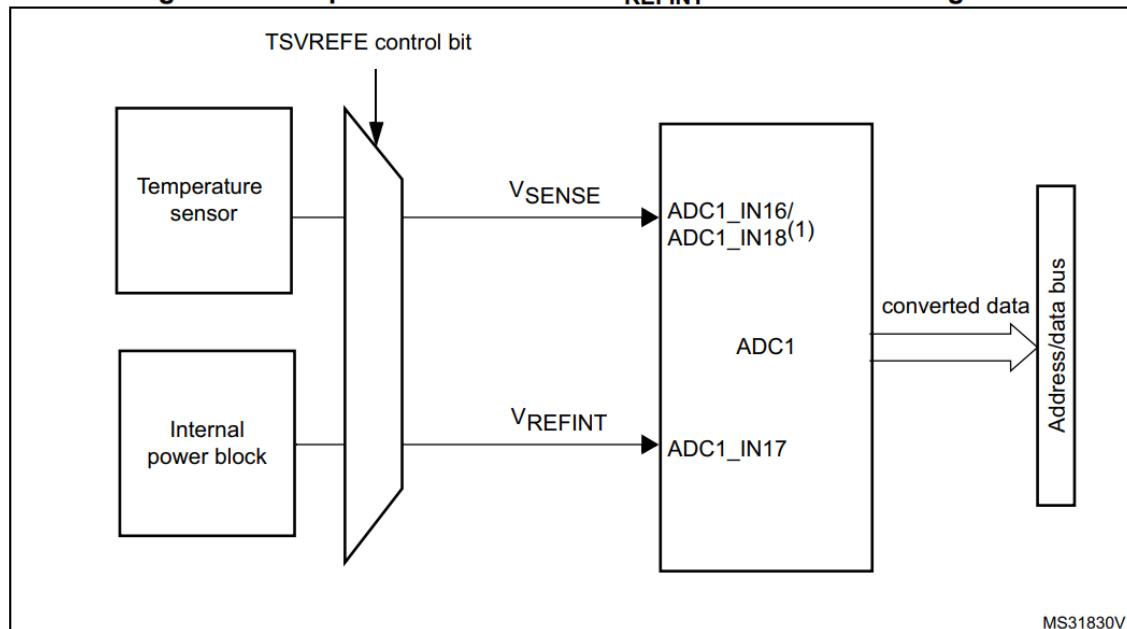
4. DMA 실습

- ADC에 읽은 데이터를 SRAM에 작성하는 P2M 실습을 해보려고 한다. ADC는 변환을 완료하면 DMA에게 Request를 알리고, SRAM에 저장하게 된다



- ADC 대상은 MCU 안에 있는 온도 센서를 활용하려고 한다

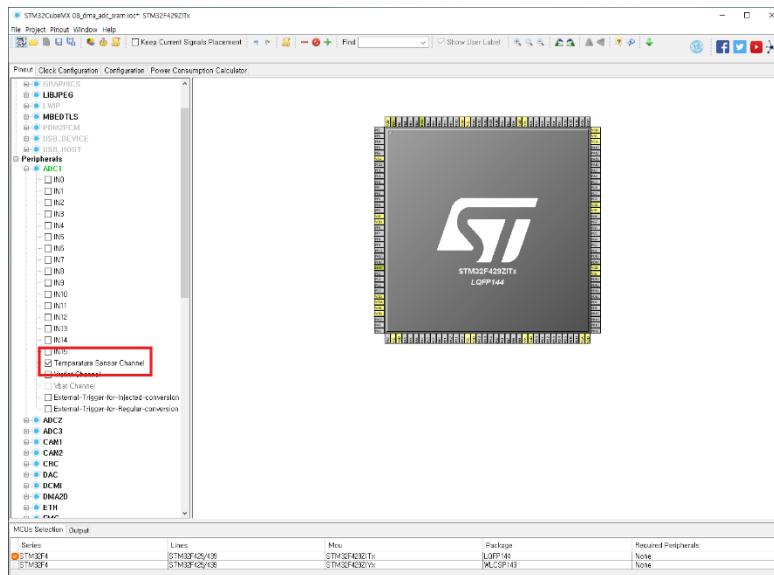
Figure 63. Temperature sensor and V_{REFINT} channel block diagram



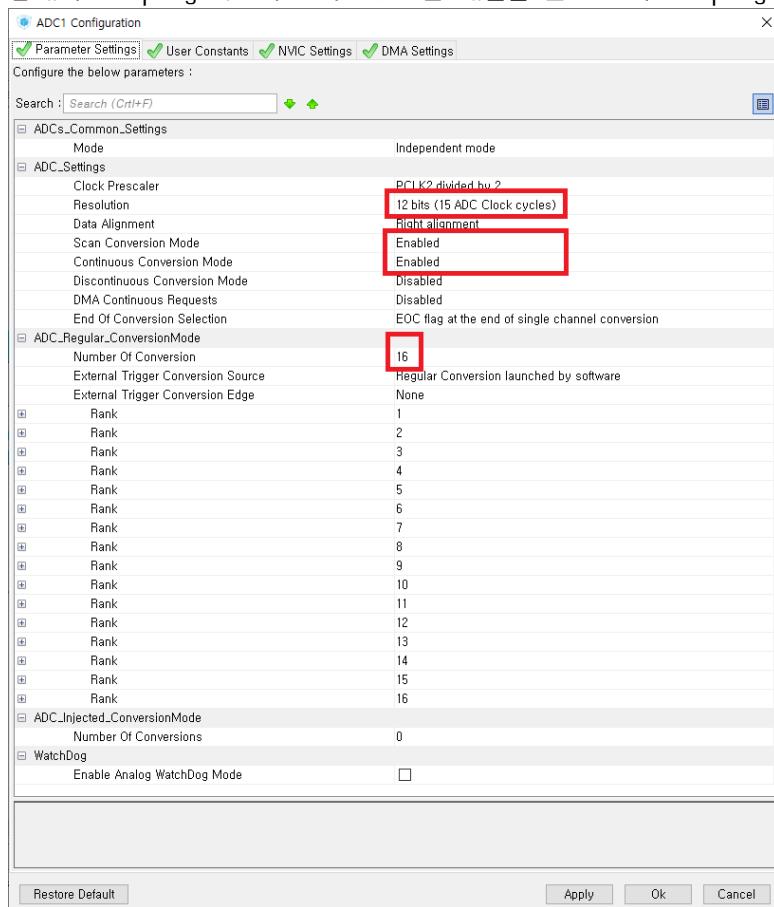
1. V_{SENSE} is input to ADC1_IN16 for the STM32F40x and STM32F41x devices and to ADC1_IN18 for the STM32F42x and STM32F43x devices.

4.1 CubeMX 설정

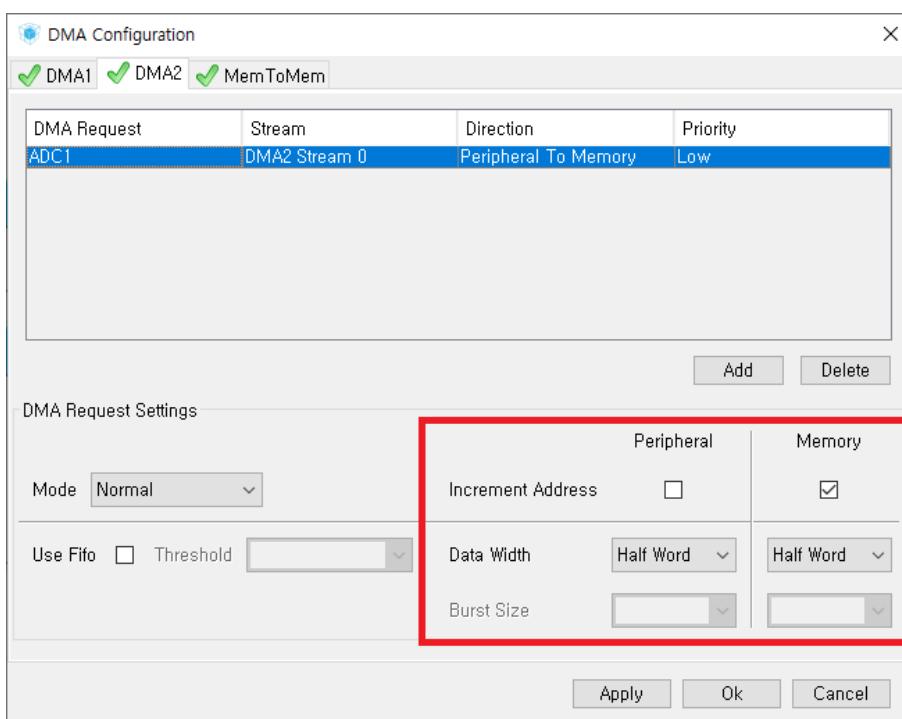
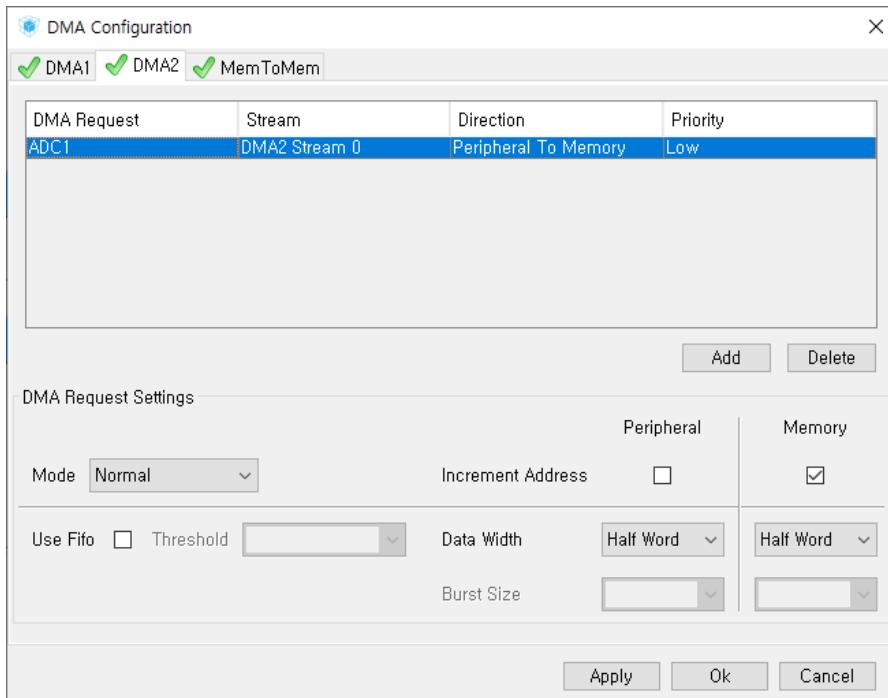
- ADC1에 있는 온도센서를 입력으로 받도록 한다



- 그리고 12bit resolution에 scan, continuous 설정까지 마치도록 한다. 그리고 16개의 Rank를 활용해서 sampling 하도록 한다. 즉 모든 채널을 온도 센서 sampling으로 활용하겠다는 것이다

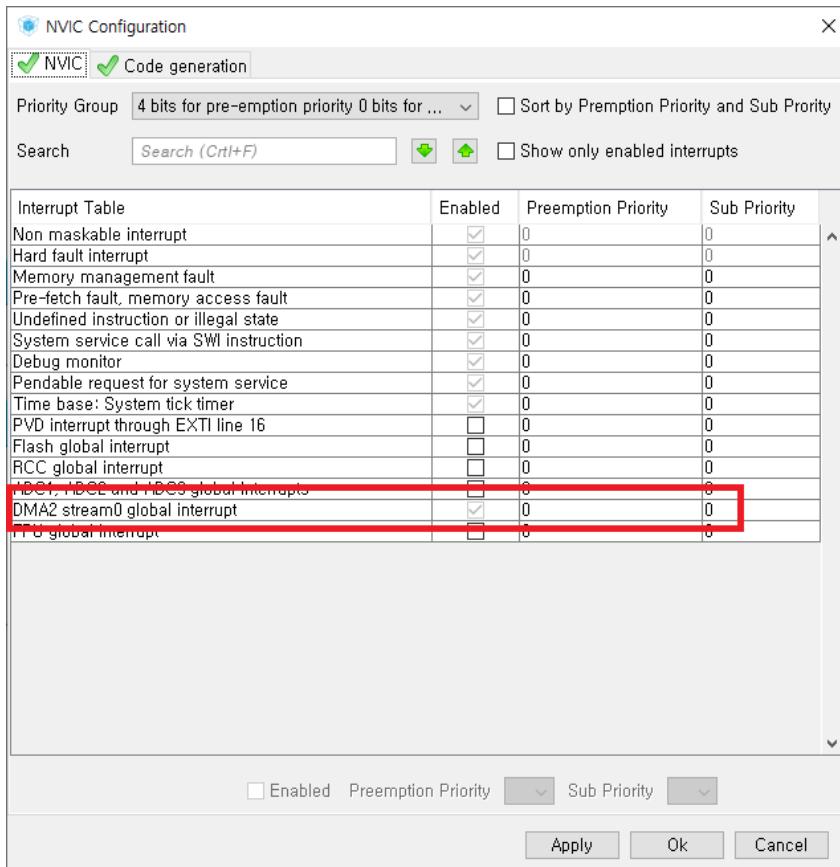


- 그리고 DMA로 설정하는데, P2M이기 때문에 Peripheral에 맞는 스트림과 채널이 정해져 있다

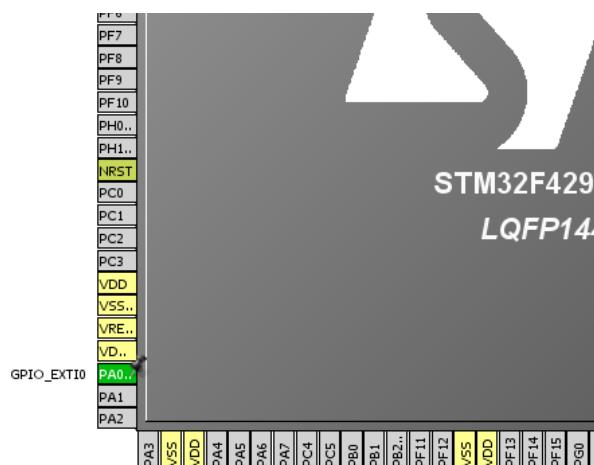


- 그리고 Memory에만 Increment가 되어있는 이유는 ADC 데이터 레지스터는 정해져 있고, SRAM은 그 값을 저장하기 위해서 자동적으로 증가해야 한다
- 그리고 12비트 온도 센서 값이기 때문에 data는 2byte = word로 설정했다

- 자동적으로 DMA NVIC이 설정된 것을 확인할 수 있다



- 추가로 버튼을 눌렀을 때 ADC 연산을 시작하기 위해서 버튼 인터럽트도 추가하도록 한다



4.2 프로그램 코드 작성

- HAL_ADC 드라이버 소스로 가면 다음과 같이 DMA Start 함수가 있다. 이를 버튼 인터럽트에 넣어서 버튼이 눌렸을 때마다 실행하도록 한다

```

191 // Includes -----
192 #include "stm32f4xx_hal.h"
193 #include "stm32f4xx_hal_adc.h"
194
195 /** @defgroup STM32F4xx_HAL_Driver
196 * @brief ADC driver modules
197 */
198
199 /** @defgroup ADC ADC
200 * @brief ADC driver modules
201 */
202
203
204 #ifndef HAL_ADC_MODULE_ENABLED
205
206 /* Private typedef -----*/
207 /* Private define -----*/
208 /* Private macro -----*/
209 /* Private variables -----*/
210 /** @defgroup ADC_Private_Functions
211 */
212
213 /* Private function prototypes -----*/
214 static void ADC_Init(ADC_HandleTypeDef* hadc);
215 static void ADC_DMAConvCplt(ADC_HandleTypeDef* hdma);
216 static void ADC_DMAError(ADC_HandleTypeDef* hdma);
217 static void ADC_DMAHalfConvCplt(ADC_HandleTypeDef* hdma);
218 */
219 */
220
221 /* Exported functions -----*/
222 /** @defgroup ADC_Exported_Functions ADC Exported Functions
223 */
224
225
226 /** @defgroup ADC_Exported_Functions_Group1 Initialization and de-initialization functions
227 * @brief Initialization and Configuration functions
228 */

```

```

/* USER CODE BEGIN 0 */
uint16_t temp_data[16];

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)temp_data, 16);
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
}

/* USER CODE END 0 */

```

temp_data는 전역 변수로 런타임 때 SRAM에 위치한다. 그리고 DMA가 완료되면 HAL_ADC_ConvCpltCallback 함수가 불린다. 자세한 내용은 Start_DMA 함수에 적혀 있다

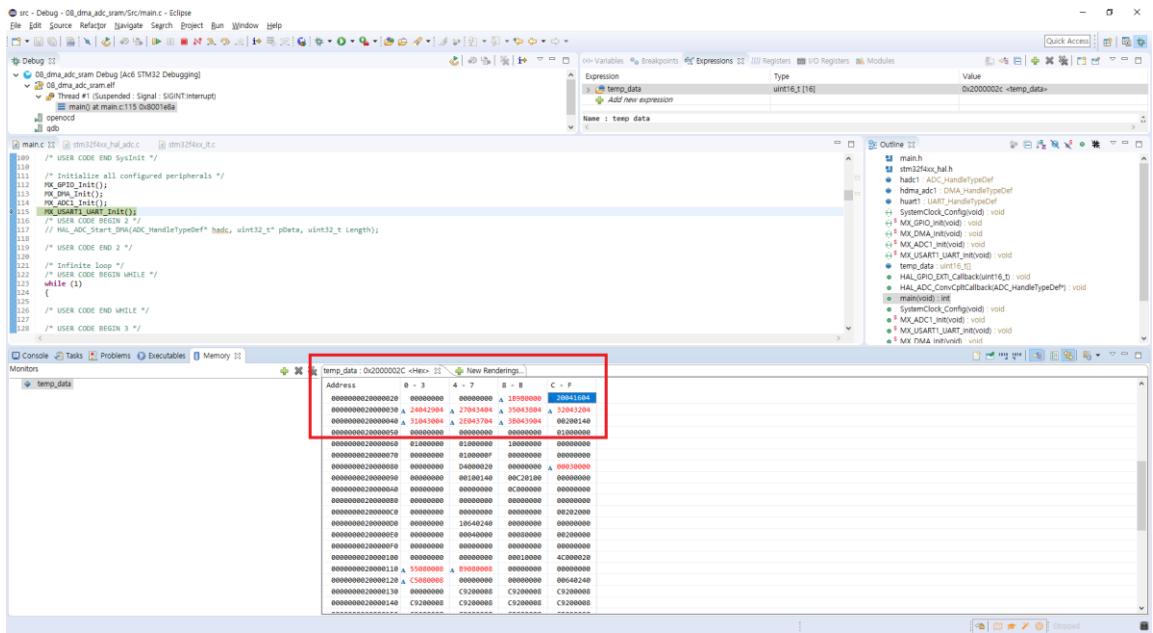
```

/* Set the DMA transfer complete callback */
hadc->DMA_Handle->XferCpltCallback = ADC_DMAConvCplt;

/* Set the DMA half transfer complete callback */
hadc->DMA_Handle->XferHalfCpltCallback = ADC_DMAHalfConvCplt;
|
/* Set the DMA error callback */
hadc->DMA_Handle->XferErrorCallback = ADC_DMAError;

```

- 디버깅 시 버튼을 눌렀을 때 값이 잘 들어간 것을 확인할 수 있다. 총 2바이트로 32바이트가 순차적으로 쓰여지는 것을 볼 수 있다



- 받은 값을 가지고 온도 계산을 해보도록 하자

8. Calculate the temperature using the following formula:

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{\text{SENSE}} - V_{25}) / \text{Avg_Slope}\} + 25$$

Where:

- V_{25} = V_{SENSE} value for 25°C
- Avg_Slope = average slope of the temperature vs. V_{SENSE} curve (given in $\text{mV}/^\circ\text{C}$ or $\mu\text{V}/^\circ\text{C}$)

Refer to the datasheet's electrical characteristics section for the actual values of V_{25} and Avg_Slope.

6.3.22 Temperature sensor characteristics

Table 80. Temperature sensor characteristics

Symbol	Parameter	Min	Typ	Max	Unit
$T_L^{(1)}$	V_{SENSE} linearity with temperature	-	± 1	± 2	$^\circ\text{C}$
Avg_Slope ⁽¹⁾	Average slope	-	2.5		$\text{mV}/^\circ\text{C}$
$V_{25}^{(1)}$	Voltage at 25°C	-	0.76		V
$t_{\text{START}}^{(2)}$	Startup time	-	6	10	μs
$T_{\text{S_temp}}^{(2)}$	ADC sampling time when reading the temperature (1 $^\circ\text{C}$ accuracy)	10	-	-	μs

1. Guaranteed by characterization results.

2. Guaranteed by design.

- 총 12비트이고, VDD=Vref는 3.3V다. 그리고 측정값 중 하나는 1074이다. 이를 계산하면 25.x값이 나온다



- 그래서 다음과 같이 코드를 작성할 수 있다

```

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

    /* USER CODE BEGIN 3 */
        __disable_irq();
        if(dma_finish) {
            dma_finish = 0;
            __enable_irq();

            float temp_sum = 0;
            for(int i=0; i<12; i++) {
                temp_sum += temp_data[i];
            }
            temp_sum = temp_sum / 12.0;

            float temp = (((temp_sum * (3.3/(2^12)-1))-0.76)/2.5)+.25;
            // print(temp);
        }

        __WFI();
    /* USER CODE END 3 */
}

```

- Float 연산이 들어가 있어서 FPU가 없이는 부담되는 연산이다