6. DMA- Acceso Directo a Memoria.

El DMA es un módulo controlador que se usa para hacer transferencia de datos entre periféricos y memoria y también entre localidades de memoria sin necesidad de intervención de la CPU. En el microcontrolador STM32F103CT6 hay un DMA con siete canales y soporta periféricos como temporizadores, ADC, SPI, I2C y USART. El DMA es un módulo que comparte el bus del sistema con el CPU y hace que el movimiento de información sea más rápido entre periféricos y memoria aprovechando los espacios de transición en la CPU.

La forma como funciona el DMA es que, después de un evento, el periférico envía una señal de pedido al controlador de DMA. El DMA responde, de acuerdo a la prioridad del canal. Para manejar la prioridad del canal, existe un árbitro. Hay cuatro niveles de prioridad: Prioridad muy alta, alta, media y baja. Si hay dos requerimientos con igual prioridad, entonces la prioridad la define el número del canal, siendo el canal con el número menor el más prioritario.

Tabla 1. Ubicación de los canales de DMA para los periféricos.

Periféricos	Canal 1	Canal 2	Canal 3	Canal 4	Canal 5	Canal 6	Canal 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1	-	TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-

Para programar el DMA, cada canal se puede controlar con cuatro registros: Dirección de memoria, dirección del periférico, número de datos y configuración. Todos los canales tienen dos registros dedicados: Registro de estado de interrupción DMA y el registro de borrado de banderas de interrupción. Una vez configurado el DMA, este se encarga del incremento de las direcciones sin afectar la CPU. Los canales del DMA pueden generar tres interrupciones: Transferencia terminada, Transferencia medio terminada y Error de transferencia.

Las direcciones de los periféricos son las siguientes:

ADC1: `ADC1->DR`
SPI1: `SPI1->DR`
SPI2: `SPI2->DR`
I2S2: `SPI2->DR`
Usart1: `USART1->DR`
Usart2: `USART2->DR`

Usart3: `USART3->DR`
I2C1: `I2C1->DR`
I2C2: `I2C2->DR`
TIM1: `TIM1->CNT`
TIM2: `TIM2->CNT`
TIM3: `TIM3->CNT`

La cantidad de datos que se transfieren desde el periférico a la memoria es programable y puede ser hasta de 65.536 datos. Para usar el DMA se debe activar el reloj con la función `RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE).` Enseguida se hace la configuración del DMA mediante la inicialización de una estructura de tipo `DMA_InitTypeDef` que tiene los siguientes parámetros:

- `DMA_PeripheralBaseAddr`: Dirección del periférico.
- `DMA_MemoryBaseAddr`: Dirección de memoria.
- `DMA_DIR`: Indica si el periférico es el destino o la fuente. Puede ser `DMA_DIR_PeripheralDST` o `DMA_DIR_PeripheralSRC`, respectivamente.
- `DMA_BufferSize`: Tamaño de la memoria.
- `DMA_PeripheralInc`: Indica si la dirección del periférico se debe incrementar o no. Puede ser `DMA_PeripheralInc_Enable` o `DMA_PeripheralInc_Disable`.
- `DMA_MemoryInc`: Indica si la dirección de memoria se debe incrementar o no. Puede ser `DMA_MemoryInc_Enable` o `DMA_MemoryInc_Disable`.
- `DMA_PeripheralDataSize`: Tamaño del dato del periférico. Puede ser `DMA_PeripheralDataSize_Byte`, `DMA_PeripheralDataSize_HalfWord`, o `DMA_PeripheralDataSize_Word`.
- `DMA_MemoryDataSize`: Tamaño del dato de la memoria. Puede ser `DMA_MemoryDataSize_Byte`, `DMA_MemoryDataSize_HalfWord`, o `DMA_MemoryDataSize_Word`.
- `DMA_Mode`: Modo de operación. Puede ser `DMA_Mode_Circular` o `DMA_Mode_Normal`.
- `DMA_Priority`: Prioridad del canal. Puede ser `DMA_Priority_VeryHigh`, `DMA_Priority_High`, `DMA_Priority_Medium`, o `DMA_Priority_Low`.
- `DMA_M2M`: Indica si el DMA se va a usar para transferencias entre memorias. Puede ser `DMA_M2M_Enable` o `DMA_M2M_Disable`.

La inicialización de esta estructura se hace con la función `DMA_StructInit(&DMA_InitStruct)` que le asigna los siguientes valores:

- `DMA_PeripheralBaseAddr = 0;`
- `DMA_MemoryBaseAddr = 0;`
- `DMA_DIR = DMA_DIR_PeripheralSRC;`
- `DMA_BufferSize = 0;`
- `DMA_PeripheralInc = DMA_PeripheralInc_Disable;`
- `DMA_MemoryInc = DMA_MemoryInc_Disable;`
- `DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;`
- `DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;`
- `DMA_Mode = DMA_Mode_Normal;`
- `DMA_Priority = DMA_Priority_Low;`

- `DMA_M2M = DMA_M2M_Disable;`

Luego se usa la función de configuración del `DMA, DMA_Init(DMA1_Channelx, &DMA_InitStruct)`, indicando el canal 'x' de 1 a 7 que se va a usar, según el periférico, como lo describe la tabla anterior. Una vez configurado el DMA, se debe habilitar con la función `DMA_Cmd(DMA1_Channelx, ENABLE)`. Después de esto el DMA está listo para atender el requerimiento del periférico. Una vez se presente un evento, el contador de unidades se empieza a decrementar, por lo que cuando un siguiente evento lo requiera, es necesario actualizar el número de datos que se desea almacenar en memoria con la función `DMA_SetCurrDataCounter(DMA1_Channelx, uint16_t DataNumber)`.

El siguiente es un ejemplo de DMA para transferencia de datos entre memoria a memoria. Primero se inicializa un bloque de memoria fuente de 800 datos, con valores desde 0 hasta 799. También se inicializa un bloque de memoria destino con ceros. Después de la configuración y habilitación del DMA, se muestra el contenido del bloque destino.

```
#define ARRAYSIZE 800
volatile uint32_t i;
int main(void)
{
    Sysclk_56M();
    UART_Init();
    LED_Init();

    //inicializa arreglos de fuente y destino
    uint32_t source[ARRAYSIZE];
    uint32_t destination[ARRAYSIZE];
    //inicializa arreglo de la fuente
    for (i=0; i<ARRAYSIZE;i++)
    {
        source[i]=i;
        destination[i]= 0;
        UART_numero(destination[i]);// muestra contenido inicial del destino
    }
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    USART_SendData(USART1,10);

    // activación del reloj del DMA
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    // crea una estructura para DMA
    DMA_InitTypeDef  DMA_InitStruct;
    //reset el canal channel1 del DMA1 a sus valores de inicio
    DMA_DeInit(DMA1_Channel1);
    // este canal se va a usar para transferencia desde memoria a memoria
    DMA_InitStruct.DMA_M2M = DMA_M2M_Enable;
    //selección de modo normal (no circular)
    DMA_InitStruct.DMA_Mode = DMA_Mode_Normal;
    //prioridad media
    DMA_InitStruct.DMA_Priority = DMA_Priority_Medium;
    //tamaño del dato de la fuente y el destino= 32bit
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
```

```
    //habilitación de incremento automático en fuente y destino
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
    //La posición asignada al registro periférico será la fuente
    DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;
    //tamaño de los datos que se transfieren
    DMA_InitStruct.DMA_BufferSize = ARRAYSIZE;
    //dirección de inicio de la fuente y el destino
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)source;
    DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)destination;
    //programa registros del DMA
    DMA_Init(DMA1_Channel1, &DMA_InitStruct);
    //habilita transferencia en el canal de DMA1
    DMA_Cmd(DMA1_Channel1, ENABLE);

  while(1)
  {
      for(i=0; i<ARRAYSIZE;i++)
      {
          UART_numero(destination[i]);
      }
      while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
      USART_SendData(USART1,10);
      while(1);
  }
}
```

En el siguiente ejemplo se muestra lo mismo del ejemplo pasado, pero con interrupción. La interrupción se hace cuando la transferencia está completa. En ese momento se enciende el led. Dentro del while(1) se observa por el encendido del led para poder mostrar como queda la memoria destino.

```
#define ARRAYSIZE 800
volatile uint32_t i;

int main(void)
{
    Sysclk_56M();
    UART_Init();
    LED_Init();

    //inicializa arreglos de fuente y destino
    uint32_t source[ARRAYSIZE];
    uint32_t destination[ARRAYSIZE];
    //inicializa arreglo de la fuente
    for (i=0; i<ARRAYSIZE;i++)
    {
        source[i]=i;
        destination[i]= 0;
        UART_numero(destination[i]);
    }
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    USART_SendData(USART1,10);

   // activación del reloj del DMA
```

```
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
   // crea una estructura para DMA
   DMA_InitTypeDef  DMA_InitStruct;
   //reset el canal channel1 del DMA1 a sus valores de inicio
   DMA_DeInit(DMA1_Channel1);
   // este canal se va a usar para transferencia desde memoria a memoria
   DMA_InitStruct.DMA_M2M = DMA_M2M_Enable;
   //selección de modo normal (no circular)
   DMA_InitStruct.DMA_Mode = DMA_Mode_Normal;
   //prioridad media
   DMA_InitStruct.DMA_Priority = DMA_Priority_Medium;
   //tamaño del dato de la fuente y el destino= 32bit
   DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
   DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
   //habilitación de incremento automático en fuente y destino
   DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
   DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
   //La posición asignada al registro periférico será la fuente
   DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;
   //tamaño de los datos que se transfieren
   DMA_InitStruct.DMA_BufferSize = ARRAYSIZE;
   //dirección de inicio de la fuente y el destino
   DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)source;
   DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)destination;
   //programa registros del DMA
   DMA_Init(DMA1_Channel1, &DMA_InitStruct);

   // habilita la interrupción por transferencia completa en el DMA1
   DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);
   // programación de interrupción
   NVIC_InitTypeDef NVIC_InitStruct;
   NVIC_InitStruct.NVIC_IRQChannel = DMA1_Channel1_IRQn;
   NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStruct);

   //habilita transferencia en el canal de DMA1
   DMA_Cmd(DMA1_Channel1, ENABLE);

    while(1)
    {
        if(GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_12) == Bit_RESET)
        {
            for(i=0; i<ARRAYSIZE;i++)
            {
                UART_numero(destination[i]);
            }
            while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
            USART_SendData(USART1,10);
            /// apaga el led pb12
            GPIO_SetBits(GPIOB, GPIO_Pin_12);
        }
    }
}

void DMA1_Channel1_IRQHandler(void)
```

```
{
  //verifica que la interrupción se hizo con la terminación completa de la transferencia
  if(DMA_GetITStatus(DMA1_IT_TC1))
  {
      GPIO_ResetBits(GPIOB, GPIO_Pin_12);// enciende el led
      // borra la bandera de interrupción
      DMA_ClearITPendingBit(DMA1_IT_GL1);
  }
  return;
}
```

En la librería se pueden ver las funciones relacionadas con DMA.

```
1.  void DMA_DeInit(DMA_Channel_TypeDef* DMAy_Channelx);
2.  void DMA_Init(DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct);
3.  void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct);
4.  void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState);
5.  void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT, FunctionalState NewState);
6.  void DMA_SetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx, uint16_t DataNumber);
7.  uint16_t DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx);
8.  FlagStatus DMA_GetFlagStatus(uint32_t DMAy_FLAG);
9.  void DMA_ClearFlag(uint32_t DMAy_FLAG);
10. ITStatus DMA_GetITStatus(uint32_t DMAy_IT);
11. void DMA_ClearITPendingBit(uint32_t DMAy_IT);
```

La función 1 es para desinicializar una inicialización previa de un canal del DMA.
La función 2, para inicializar un canal del DMA con una estructura con información de la transferencia.
La función 3 es para inicializar una estructura con datos por defecto.
La función 4 es para habilitar un canal del DMA. También sirve para deshabilitarlo.
La función 5 es para configurar la interrupción por DMA en un canal específico.
La función 6 pone el contador del DMA con el número de datos que se van a transferir.
La función 7 lee el valor del contador de un canal del DMA.
La función 8 lee las banderas del DMA.
La función 9 borra las banderas del DMA.
La función 10 devuelve el estado de las banderas de interrupción del DMA.
La función 11 borra las banderas de interrupción del DMA.